The Goulden-Jackson cluster method

Dimitrios Noulas

February 21, 2021

Για τον αδερφό μου Θωμά και
την γιαγιά μου Αικατερίνη

Prologue

The current dissertation was undertaken in order to fullfil the graduation requirements in the Department of Informatics at the University of Piraeus, as well as to expand my own mathematical knowledge. It's scope is to research about a known theorem in combinatorics, the ideas behind it and how it can be applied to solve different kinds of enumeration problems. This research is also complemented by python scripts at the last sections which help solve some of the problems with the use of this theorem.

At this point, I would like to thank my supervisor, Prof. Aristidis Sapounakis for giving me the opportunity to undertake this dissertation based on an interesting research question. I would also like to express my gratitude towards Prof. Ioannis Tasoulas for all of his help, the research material he provided me as well as his guidance and all the countless hours we spent cooperating or just discussing about mathematics.

Introduction

The focus of this dissertation is the efficient enumeration of words with certain subwords based on the Goulden-Jackson cluster theorem.

In the first chapter, some preliminary notions are given from combinatorics and graph theory and different mathematical techniques are demonstrated on words.

The second chapter is split in two cases, the enumeration of a single pattern and the enumeration of multiple patterns simultaneously. For both cases, words as mathematical objects are extended to the notion of marked words from which a special case arises, the clusters. In more depth, it can be seen in both cases how the use of generating functions help achieve the intended enumeration, however, deriving a generating function for words is not always straightforward. This is where the necessary connections are made between the generating function of words and the generating function of marked words, thus demonstrating the inclusion-exclusion approach. Furthermore, the derivation of the second is proven to be dependent on the generating function of clusters, hence reducing the original problem to a straightforward one.

In the last chapter, different kinds of applications of the Goulden Jackson cluster method take place. The first kind is how problems of enumeration unrelated, at first, to words can be solved using this method. The second application is on self-avoiding walks which are related to physics. The method here is used to count a certain type of patterns which turn to overcount self-avoiding walks, thus creating an upper bound for a known constant. The final application is on biology, where the method is used to provide theoretical statistics for words such that biologists can compare them to the actual statistics on words from genomes.

Finally, Python and Sage scripts complement this dissertation, on which most of the applications in the 3rd chapter are based on.

Contents

Chapter 1

Preliminaries

1.1 Patterns in words

Let L be a finite alphabet and let L^* be the set of all words using letters from the alphabet L (including the *empty word, i.e. the word having no letters, denoted by ε). Now, let w, τ be some words in L^*.*

Definition 1.1.1 (Occurrences of patterns)**.** *We say that the word τ, called in this context* pattern *or* string, occurs *in w iff there exist $v, u \in L^*$, such that*

$$w = v\tau u,$$

equivalently if $w = w_1 w_2 \cdots w_n$ and $\tau = \tau_1\tau_2 \cdots \tau_k$ then there exists $j \in [n-k]$ such that

$$w_{j-1+i} = \tau_i \textit{ for all } i \in [k],$$

in this case we say that there is an occurrence of τ, *or* τ occurs at position j in w. *The number of occurrences of the pattern τ in w, is denoted $|w|_\tau$. If w has no occurrences of τ, then we say that w* avoids *the pattern τ. Two occurences of a pattern τ at positions j_1, j_2 in w* overlap *iff $|j_1 - j_2| < |\tau|$. Similarly, two occurences of two patterns τ_1, τ_2 at positions j_1, j_2 in w with $j_1 < j_2$* overlap *iff $|j_1 - j_2| < |\tau_1|$.*

Example 1.1.2. *The word $w = abbabbabbaaabb$ has two occurrences of the pattern $\tau_1 = babb$ at* *positions* 3 *and* 6*, i.e.* $|w|_\tau = 2$*; these two occurrences overlap in* w *since* $|3-6| < |\tau_1| = 4$*, while* w *avoids the pattern* $\tau_2 = bbb$*.*

Definition 1.1.3 (Periodic words)**.** *A word* $w = w_1w_2\cdots w_n$*,* $n \geq 1$*, is called* periodic *iff there exists a positive integer* $p \in [n-1]$ *such that* $w_{p+i} = w_i$ *for all* $i \in [n-p]$*. The number* p *is called the* period *of* w*.*

Example 1.1.4. *The word* $w = abbabbabbaaabb$ *is periodic since for* $p = 11$ *we have that* $w_{11+i} = w_i$ *for all* $i \in [14-11] = [3]$*, so that* w *has period* 11*. The word* $w = abbabbabbbbb$ *is not periodic.*

Definition 1.1.5 (prefix and suffix)**.**

A nonempty word v *is a prefix (resp. suffix) of* w *if* $w = vu$ *(resp.* $w = uv$*) for any nonempty* $u \in L^*$*.*

Definition 1.1.6 (Borders of a word)**.** *A nonempty word* $v \neq w$ *that is both a prefix and a suffix of* w *is called* border *of* w*, equivalently if* $w = w_1w_2\cdots w_n$ *then there exists* $p \in [n-1]$ *such that*

$$w_{i+p} = w_i \text{ for all } i \in [n-p].$$

The set of borders of w *is denoted by* $\mathcal{V}_w$*.*

Example 1.1.7. *The word* $v = abb$ *is a border of* $w = abbabbabbaaabb$*. so that* $\mathcal{V}_w = \{abb\}$*.*

Moreover, for $w = ababababa$ *we have that* $\mathcal{V}_w = \{a, aba, ababa, abababa, ababababa\}$*.*

Remark. *Obviously, a word* w *is periodic iff* w *has a border.*

Definition 1.1.8 (Autocorrelation set of a word)**.** *For every* $v \in \mathcal{V}_w$ *we denote by* $l(v)$ *the prefix of* w *preceding its suffix* v*, i.e.*

$$w = l(v)v$$

and by $r(v)$ *the suffix of* w *preciding its prefix* v*, i.e.*

$$w = vr(v).$$

The set of all $r(v)$ *where* $v \in \mathcal{V}_w$ *is called the* **autocorrelation set** *of* w *and is denoted by* $\mathcal{C}_w$*.*

Example 1.1.9. *If $w = abaccccaba$ and $v = aba$ then $l(v) = abacccc$ and $r(v) = ccccaba$.*

If $w = baabaab$ then the autocorrelation set of w is $\mathcal{C}_w = \{aab, aabaab\}$

Lemma 1. *A nonempty word u, with $|u| < |w|$, belongs to the autocorrelation set $\mathcal{C}_w$ of a word w iff*

$$wu = u'w \text{ for some word } u' \in L^*.$$

Proof. Indeed, if u belongs to the autocorrelation set $\mathcal{C}_w$ then there exists a unique $v \in \mathcal{V}_w$ such that $u = r(v)$ and $w = l(v)v$. By concatenating $r(v)$ to w we get:

$$wr(v) = l(v)vr(v) \Leftrightarrow$$
$$wr(v) = l(v)w \Leftrightarrow$$
$$wu = u'w$$

Conversely, if $wu = u'w$ then u is a suffix of w. Splitting w to a prefix and suffix such as $w = vu$ leads to:

$$wu = u'vu \Leftrightarrow$$
$$w = u'v$$

The suffix v is a prefix as well, thus w contains a border v such as $wu = u'vr(v)$. The word u' is also a prefix of w whilst being next to the border v. Thus $u'v = w$ which leads to $r(v) = u$, implying that u belongs in $\mathcal{C}_w$. □

Definition 1.1.10 (Correlation set of two words)**.** *Let $w, w' \in L^*$. The* correlation set $\mathcal{C}_{w,w'}$ *of the words w, w' is the set of non-empty words u, with $|u| < |w|$, such that*

$$wu = u'w', \text{ for some } u' \in L^*.$$

Remark. *Note that $\mathcal{C}_{w,w} = \mathcal{C}_w$ and in general $\mathcal{C}_{w,w'} \neq \mathcal{C}_{w',w}$. Also, let $w_1, w_2, \ldots, w_n$ be words in L^*. Then:*

$$\mathcal{C}_{w_i,*} = \bigcup_{s=1}^{n} \mathcal{C}_{w_i,w_s} \text{ and } \mathcal{C}_{*,w_j} = \bigcup_{s=1}^{n} \mathcal{C}_{w_s,w_j}$$

Example 1.1.11. *If $w_1 = baaaa$ and $w_2 = aaaab$ then*

$$\mathcal{C}_{w_1} = \varnothing, \mathcal{C}_{w_2} = \varnothing, \mathcal{C}_{w_1,w_2} = \{b, ab, aab, aaab\}, \mathcal{C}_{w_2,w_1} = \{aaaa\}.$$

1.2 Words and graphs

Definition 1.2.1 (Weighted directed graph)**.** *An* edge-weighted directed graph *or simply* weighted digraph *G is an ordered quadruple (V, E, ϕ, w) where $V \neq \varnothing$, $\phi : E \to V \times V$, $w : E \to L^*$.*

The elements of V are called vertices *and the elements of E are called* edges.

If $\phi(e) = (u, v)$, the vertex u (respectively v) is called start *(resp.* end*) of the edge e and is denoted as $int(e)$ (resp. $fin(e)$).*

Every $e \in E$ has as a weight $w(e)$ a word in L^*.

Remark. *In the following, the weight of the edge e depends only on the $fin(e)$ and $int(e)$.*

Definition 1.2.2 (Correlation graph of patterns)**.** *Let T be a set of non-empty patterns in the alphabet L. The correlation graph $\mathbb{G} = (V, E, \phi, w)$ of T is defined as follows:*

i) $V = T \cup \{\varepsilon\}$

ii) E consists of all the edges e defined as follows: For every pair (τ, τ') in $T \times T$ and for every u in $\mathcal{C}_{\tau,\tau'}$ we define the edge e in E with $\phi(e) = (\tau, \tau')$ and $w(e) = u$.

For every τ in T we define the edge e with $\phi(e) = (\varepsilon, \tau)$ and $w(e) = \tau$.

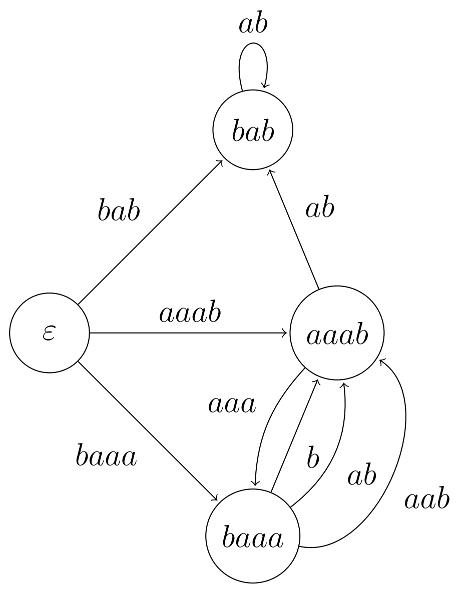
Remark. *The correlation graph $\mathbb{G}$ has the following properties:*

i) It is weakly connected since ε has an indegree of 0 and and outdegree of $|T|$.

ii) Every $u \in T$ has an indegree of $1 + |\mathcal{C}_{u,*}|$ and an outdegree of $|\mathcal{C}_{*,u}|$.

iii) $|E| = |T| + \sum_{i=1}^{|T|} |\mathcal{C}_{*,u_i}|$

Example 1.2.3. *If $T = \{aaab, baaa, bab\}$ then $\mathbb{G}$ is:*



Definition 1.2.4 (Walk in graph)**.** *Let $G = (V, E, \phi, w)$ be a weighted digraph. A* walk *W in G of length n, $n \geq 1$ starting from u and ending at v is a sequence of the form $(e_1, e_2, \ldots, e_n)$, where $int(e_1) = u, fin(e_i) = int(e_{i+1})$ for all $i \in [n-1]$ and $fin(e_n) = v$. A walk is called $closed$ if $u = v$.*

The weight of the walk W is defined as the product of the weights of it's edges i.e., equal the word

$$w(W) = w(e_1)w(e_2)\cdots w(e_n)$$

Definition 1.2.5 (Adjacency matrix of a weighted digraph)**.** *For every weighted digraph $G = (V, E, \phi, w)$ we define the matrix $A = [A_{ij}]$, of dimensions $|V| \times |V|$ as follows: A_{ij} is the formal sum of*

$$A_{ij} = \sum_e w(e)$$

where the sum is over every edge e starting from v_i and ending at v_j. The matrix A is called the $adjacency\ matrix$ of G with respect to the weight function w.

In order to have properly defined the adjacency matrix and the weight of a path on a digraph weighted with words, we need the following two operations between two words w_1, w_2:

i) A commutative formal sum of w_1 and w_2 which is noted as $w_1 + w_2$ where ε is the identity element.

ii) A non-commutative product which actually is the concatenation of w_1 and w_2 noted as w_1w_2 where $\varepsilon w_1 = w_1\varepsilon = \varepsilon$.

Now, the matrix representation of $\mathbb{G}$ regarding the words $\tau_1, \tau_2, \dots, \tau_r$ is the adjacency matrix $\mathbb{A}$:

$$\mathbb{A} = \begin{bmatrix} \varepsilon & \tau_1 & \tau_2 & \tau_3 & \dots & \tau_r \\ \varepsilon & \mathcal{C}_{\tau_1,\tau_1}{}' & \mathcal{C}_{\tau_1,\tau_2}{}' & \mathcal{C}_{\tau_1,\tau_3}{}' & \dots & \mathcal{C}_{\tau_1,\tau_r}{}' \\ \varepsilon & \mathcal{C}_{\tau_2,\tau_1}{}' & \mathcal{C}_{\tau_2,\tau_2}{}' & \mathcal{C}_{\tau_2,\tau_3}{}' & \dots & \mathcal{C}_{\tau_2,\tau_r}{}' \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \varepsilon & \mathcal{C}_{\tau_r,\tau_1}{}' & \mathcal{C}_{\tau_r,\tau_2}{}' & \mathcal{C}_{\tau_r,\tau_3}{}' & \dots & \mathcal{C}_{\tau_r,\tau_r}{}' \end{bmatrix}$$

where $\mathcal{C}_{\tau_i,\tau_j}{}' = \sum\limits_{u \in \mathcal{C}_{\tau_i,\tau_j}} u.$

Example 1.2.6. *Following the previous example, we have:*

$$\mathbb{A} = \begin{bmatrix} \varepsilon & aaab & baaa & bab \\ \varepsilon & \varepsilon & aaa & ab \\ \varepsilon & b + ab + aab & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & ab \end{bmatrix}$$

We also denote as

$$A_{ij}(n) = \sum_{W} w(W)$$

the sum of the weights of all walks of length n starting from the vertex v_i and ending at the vertex v_j. This notation will be useful in proving the transfer-matrix method.
It is trivial that $A_{ij} = A_{ij}(1)$.

1.3 The Transfer-Matrix Method

Proposition 1 (Transfer-Matrix Theorem)**.** *Let A be the adjacency matrix of the weighted digraph $G = (V, E, \phi, w)$ and let v_i, v_j be two vertices of G. The sum of the weights for all walks of length $n, n \geq 1$ from v_i to v_j is equal to the (i, j) element of the matrix A^n.*

Proof. We will use induction on n:

Base case: For $n = 1$ the proposition holds by the definition of A_{ij}

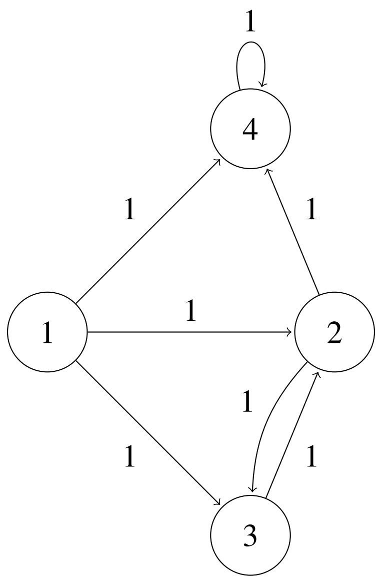
Inductive hypothesis: Assume that the proposition holds for all values of n up to some $k, k > 0$.

Inductive step: for $n = k + 1$:

$$(A^{k+1})_{ij} = (A^k A)_{ij} = \sum_{\lambda=1}^{|V|} (A^k)_{i\lambda} A_{\lambda j}$$

From the inductive hypothesis the term $(A^k)_{i\lambda}$ is the sum of weights for all walks of length k which start from v_i and end in v_λ. Furthermore, $A_{\lambda j}$ is the sum of weights for all edges e (or walks of length 1) with $int(e) = v_\lambda$ and $fin(e) = v_j$. Thus, $(A^k)_{i\lambda} A_{\lambda j}$ is the sum of weights for all walks of length $k+1$ which start from v_i and reach v_λ right before ending in v_j. This leads to $\sum\limits_{\lambda=1}^{|V|} (A^k)_{i\lambda} A_{\lambda j}$ being the sum of weights for all paths of length $k+1$ which start from v_i and end in v_j with next to last vertex any $v \in V$. Implying, it is indeed the sum of weights for all paths of length $k+1$ which start from v_i and end in v_j. $\square$

Example 1.3.1. *Consider the following graph G:*



The adjacency matrix A of G is:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$A^4 = \begin{bmatrix} 0 & 1 & 1 & 4 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transfer-matrix theorem states that since $a_{1,4} = 4$, there are four paths of length 4 starting from 1 and ending in 4. These are: $(1,4,4,4,4),(1,2,4,4,4),(1,2,3,2,4),(1,3,2,4,4)$.

Proposition 2. *$det(I - xA)$ is invertible for every $A \in M$ such that $det(I - xA) \in R[[x]]$, where $R[[x]]$ is the ring of formal power series.*

Proof. It is well known that a formal series is invertible if and only if it has a nonzero constant term. Therefore:

$$\begin{aligned} det(I - xA) &= det(I - \frac{1}{z}A) \\ &= det(\frac{1}{z}(zI - A)) \\ &= (\frac{1}{z})^n det(zI - A) \\ &= \frac{(-1)^n}{z^n} det(A - zI) \\ &= \frac{(-1)^n}{z^n}((-1)^n z^n + b_{n-1}z^{n-1} + \ldots + b_1 z + b_0) \\ &= 1 + (-1)^n(b_{n-1}z^{-1} + \ldots + b_1 z^{-(n-1)} + b_0 z^{-n}) \\ &= 1 + (-1)^n(b_{n-1}x + \ldots + b_1 x^{n-1} + b_0 x^n) \end{aligned}$$

where $b_0, b_1, \ldots, b_n \in R$. □

Proposition 3. $\sum\limits_{n=0} A^n x^n = (I - xA)^{-1}$ *where* $I - xA$, $\sum\limits_{n=0} A^n x^n \in M[[x]]$, *where M is the set of* *all $m \times m$ matrices.*

Proof.

$$\begin{aligned}
[x^n](I - xA)\sum_{j=0} A^j x^j &= \sum_{k=0}^{n}[x^k](I - xA)[x^{n-k}]\sum_{j=0} A^j x^j \\
&= \begin{cases} IA^0, & n = 0 \\ IA^n + (-A)A^{n-1}, & n \geq 1 \end{cases} \\
&= \begin{cases} I, & n = 0 \\ 0, & n \geq 1 \end{cases}
\end{aligned}$$

□

Corollary 1. *It follows from the previous proposition that if $(A)_{ij}$ is the i, j-th element of a matrix $A \in M$ then $(\sum\limits_{n=0} A^n x^n)_{ij} = ((I - xA)^{-1})_{ij}$.*

1.4 Generating Functions

Generating functions are a very useful tool when solving enumeration problems. They can find an exact, recurrence or an asymptotic formula for non negative integer sequences as well as calculate mean averages, statistical properties and prove identities. The manipulation of generating functions is carried out in the ring of formal power series, hence there is no need to take matters of convergence into account. The theory of generating functions has been rigorously described in the books Generatingfunctionology (2005) [7] of H. Wilf and Analytic Combinatorics(2013)[8] of P. Flajolet and R. Sedgewick. In the following, some notations are given about ordinary generating functions or simply generating functions.

There are two ways to define a generating function. The first is based on a non negative sequence (a_n) such as:

$$F(x) = \sum_{n=0}^{\infty} a_n x^n$$

For the second approach, let S be a combinatorial class. Every function $p : S \to \mathbb{N}$ is called parameter *and the generating function of S respecting the parameter p is defined as:*

$$F(x) = \sum_{a \in S} x^{p(\alpha)}$$

Remark. *The parameter p can partition S into the sets $S_n, n \in \mathbb{N}$ where:*

$$S_n = \{\alpha \in S : p(\alpha) = n\}$$

These two approaches are equivalent if we let a_n be the cardinality of S_n.
Indeed,

$$\sum_{\alpha \in S} x^{p(\alpha)} = \sum_{n \in \mathbb{N}} \sum_{\alpha \in S_n} x^{p(a)} = \sum_{n \in \mathbb{N}} \sum_{\alpha \in S_n} x^n = \sum_{n \in \mathbb{N}} x^n \sum_{\alpha \in S_n} 1 = \sum_{n \in \mathbb{N}} |S_n| x^n$$

In other words, an ordinary generating function $F(x)$ is a formal power series with respect to the variable x, in which the coefficient of x^n denoted as $[x^n]F(x)$ is the number of elements α in the set S such that $p(\alpha) = n$.

The bivariate generating function of a sequence $(a_{n,k})$ is defined as:

$$F(x, y) = \sum_{n=0}^{\infty} \sum_{k=0}^{\infty} a_{n,k} x^n y^k$$

Equivalently, the bivariate generating function of S with respect to the parameters $p, q : S \to \mathbb{N}$ is defined as:

$$F(x, y) = \sum_{\alpha \in S} x^{p(\alpha)} y^{q(\alpha)}$$

and the coefficient of $x^n y^k$ is denoted as $[x^n y^k]F(x, y)]$.
Multivariate generating functions with more than two variables are defined in a similar way.

Combinatorial Interpretations

Let $F(x, y)$ be the generating function of S with respect to the parameters p, q.

i) Let $y = 1$, then the generating function $F(x, 1) = F(x)$ enumerates the elements of S based *on solely the parameter p.*

ii) *Let $y = 0$, then the generating function $F(x, 0)$ enumerates the elements of S whose image of the parameter q is 0 regarding the parameter p.*

iii) $\frac{[x^n]\frac{\partial F}{\partial y}\big|_{y=1}}{[x^n]F(x,1)}$ *Is the (mean) average number of objects regarding the parameter q, where their image of the parameter p is n.*

In general, the mean average of a variable X, which consists of the values $x_1, x_2, \ldots, x_k$ as well as their respective frequencies $n_1, n_2, \ldots, n_k$, is denoted by:

$$\mu = \frac{\sum\limits_{i=1}^{k} x_i n_i}{\sum\limits_{i=1}^{k} n_i}$$

Indeed, in the case of a bivariate generating function:

$$\begin{aligned}\frac{\partial F}{\partial y} &= \sum_{n=0}\sum_{k=0} k a_{n,k} x^n y^{k-1} \\ &= \sum_{n=0}\sum_{k=1} k a_{n,k} x^n y^{k-1}\end{aligned}$$

$$\frac{\partial F}{\partial y}\bigg|_{y=1} = \sum_{n=0}\sum_{k=1} k a_{n,k} x^n$$

Clearly, $[x^n] = \sum\limits_{k=1} k a_{n,k} = a_{n,1} + 2a_{n,2} + \ldots + m a_{n,m}$. The $k a_{n,k}$ term means that k objects are accounted for every object with the image of the first parameter equal to n and equivalently the second's equal to k. Thus, dividing by $[x^n]F(x, 1)$ which is the count of all objects with the image of the first parameter equal to n leads to:

$$\mu(n) = \frac{a_{n,1} + 2a_{n,2} + 3a_{n,3} + \ldots + m a_{n,m}}{a_{n,0} + a_{n,1} + a_{n,2} + a_{n,3} + \ldots + a_{n,m}} = \frac{\sum\limits_{k=1}^{m} k a_{n,k}}{\sum\limits_{k=0}^{m} a_{n,k}}$$

Remark. *In the case of strings, it is well known that the amount of words of length n in a finite alphabet A will be $|A|^n$. Therefore, substituting x with $\frac{x}{|A|}$ in $\frac{\partial F}{\partial y}|_{y=1}$ will make it equal to $\mu(n)$ as well.*

iv) The coefficients of a generating function can easily be evaluated using the Taylor theorem which states that a function $f : \mathbb{R} \to \mathbb{R}$ which is $k \in \mathbb{N}^*$ times differentiable at the point $x_0 \in \mathbb{R}$ is approximated by:

$$f(x) = \sum_{n=0}^{k} \frac{f^{(k)}(x_0)}{n!}(x - x_0)^n + R_{k,f,x_0}(x)$$

where R_{k,f,x_0} is the remainder.

The usefulness of these interpretations can be seen in the last chapter when it comes to solving problems. Regarding the Taylor theorem, in our cases the generating functions are rational and with constant term ± 1 in the denominator, thus they are infinitely differentiable at $x_0 = 0$ and as $k \to \infty : R_{k,f,0} \to 0$. Applying this in Python's sage is as easy as:

```
sage: G(x)=-(x^7 - x^6 + x^5 - 5*x^4 - 7*x^3 + x^2 + 5*x + 1)/
(x^9 - x^8 - 2*x^6 - 10*x^5 + 4*x^4 + 2*x^3 + 3*x - 1)
sage: G.series(x,16)
x |--> 1 + 8*x + 25*x^2 + 70*x^3 + 225*x^4 + 748*x^5 + 2401*x^6
+ 7668*x^7 + 24649*x^8 + 79344*x^9 + 255025*x^10 + 819494*x^11
+ 2634129*x^12 + 8467464*x^13 + 27217089*x^14 + 87483296*x^15
+ Order(x^16)
```

Chapter 2

The Goulden-Jackson cluster method

2.1 Historical review

Some of the problems that showed up after the rise of the computer science in the previous decades was of the kind: How many binary words of length n include the binary word 101 and how many times does it occur in each word. To solve this problem of enumeration with generating functions one needs to consider that each binary word can be partioned as the empty word ε, a binary word starting with 1 or a binary word starting with 0 and then act as follows:

$$\begin{aligned}
F(x,y) = \sum_{a\in A^*} x^{|a|}y^{|a|_{101}} &= 1 + \sum_{a\in A^*} x^{|1a|}y^{|1a|_{101}} + \sum_{a\in A^*} x^{|0a|}y^{|0a|_{101}} \\
&= 1 + x\sum_{a\in A^*} x^{|a|}y^{|1a|_{101}} + xF(x,y) \\
&= 1 + xF(x,y) + x\big(1 + \sum_{a\in A^*} x^{|1a|}y^{|11a|_{101}} + \sum_{a\in A^*} x^{|0a|}y^{|10a|_{101}}\big) \\
&= 1 + xF(x,y) + x\big(1 + xF(x,y) + x\sum_{a\in A^*} x^{|a|}y^{|10a|_{101}}\big) \\
&= 1 + x + (x+x^2)F(x,y) + x^2\sum_{a\in A^*} x^{|a|}y^{|10a|_{101}} \\
&= 1 + x + (x+x^2)F(x,y) \\
&+ x^2\big(1 + \sum_{a\in A^*} x^{|1a|}y^{|101a|_{101}} + \sum_{a\in A^*} x^{|0a|}y^{|100a|_{101}}\big) \\
&= 1 + x + (x+x^2)F(x,y) + x^2 + x^3yF(x,y) + x^3F(x,y) \\
&= 1 + x + x^2 + (x + x^2 + x^3 + x^3y)F(x,y)
\end{aligned}$$

$$F(x,y) = \sum_{a\in A^*} x^{|a|}y^{|a|_{101}} = \frac{1+x+x^2}{1-x-x^2-x^3-x^3y}$$

This, however, requires a lot of algebraic manipulation with an alphabet of only two letters and a pattern of length 2. To solve harder problems than this where the alphabet has many letters and the problem requires counting occurences of more than two patterns simultaneously, one more powerful tool is required. This is exactly what the Goulden-Jackson cluster method (GJCM) does.

The GJCM was introduced By I.P. Goulden and D.M. Jackson (1979) in [3],[4] as a general method to count the number of words of L^* avoiding patterns from a prescribed set. Two years later the notion of the autocorrelation of a word which is extended to correlation between words is introduced by L.J. Guibas and A.M. Odlyzko (1981) in [15],[16], a notion that is also described in this thesis. In contrast to Goulden and Jackson (1979)[3], Guibas and Odlyzko (1981)[15],[16] worked with specific cases such as: enumerating sequences avoiding a pattern or sequences ending with the first occurence of a pattern. Later, M.Régnier and Szpankowski (1997) [17] used multivariate analysis to count several words simultaneously. A year later they applied their previous work to markovian sequences (1998)[18]. The method that was formally introduced *by Goulden and Jackson (1979)[3] was about the reduced case where the patterns do not occur in each other. Almost twenty years after that, J. Noonan and D. Zeilberger (1999) [6] solved the problem of the non-reduced case, but without publishing an explicit formula. They provided, however, a sparse linear system as a solution along with the maple packages that solve it. This generalization was made in order to enumerate the words of L^* with a fixed number of occurrences of patterns from a prescribed set. Six years after that, along with the relevant book of M. Lothaire* Applied Combinatorics on Words *(2005) [19], Y. Kong applied the method to an assymetrical Bernoulli model that is used to generate symbols (2005)[20]. His work was based on the results of Noonan and Zeilberger (1999) [6] for the reduced case. Kong also emphasized on the idea of Goulden and Jackson [3],[4] about the inclusion-exclusion approach for its simplicity, while comparing it to the Régnier and Szpankowski method [17]. Following up, alternative proofs for both the reduced and non-reduced cases were given by F. Bassino, J. Clément, et al. (2007) [2]. In a newer version of their paper they provided a new approach with the use of Automata (2012)[1] and more specifically based on the Aho-Corasick Automaton (1975) [21]. In all the above cases the enumerations were made possible by using generating functions. It is worth mentioning that some applications of the GJCM in biology and more specifically in genomes were made by B. Hao, et al. (2000)[10], [11], [12] and (2002) [13]. A notable example is the creation of fractals based on the under or over representation of certain DNA strings and the use of the GJCM to evaluate the dimension of these fractals [10].*

2.2 Counting one pattern

The case of counting words with one pattern is interesting in its own and will give a nice introduction to the GJCM.

2.2.1 Marked words

The occurrences of the pattern τ in w can be encoded by the elements of the set

$$S_\tau(w) = \{i \in [n] : \tau \textit{ starts at the } i\textit{-th position of } w\},$$

that is, each element of $S_\tau(w)$ indicates the position of the first letter of an occurrence of τ in w. Clearly, $S_\tau(\varepsilon) = \varnothing$.

Definition 2.2.1 (Marked Word)**.** *A* marked word *is a pair (w, I), where w is a word and $I \subseteq S_\tau(w)$. An occurrence of τ, starting at the i-th position of w, is* marked *in (w, I) iff $i \in I$. Practically, a marked word is a word w having some of its occurrences of τ marked. These markings are encoded by the set I.*

For example, let $\tau = aba$ be a pattern and $w = abaaaba$ be a word. Then $(w, \{5\})$ is a marked word.

Clearly, the word w generates $2^{|w|_\tau}$ marked words, where $\binom{|w|_\tau}{k}$ of them have exactly k markings. Every word w can be considered to be marked, via the mapping $w \mapsto (w, \varnothing)$.

The concatenation of two marked words (w, I), (v, J) is defined as follows:

$$(w, I)(v, J) = (wv, I \cup (|w| + J)).$$

where $|w| + J = \{|w| + j : j \in J\}$.

We denote by MW the set of marked words, and we define

$$H = H(x, y) = \sum_{(w,I)\in MW} x^{|w|} y^{|I|}$$

to be its generating function, with respect to the length $|w|$ and the number of marked occurrences of τ.

The introduction of the set of marked words and its corresponding generating function H is justified by the following proposition.

Proposition 4. *Let $F = F(x, y) = \sum\limits_{w\in L^*} x^{|w|} y^{|w|_\tau}$ be the generating function of L^* which counts the occurrences of the pattern τ and $H = H(x, y) = \sum\limits_{(w,I)\in MW} x^{|w|} y^{|I|}$ be the generating function of the set of marked words which counts the marked occurrences of τ. Then*

$$H(x, y) = F(x, y + 1).$$

Proof. Indeed,

$$\begin{aligned}
F(x, y+1) &= \sum_{w \in L^*} x^{|w|}(y+1)^{|w|_\tau} \\
&= \sum_{w \in L^*} x^{|w|} \sum_{k=0}^{|w|_\tau} \binom{|w|_\tau}{k} y^k \\
&= \sum_{w \in L^*} x^{|w|} \sum_{I \subseteq S_\tau(w)} y^{|I|} \\
&= \sum_{(w,I) \in MW} x^{|w|} y^{|I|} \\
&= H(x, y)
\end{aligned}$$

□

2.2.2 Clusters

It turns out that evaluation of H is much more straight-forward than the evaluation of F but we need the notion of cluster.

Definition 2.2.2 (Cluster)**.** *We recall that a* cluster *(of τ) is a marked word that is not a concatenation of two marked words. Equivalenty, a cluster is a marked word (c, I) satisfying the following two conditions:*

i) Every letter in c belongs to some marked occurrence of τ.

ii) Any two consecutive marked occurrences of τ in the cluster overlap.

Note that a cluster may contain occurrences of τ which are not marked.

For example, let $\tau = ababa$ be a pattern and $c = ababababa$ be a word. Then $(c, \{1, 5\})$ is a cluster. Note that, there is an occurrence of τ starting at the 3rd position of c which is not marked.

We remark that (c, I) is a cluster iff it satisfies the following conditions

$$\min I = 1, \max I = |c| - |\tau| + 1$$

and the absolute differrence of any two consecutive elements of I is less than $|\tau|$.

Definition 2.2.3 (Set of Clusters)**.** *We denote the set of all clusters of τ by C_τ. The trivial cluster $(\tau, \{1\})$ of C_τ is denoted by τ, for simplicity.*

Definition 2.2.4 (Cluster Generating Function). *We define the generating function of the set of clusters C_τ for the pattern τ to be*

$$C(x,y) = \sum_{(c,I)\in C_\tau} x^{|c|}y^{|I|}$$

In order to evaluate C, we define the operation

$$* : \mathcal{V}_\tau \times C_\tau \to C_\tau \setminus \{\tau\}$$

so that for a border $v \in \mathcal{V}_\tau$ and $(c', I') \in C_\tau$ we have:

$$v * (c', I') = (l(v)c', I),$$

where $I = \{1\} \cup (|l(v)| + I')$. It is easy to check, using the definition of clusters, that $v * (c', I') \in C_\tau \setminus \{\tau\}$.

This operation can be used to decompose every element of $C_\tau \setminus \{\tau\}$. Indeed, if $(c, I) \in C_\tau \setminus \{\tau\}$, then the common segment v of the first and second marked occurrence of τ in (c, I) is a border of τ. Setting $c = l(v)c'$ and $I' = -|l(v)| + (I \setminus \{1\})$, we obtain that

$$(c, I) = v * (c', I').$$

Proposition 5. *For every pattern $\tau \in L^*$ the cluster generating function of MW is given by the formula*

$$C_\tau(x,y) = \frac{x^{|\tau|}y}{1 - x^\tau y \sum\limits_{v\in V_\tau} x^{-|v|}}.$$

Proof.

$$\begin{aligned}
C_\tau(x,y) &= \sum_{(c,I)\in C_\tau} x^{|c|}y^{|I|} \\
&= x^{|\tau|}y + \sum_{v\in\mathcal{V}_\tau}\sum_{(c'I')\in C_\tau} x^{|l(v)|}x^{|c'|}y^{|I'|+1} \\
&= x^{|\tau|}y + yC_\tau(x,y)\sum_{v\in\mathcal{V}_\tau} x^{|l(v)|}
\end{aligned}$$

which leads to

$$C_\tau(x,y) = \frac{x^{|\tau|}y}{1 - x^{|\tau|}y\sum_{v\in V_\tau} x^{-|v|}}.$$ □

Now we have all the tools needed to evaluate $H(x,y)$. Every marked word can be decomposed uniquely as the concatenation of single letters which do not appear in any marked occurrence of τ and of clusters.

Every $(w,I)\in MW$ is uniquely decomposed as follows:

$$(w,I) = \begin{cases} \varepsilon, \ \textit{or} \\ a(w',I'), \ \textit{or} \\ (c,J)(w',I') \end{cases}$$

where $(c,J)\in C_\tau$, $(w',I')\in MW$ and $a\in L$.

Proposition 6. *For every pattern $\tau\in L^*$ we have that:*

$$F(x,y) = \frac{1}{1 - x|L| - C_\tau(x,y-1)}$$

where

$$C_\tau(x,y) = \frac{x^{|\tau|}y}{1 - x^{|\tau|}y\sum_{v\in V_\tau} x^{-|v|}}.$$

Proof.

$$\begin{aligned} H(x,y) &= \sum_{(w,I)\in MW} x^{|w|}y^{|I|} \\ &= 1 + \sum_{a\in L}\sum_{(w',I')\in MW} x^{|w'|+1}y^{|I'|} + \sum_{(c,J)\in C_\tau}\sum_{(w',I')\in MW} x^{|c|+|w'|}y^{|J|+|I'|} \\ &= 1 + x|L|H(x,y) + C_\tau(x,y)H(x,y) \end{aligned}$$

hence we have that

$$H(x,y) = \frac{1}{1-x|L|-C_\tau(x,y)}$$

□

2.3 Counting multiple patterns - the reduced case

We will now present how to count multiple patterns. For simplicity we consider the reduced case. The non-reduced case is more technical and is outside the scope of this thesis. In this case no pattern τ_j is a subword of another pattern τ_s. Essentialy, no marked occurrence is a factor of another marked occurrence.

2.3.1 Marked words

Now we need to distinguish each marked occurrence of a pattern individually. The occurrences of the patterns $\tau_1, \tau_2, \ldots, \tau_r$ in w can be encoded by the elements of the sets

$$S_{\tau_j}(w) = \{i \in [n] : \tau_j \text{ starts at the } i\text{-th position of } w\},$$

that is, each element of $S_{\tau_j}(w)$ indicates the position of the first letter of an occurrence of τ_j in w. Clearly, $S_{\tau_j}(\varepsilon) = \varnothing$.

Definition 2.3.1 (Marked Word)**.** *A* marked word *is a (r+1)-tuple $(w, I_1, I_2, \ldots, I_r)$, where w is a word and $I_j \subseteq S_{\tau_j}(w)$, $j \in [r]$. An occurrence of τ_j, starting at the i-th position of w, is* marked *in $(w, I_1, I_2, \ldots, I_r)$ iff $i \in I_j$. Practically, a marked word is a word w having some of its occurrences* *of* $\tau_1, \tau_2, \ldots, \tau_r$ *marked. These markings are encoded by the sets* $I_1, I_2, \ldots, I_r$*.*

For example, let $\tau_1 = aab$ *and* $\tau_2 = baa$ *be two patterns and* $w = aaabaaaab$ *be a word. Then* $(w, \{7\}, \{4\})$ *is a marked word (of* τ_1, τ_2 *).*

Clearly, the word w *generates* $2^{\sum_{j=1}^{r} |w|_{\tau_j}}$ *marked words, where* $\binom{\sum_{j=1}^{r} |w|_{\tau_j}}{k}$ *of them have exactly* k *markings. Every word* w *can be considered to be marked, via the mapping* $w \mapsto (w, \varnothing, \varnothing, \ldots, \varnothing)$*.*

The concatenation of two marked words $(w, I_1, I_2, \ldots, I_r)$*,* $(v, J_1, J_2, \ldots, J_r)$ *is defined as follows:*

$$(w, I_1, I_2, \ldots, I_r)(v, J_1, J_2, \ldots, J_r) = (wv, I_1 \cup (|w| + J_1), I_2 \cup (|w| + J_2), \ldots, I_r \cup (|w| + J_r)).$$

where $|w| + J = \{|w| + j : j \in J\}$*.*

We denote by MW *the set of marked words, and we define*

$$H = H(x, y_1, y_2, \ldots, y_r) = \sum_{(w,I_1,I_2,\ldots,I_r)\in MW} x^{|w|} y^{|I_1|} y^{|I_2|} \cdots y^{|I_r|} = \sum_{(w,I_1,I_2,\ldots,I_r)\in MW} x^{|w|} \prod_{j=1}^{r} y^{|I_j|}$$

to be its generating function, with respect to the length $|w|$ *and the number of marked occurrences of* $\tau_1, \tau_2, \ldots, \tau_r$*.*

Proposition 7. *Let* $F = F(x, y_1, y_2, \ldots, y_r) = \sum_{w \in L^*} x^{|w|} \prod_{j=1}^{r} y_j^{|w|_{\tau_j}}$ *be the generating function of* L^* *which counts the occurrences of the pattern* $\tau_1, \tau_2, \ldots, \tau_r$ *and* $H = H(x, y_1, y_2, \ldots, y_r) = \sum_{(w,I_1,I_2,\ldots,I_r)\in MW} x^{|w|} \prod_{j=1}^{r} y_j^{|I_j|}$ *be the generating function of the set of marked words which counts the marked occurrences of* $\tau_1, \tau_2, \ldots, \tau_r$*. Then*

$$H(x, y_1, y_2, \ldots, y_r) = F(x, y_1 + 1, y_2 + 1, \ldots, y_r + 1).$$

Proof. Indeed,

$$\begin{aligned}
F(x, y_1+1, y_2+1, \ldots, y_r+1) &= \sum_{w\in L^*} x^{|w|} \prod_{j=1}^{r} (y_j+1)^{|w|_{\tau_j}} \\
&= \sum_{w\in L^*} x^{|w|} \prod_{j=1}^{r} \sum_{k=0}^{|w|_{\tau_j}} \binom{|w|_{\tau_j}}{k} y_j^k \\
&= \sum_{w\in L^*} x^{|w|} \prod_{j=1}^{r} \sum_{I_j\subseteq S_{\tau_j}(w)} y_j^{|I_j|} \\
&= \sum_{(w,I_1,I_2,\ldots I_r)\in MW} x^{|w|} \prod_{j=1}^{r} y_j^{|I_j|} \\
&= H(x, y_1, y_2, \ldots, y_r) \qquad \square
\end{aligned}$$

2.3.2 Clusters

For the evaluation of H we need to generalize the notion of cluster.

A cluster *(of $\tau_1, \tau_2, \ldots, \tau_r$) is a marked word that is not a concatenation of two marked words. Equivalenty, a cluster is a marked word $(c, I_1, I_2, \ldots, I_r)$ satisfying the following two conditions:*

i) Every letter in c belongs to some marked occurrence of $\tau_1, \tau_2, \ldots, \tau_r$.

ii) Any two consecutive marked occurrences of τ_i, τ_j in the cluster overlap.

Note that a cluster may contain occurrences of τ_j's which are not marked.

For example, let $\tau_1 = ababa$ and $\tau_2 = babab$ be two patterns and $c = abababababab a$ be a word. Then $(c, \{1,9\}, \{4,8\})$ is a cluster. Note that, there are occurrences starting at the 2nd and 3rd position of c which are not marked.

We remark that $(c, I_1, I_2, \ldots, I_r)$ is a cluster iff it satisfies the following conditions:

$$\min \bigcup_{j=1}^{r} I_j = 1, \max \bigcup_{j=1}^{r} I_j = |c| - |\tau_s| + 1, \textit{ for some } \tau_s,\ s \in [r].$$

and the absolute differrence of any two consecutive elements p, q of $\bigcup_{j=1}^{r} I_j$ is less than $|\tau_s|$ whenever $p \in I_s$.

We denote the set of all clusters of $\tau_1, \tau_2, \ldots, \tau_r$ *by* $C_{\tau_1,\tau_2,\ldots,\tau_r}$.

The trivial clusters $(\tau_j, \varnothing, \ldots, \varnothing, \underbrace{\{1\}}_{j\text{-th position}}, \varnothing, \ldots, \varnothing)$ *of* $C_{\tau_1,\tau_2,\ldots,\tau_r}$ *are denoted by* τ_j*, for simplicity.*

The set $C_{\tau_1,\tau_2,\ldots,\tau_r}$ *is partitioned into* r *subsets where* I_j *contains* 1*. The set of clusters where* $1 \in I_j$ *is denoted by* $C^{\tau_j}_{\tau_1,\tau_2,\ldots,\tau_r}$.

We define the generating function of the set of clusters C_τ *for the pattern* τ *to be*

$$C(x, y_1, y_2, \ldots, y_r) = \sum_{(c,I_1,I_2,\ldots,I_r)\in C_{\tau_1,\tau_2,\ldots,\tau_r}} x^{|c|} \prod_{j=1}^{r} y_j^{|I_j|}$$

Every marked word can be decomposed uniquely as the concatenation of single letters which do not appear in any marked occurrence of τ *and of clusters.*

Every $(w, I_1, I_2, \ldots, I_r) \in MW$ *is uniquely decomposed as follows:*

$$(w, I_1, I_2, \ldots, I_r) = \begin{cases} \varepsilon, \text{ or} \\ a(w', I'_1, I'_2, \ldots, I'_r), \text{ or} \\ (c, J_1, J_2, \ldots, J_r)(w', I'_1, I'_2, \ldots, I'_r) \end{cases}$$

where $(c, J_1, J_2, \ldots, J_r) \in C_{\tau_1,\tau_2,\ldots\tau_r}$, $(w', I'_1, I'_2, \ldots, I'_r) \in MW$ *and* $a \in L$.

Proposition 8.

$$F(x, y_1, \ldots, y_r) = \frac{1}{1 - x|L| - C_\tau(x, y_1 - 1, y_2 - 1, \ldots, y_r - 1)}$$

where

$$C_\tau(x, y_1, y_2, \ldots, y_r) = \sum_{(c,I_1,I_2,\ldots,I_r)\in C_{\tau_1,\tau_2,\ldots,\tau_r}} x^{|c|} \prod_{j=1}^{r} y_j^{|I_j|}$$

Proof.

$$\begin{aligned} H(x,y) &= \sum_{(w,I_1,I_2,\ldots,I_r)\in MW} x^{|w|} \prod_{j=1}^{r} y_j^{|I_j|} \\ &= 1 + \sum_{a\in L} \sum_{(w',I'_1,I'_2,\ldots,I'_r)\in MW} x^{|w'|+1} \prod_{j=1}^{r} y_j^{|I'_j|} + \sum_{(c,J_1,J_2,\ldots,J_r)\in C_{\tau_1,\tau_2,\ldots,\tau_r}} \sum_{(w',I'_1,I'_2,\ldots,I'_r)\in MW} x^{|c|+|w'|} y^{|J|+|I'|} \\ &= 1 + x|L|H(x, y_1, y_2, \ldots, y_r) + C_\tau(x, y_1, y_2, \ldots y_r)H(x, y_1, y_2, \ldots, y_r) \end{aligned}$$

hence we have that

$$H(x,y) = \frac{1}{1 - x|L| - C_\tau(x, y_1, y_2, \ldots, y_r)}. \qquad \square$$

In other words, the problem of evaluating F is reduced to the problem of evaluating the cluster generating function for multiple patterns. This, however, is out of the scope of this dissertation. The purpose was to demonstrate the previous mathematical techniques regarding words and the inclusion-exclusion approach on how to count them. In the remaining part of this chapter we establish a bijection between clusters and weights of paths and then we state the main theorem.

Proposition 9. *A marked word $(c, I_1, I_2, \ldots, I_r)$ (of $\tau_1, \tau_2, \ldots, \tau_r$) is a cluster if and only if c is the weight of a walk starting from ε in the correlation graph $\mathbb{G}$ of $\tau_1, \tau_2, \ldots, \tau_r$.*

Proof. Let W be a walk in $\mathbb{G}$ starting from ε with length k, i.e.

$$W = \varepsilon \xrightarrow{e_0} v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \ldots \xrightarrow{e_k} v_k$$

Then, $v_0, v_1, v_2, \ldots, v_k \in T$, $w(e_0) \in T$ and $w(e_j) \in \mathcal{C}_{\tau_j, \tau_{j-1}}$ We define a marked word $(w, I_1, I_2, \ldots, I_r)$ as follows:

$w = w(e_k)w(e_{k-1}) \ldots w(e_2)w(e_1)$ and the sets $I_1, I_2, \ldots, I_r$ are defined inductively:

$I_1, I_2, \ldots, I_r = \varnothing$. For every edge e_j set:

i) Increase all elements of $I_1, I_2, \ldots, I_r$ by $|w(e_j)|$ (if there are any).

ii) Add 1 as a new element to the set I_s where $fin(e_j) = v_j = \tau_s$.

In other words, we start from a pattern and mark its occurrence, then we concatenate elements from the correlation sets creating new overlapping (by definition of correlation sets) occurrences. We mark these occurrences at the starting position of the new word $w(e_j) \ldots w(e_1)w(e_0)$ as well as shifting the previous marked occurrences to their new positions. As a consequence, any two consecutive marked occurrences overlap and every letter of $w(e_k) \ldots w(e_1)w(e_0)$ belongs in a correlation set or the starting pattern from which marked occurrences are created, implying it belongs in a marked occurrence. Hence $(w(e_k) \ldots w(e_1)w(e_0), I_1, I_2, \ldots, I_r)$ is a cluster.

Conversely, let $(c, I_1, \ldots, I_r)$ be a cluster of $\tau_1, \tau_2, \ldots, \tau_r$. We build a sequence of the form $((v_1, p_1), (v_2, p_2), \ldots, (v_k, p_k))$ where p_j is the starting position of the word v_j (a pattern) in c. The sequence is sorted in descending order of the positions p such as: $p_1 = |c| - |v_1| + 1 > p_2 > \cdots > p_k = 1$. Furthermore, let $w[m,n]$ be the subword which starts from the m-th position and ends at the n-th position of c. For example, if $c = aabaa$ then $w[3,5] = baa$. Each word $w[p_j, |c|]$ for all $j \in [k]$ is a suffix of c and starts with a marked occurrence of a pattern. Therefore, all of it's letters belong in a marked occurrence and every two consecutive marked occurences overlap, as this happens by definition in c and $w[p_j, |c|]$ can't contain the prefix of a marked occurrence without it's suffix. We set $y_1 = v_1$ and each y_j is the word such that $y_j w[p_{j-1}, |c|] = w[p_j, |c|]$, for all $j \in \{2, \ldots, k\}$. By definition of the correlation sets, $y_j \in \mathcal{C}_{v_j, v_{j-1}}$ for all $j \in \{2, \ldots, k\}$. This leads to a unique walk $c = y_k \ldots y_2 y_1$ of the graph $\mathbb{G}$ such as:

$$\varepsilon \xrightarrow{y_1} v_1 \xrightarrow{y_2} v_2 \xrightarrow{y_3} \ldots \xrightarrow{y_k} v_k$$

Hence there is a bijection mapping walks of $\mathbb{G}$ to clusters. □

Proposition 10 (Cluster Generating Function)**.** *Let T be a set of r patterns and $\mathbb{A}$ the corresponding adjacency matrix. We use the transformation $\phi : \mathcal{C}_{\tau_i,\tau_j} \to R[x, y_j], \phi(u) = x^{|u|} y_j$ and map each element $a_{i,j}$ of $\mathbb{A}$ to $\phi(a_{i,j})$. This image will be the i,j-th element of a new matrix $\overline{\mathbb{A}}$. Then:*

$$C_{\mathcal{T}}(x, y_1, y_2, \ldots, y_r) = [1, 0, 0, \ldots, 0] \quad (I - \overline{\mathbb{A}})^{-1} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Example 2.3.2. *Let $\tau_1 = aaab$ and $\tau_2 = baaa$ in $\{a,b\}^*$. The adjacency matrix is:*

$$\mathbb{A} = \begin{bmatrix} \varepsilon & aaab & baaa \\ \varepsilon & \varepsilon & aaa \\ \varepsilon & b + ba + baa & \varepsilon \end{bmatrix}$$

Thus, the cluster generating function is:

$$C(x,y_1,y_2) = [1,0,0]\begin{bmatrix} 1 & -x^4y_1 & -x^4y_2 \\ 0 & 1 & -x^3y_2 \\ 0 & -(x+x^2+x^3)y_1 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \frac{x^4y_1 + x^4y_2 + x^7y_1y_2 + (x^5+x^6+x^7)y_1y_2}{1 - y_1y_2(x^4+x^5+x^6)}$$

Chapter 3

Applications of the Goulden-Jackson cluster method

Using the GJCM we can evaluate the generating function for various sets of patterns.

3.1 Different kinds of positions on $3 \times n$ Chessboards

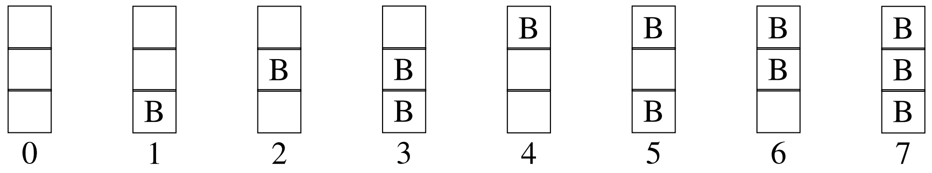
In this chapter, we consider a set of problems where we do not need to count words but formations on a $3 \times n, n \in \mathbb{N}^*$ *chessboard with various properties. By associating each column to a letter we can reduce the problem of counting proper formations to the problem of counting words avoiding specifics set of patterns. For all of these we use some Python scripts to find the correlation set and the correlation matrix and the program sage to perform the computations. Most of the corresponding generating functions' coefficients appear in OEIS but some of them are new.*

3.1.1 Non-Attacking Bishop positions

In [5], Low and Kapbasov evaluated the generating function which counts the number of non-attacking bishop positions on the $3 \times n$ *chessboards. Using generating functions regarding black and white squares they provided the generating function:*

$$\mathbb{B}_3(x) = \frac{-1 - 5x - x^2 + 7x^3 + 5x^4 - x^5 + x^6 - x^7}{-1 + 3x + 2x^3 + 4x^4 - 10x^5 - 2x^6 - x^8 + x^9}$$

However, they provided the idea of associating the 2^3 bishop positions with letters such as:



From which they yielded the 88 forbidden patterns:

12, 13, 16, 17, 21, 23, 24, 25, 26, 27, 31, 32, 33, 34, 35, 36, 37, 42, 43, 46, 47, 52, 53, 56, 57, 61, 62, 63, 64, 65, 66, 67, 71, 72, 73, 74, 75, 76, 77

104, 105, 106, 107, 114, 115, 144, 145, 154, 155, 304, 305, 306, 307, 401, 403, 405, 407, 411, 415, 441, 445, 451, 455, 501, 503, 504, 505, 506, 507, 511, 514, 515, 541, 544, 545, 551, 554, 555, 601, 603, 605, 607, 701, 703, 704, 705, 706, 707

Utilizing this idea, the same result will be provided alternatively by using the GJCM to evaluate the generating function $F(x, y_1, \ldots, y_{88})$ which counts the occurrences of the above 88 patterns such that:

$$F(x, 0, 0, \ldots, 0) = \mathbb{B}_3(x)$$

Note that, no pattern is a subword of another pattern and therefore the patterns belong in the reduced case. This happens because the patterns of attacking positions will always be minimal as there is no point to include, for example, the pattern 770 when the attacks happen only in 77. If the minimal patterns which contain the attacks are avoided, then patterns like 770 will be certainly avoided as well.
Moreover, the following python script is used to calculate the correlation sets of the 88 patterns as well as prepare the above matrices as string inputs for the sage package.

```
patterns=['12','13','16','17',
          '21','23','24','25','26','27',
          '31','32','33','34','35','36','37',
          '42','43','46','47',
          '52','53','56','57',
          '61','62','63','64','65','66','67',
          '71','72','73','74','75','76','77',
          '104','105','106','107',
          '114','115','144','145','154','155',
          '304','305','306','307',
          '401','403','405','407',
          '411','415',
          '441','445',
          '451','455',
          '501','503','504','505','506','507',
          '511','514','515',
          '541','544','545',
          '551','554','555',
          '601','603','605','607',
          '701','703','704','705','706','707']

def correlations( a, b ):
        set=[]
        for i in range( 1, min( len(a), len(b) ) ):
            if a[-i:] == b[:i]:
              s=a + b[i:]
              set.append(s[:-len(b)])
        print('R{'+a+',' +b+'}=',set)

        if not set:
          return '0,'
        p=''
        if len(set)==1:
          p+='x^'+str(len(set[0]))+','
          return p
        else:
          for i in range(0,len(set)):
            p+='x^'+str(len(set[i]))+'+'
          return p[:-1]+','

corr_matrix=''
for i in range(0,len(patterns)):
    for j in range(0,len(patterns)):
      corr_matrix+=correlations(patterns[j],patterns[i])
print('['+corr_matrix[:-1]+']')

matrix_1xn=''
for i in range(0,len(patterns)):
  matrix_1xn+='-x^'+str(len(patterns[i]))+','
print('['+matrix_1xn[:-1]+']')
```

```
matrix_nx1=''
for i in range(0, len(patterns)):
  matrix_nx1+='1,'
print('['+matrix_nx1[:-1]+']')
```

Now, the sage package is used for the matrix calculations. The " " parts in the following code are to be replaced by the outputs of the respective print commands from the Python script and the equation $A^{-1} = \frac{1}{det(A)}adj(A)$ is used to calculate the inverse matrix. All of these calculations are instant except for the det and adj which both require only a few minutes.

```
sage: R.<x> = PolynomialRing(ZZ)
sage: M = MatrixSpace(R,88,sparse=True)
sage: k= "correlation matrix string"
sage: l=M(k)
sage: i = matrix.identity(88,sparse=True)
sage: v=l+i
sage: z=matrix(R,1,88,"1x88 matrix string")
sage: q=matrix(R,88,1,"88x1 matrix string")
sage: a=v.adjoint()
sage: d=v.det();d
-x^14 + 2*x^13 - x^12 + 4*x^11 + 7*x^10 - 20*x^9 - x^8 - 4*x^7 - 3*x^6 +
26*x^5 + 13*x^4 - 20*x^3 - 7*x^2 + 4*x + 1
sage: C(x)=z*(1/d)*a*q;C
x |--> [(x^9 - 9*x^8 + 9*x^7 - 11*x^6 + 31*x^5 + 55*x^4 - 13*x^3 - 39*x^2)
/(x^7 - x^6 + x^5 - 5*x^4 - 7*x^3 + x^2 + 5*x + 1)]
sage: F(x)=1/(1-8*x-C(x))
sage: F.simplify_rational()
x |--> -(x^7 - x^6 + x^5 - 5*x^4 - 7*x^3 + x^2 + 5*x + 1)/(x^9 - x^8 - 2*x^6 - 10*
```

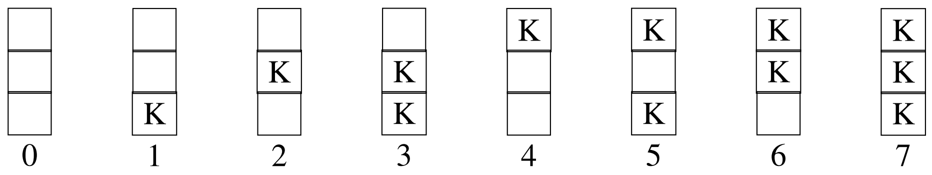
Indeed, the final output is $\mathbb{B}_3(x)$. The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$.

Number of rows n:	*Number of valid positions:*
1	*8*
2	*25*
3	*70*
4	*225*
5	*748*
6	*2401*
7	*7668*
8	*24649*
9	*79344*
10	*255025*
11	*819494*
12	*2634129*
13	*8467464*
14	*27217089*
15	*87483296*

This sequence also appears as $A287120$ in the OEIS. [9]

3.1.2 Non-Attacking Knight positions

Similarly to non-attacking Bishop positions, if we encode the knight positions in every column as:



We yield the 134 forbidden patterns:

14,15,16,17,34,35,36,37,41,43,45,47,51,53,54,55,56,57,61,63,65,67,71,73,74,75,76,77

102,112,122,132,103,113,123,133,106,126,107,127,201,211,221,231,203,213,223,233,204, 224,244,264,205,225,206,226,246,266,207,227,301,311,321,331,302,312,322,332,303,313, 323,333,304,324,305,325,306,326,307,327,402,422,442,462,403,423,406,426,446,466,407, 427,502,522,503,523,506,526,507,527,601,621,602,622,642,662,603,623,604,624,644,664, 605,625,606,626,646,666,607,627,701,721,702,722,703,723,704,724,705,725,706,726,707, 727

Therefore, by alternating the $patterns$ *array in the python script such as:*

```
patterns=['14','15','16','17',
'34','35','36','37',
'41','43','45','47',
'51','53','54','55','56','57',
'61','63','65','67',
'71','73','74','75','76','77',
'102','112','122','132',
'103','113','123','133',
'106','126',
'107','127',
'201','211','221','231','233',
'203','213','223',
'204','224','244','264',
'205','225',
'206','226','246','266',
'207','227',
'301','311','321','331',
'302','312','322','332',
'303','313','323','333',
'304','324',
'305','325',
'306','326',
'307','327',
'402','422','442','462',
'403','423',
'406','426','446','466',
'407','427',
'502','522',
'503','523',
'506','526',
'507','527',
'601','621',
'602','622','642','662',
'603','623',
'604','624','644','664',
'605','625',
'606','626','646','666',
'607','627',
'701','721',
'702','722',
'703','723',
'704','724',
'705','725',
'706','726',
'707','727']
```

Sage computes the generating function such as:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: M = MatrixSpace(R,135,sparse=True)
sage: k= "correlation matrix string"
sage: l=M(k)
sage: i = matrix.identity(135,sparse=True)
sage: v=l+i
sage: z=matrix(R,1,135,"1x135 matrix string")
sage: q=matrix(R,135,1,"135x1 matrix string")
sage: a=v.adjoint()
sage: d=v.det();d
9*x^19 - 18*x^18 - 3*x^17 + 43*x^16 - 78*x^15 + 54*x^14 - 4*x^13
- 194*x^12 + 495*x^11 - 34*x^10 - 508*x^9 + 234*x^8 - 104*x^7
+ 5*x^6 + 200*x^5 - 60*x^4 - 45*x^3 + 3*x^2 + 4*x + 1
sage: C(x)=z*(1/d)*a*q
sage: F(x)=1/(1-8*x-C(x));F
x |--> 1/(-8*x + [(252*x^16 - 468*x^15 - 528*x^14 + 552*x^13
- 816*x^12 + 1624*x^11 + 2230*x^10 - 1874*x^9 - 348*x^8 - 456*x^7
- 810*x^6 + 658*x^5 + 312*x^4 - 74*x^3 - 36*x^2 - 5*x - 1)/(36*x^15
- 72*x^14 - 60*x^13 + 72*x^12 - 120*x^11 + 250*x^10 + 270*x^9
- 256*x^8 - 30*x^7 - 78*x^6 - 98*x^5 + 92*x^4 + 36*x^3
- 8*x^2 - 5*x - 1)])
```

The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$.

Number of rows n:	*Number of valid positions:*
1	*8*
2	*36*
3	*94*
4	*278*
5	*1062*
6	*3650*
7	*11856*
8	*39444*
9	*135704*
10	*456980*
11	*1534668*
12	*5166204*
13	*17480600*
14	*58888528*
15	*198548648*

This sequence did not originally appear in the OEIS. It was added later and now appears as $A321251$ [9].

3.1.3 Avoiding two adjacent 1's

In this section, we calculate the generating function that counts the number of $3 \times n$ chessboards which avoid having two consecutive 1's in any row or column using the GJCM. By the same idea, our alphabet in this case is:

$$\begin{array}{ccccc} \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{1} \\ \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{0} \\ \boxed{0} & \boxed{1} & \boxed{0} & \boxed{0} & \boxed{1} \\ 0 & 1 & 2 & 4 & 5 \end{array}$$

Therefore, the forbidden patterns are:

11,15,22,44,45,51,54,55

By alternating the $patterns$ *array in the python script such as:*

```
patterns=['11','15',
          '22',
          '44','45',
          '51','54','55']
```

Sage computes the generating function such as:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: M = MatrixSpace(R,8,sparse=True)
sage: k=[x^1,0,0,0,0,x^1,0,0,x^1,0,0,0,0,x^1,0,0,0,0,x^1,0,0,
0,0,0,0,0,0,x^1,0,0,x^1,0,0,0,0,x^1,0,0,x^1,0,0,x^1,0,0,x^1,
0,0,x^1,0,x^1,0,0,x^1,0,0,x^1,0,x^1,0,0,x^1,0,0,x^1]
sage: l=M(k)
sage: i = matrix.identity(8,sparse=True)
sage: v=l+i
sage: z=matrix(R,1,8,[-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2])
sage: q=matrix(R,8,1,[1,1,1,1,1,1,1,1])
sage: a=v.adjoint()
sage: d=v.det();d
-x^4 + 4*x^2 + 4*x + 1
sage: C(x)=z*(1/d)*a*q;C
x |--> [(-4*x^4 + 6*x^3 + 8*x^2)/(x^3 - x^2 - 3*x - 1)]
sage: F(x)=1/(1-5*x-C(x));F
x |--> 1/(-5*x + [(4*x^4 - 5*x^3 - 9*x^2 - 3*x - 1)/(x^3 - x^2 - 3*x - 1)])
```

The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$.

Number of rows n:	*Number of valid positions:*
1	*5*
2	*17*
3	*63*
4	*227*
5	*827*
6	*2999*
7	*10897*
8	*39561*
9	*143677*
10	*521721*
11	*1894607*
12	*6879979*
13	*24983923*
14	*90725999*
15	*329460929*

This has been studied before as the number of $3 \times n(0,1)$*-matrices with no consecutive 1's in any row or column and it appears as the* $A051736$ *integer sequence in the OEIS. [9]*

3.1.4 Avoiding 2×2 blocks of 1's

Suppose that in $3 \times n$ *chessboards containing 0's and 1's we want to avoid the following block:*

1	1
1	1

By the same idea, our alphabet in this case will be:
Yielding the 7 forbidden patterns:

33,37,66,67,73,76,77

$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$	$\boxed{1}$	$\boxed{1}$
$\boxed{0}$	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$	$\boxed{0}$	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$
$\boxed{0}$	$\boxed{1}$	$\boxed{0}$	$\boxed{1}$	$\boxed{0}$	$\boxed{1}$	$\boxed{0}$	$\boxed{1}$
0	1	2	3	4	5	6	7

Similarly alternating the $patterns$ array in the python script and computing in sage:

```
patterns=['33','37','66','67','73','76','77']
```

```
sage: R.<x> = PolynomialRing(ZZ)
sage: M = MatrixSpace(R,7,sparse=True)
sage: k=[x^1,0,0,0,x^1,0,0,x^1,0,0,0,x^1,0,0,0,0,x^1,0,0,x^1,0,0,0,x^1,
0,0,x^1,0,0,x^1,0,x^1,0,0,x^1,0,x^1,0,x^1,0,0,x^1,0,x^1,0,x^1,0,0,x^1]
sage: l=M(k)
sage: i = matrix.identity(7,sparse=True)
sage: v=l+i
sage: z=matrix(R,1,7,[-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2])
sage: q=matrix(R,7,1,[1,1,1,1,1,1,1])
sage: a=v.adjoint()
sage: d=v.det();d
-x^3 + x^2 + 3*x + 1
sage: C(x)=z*(1/d)*a*q;C
x |--> [(3*x^3 - 7*x^2)/(-x^2 + 2*x + 1)]
sage: F(x)=1/(1-8*x-C(x));F
x |--> 1/(-8*x + [(-3*x^3 + 6*x^2 + 2*x + 1)/(-x^2 + 2*x + 1)])
```

The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$.

Number of rows n:	*Number of valid positions:*
1	*8*
2	*57*
3	*417*
4	*3032*
5	*22077*
6	*160697*
7	*1169792*
8	*8515337*
9	*61986457*
10	*451223152*
11	*3284626797*
12	*23910060017*
13	*174050512312*
14	*1266980540057*
15	*9222838063377*

This has been studied before as the number of $n \times 3$ binary matrices with no 2×2 block having four 1's and it's the A181246 integer sequence in the OEIS.

3.1.5 Counting each 2×2 block of 1's independently

Now, suppose that we want to count each 2×2 block of 1's with respect to what word it comes from.
The following python script computes the correlation sets as well as prepares the matrices for the sage package in order to compute the generating function $F(x, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$.

```
patterns=['33','37','66','67','73','76','77']

def correlations( a, b ):
        set=[]w
        for i in range( 1, min( len(a), len(b) ) ):
            if a[-i:] == b[:i]:
              s=a + b[i:]
              set.append(s[:-len(b)])
```

```
    print('R{'+a+',' +b+'}=',set)

    if not set:
     return '0 '
    p=''
    if len(set)==1:
     p+='x^'+str(len(set[0]))
     return p
    else:
     p='('
     for i in range(0,len(set)):
      p+='x^'+str(len(set[i]))+'+'
     return p[:-1]+')'

corr_matrix=''
for i in range(0,len(patterns)):
  for j in range(0,len(patterns)):
   index=j+1
   corr_matrix+=correlations(patterns[j],patterns[i])+'* y_'+str(index)+','
print('['+corr_matrix[:-1]+']')



matrix_1xn=''
for i in range(0,len(patterns)):
 index=i+1
 matrix_1xn+='x^'+str(len(patterns[i])) +' * y_'+str(index) + ','
print('matrix_1xn:')
print('['+matrix_1xn[:-1]+']')

matrix_nx1=''
for i in range(0, len(patterns)):
 matrix_nx1+='1,'
print('nx1:')
print('['+matrix_nx1[:-1]+']')
```

Computing in sage:

```
sage: R.<x,y_1,y_2,y_3,y_4,y_5,y_6,y_7> = PolynomialRing(ZZ)
sage: M = MatrixSpace(R,7,sparse=True)
sage: k=[x^1* y_1,0 * y_2,0 * y_3,0 * y_4,x^1* y_5,0 * y_6,
0 * y_7,x^1* y_1,0 * y_2,0 * y_3,0 * y_4,x^1* y_5,0 * y_6,
0 * y_7,0 * y_1,0 * y_2,x^1* y_3,0 * y_4,0 * y_5,x^1* y_6,
0 * y_7,0 * y_1,0 * y_2,x^1* y_3,0 * y_4,0 * y_5,x^1*y_6,
0 * y_7,0 * y_1,x^1* y_2,0 * y_3,x^1* y_4,0 * y_5,0 * y_6,
x^1* y_7,0 * y_1,x^1* y_2,0 * y_3,x^1* y_4,0 * y_5,0 * y_6,
x^1* y_7,0 * y_1,x^1* y_2,0 * y_3,x^1* y_4,0 * y_5,0 * y_6,
x^1* y_7]
sage: l=M(k)
sage: i = matrix.identity(7,sparse=True)
sage: v=i-l
sage: z=matrix(R,1,7,[x^2 * y_1,x^2 * y_2,x^2 * y_3,x^2
* y_4,x^2 * y_5,x^2 * y_6,x^2 * y_7])
sage: q=matrix(R,7,1,[1,1,1,1,1,1,1])
sage: a=v.adjoint()
sage: d=v.det();d
x^3*y_2*y_3*y_5 + x^3*y_1*y_4*y_6 - x^3*y_1*y_3*y_7
+ x^2*y_1*y_3 - x^2*y_2*y_5 - x^2*y_4*y_6 + x^2*y_1*y_7
+ x^2*y_3*y_7 - x*y_1 - x*y_3 - x*y_7 + 1
sage: C(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7)=z*(1/d)*a*q;C
(x, y_1, y_2, y_3, y_4, y_5, y_6, y_7) |-->
[(-3*x^4*y_2*y_3*y_5 - 3*x^4*y_1*y_4*y_6 + 3*x^4*y_1*y_3*y_7
- 2*x^3*y_1*y_3 - x^3*y_2*y_3 - x^3*y_1*y_4 + 2*x^3*y_2*y_5
- x^3*y_3*y_5 + x^3*y_4*y_5 - x^3*y_1*y_6 + x^3*y_2*y_6
+ 2*x^3*y_4*y_6 - 2*x^3*y_1*y_7 - 2*x^3*y_3*y_7 + x^2*y_1
+ x^2*y_2 + x^2*y_3 + x^2*y_4 + x^2*y_5 + x^2*y_6 + x^2*y_7)
/(x^3*y_2*y_3*y_5 + x^3*y_1*y_4*y_6 - x^3*y_1*y_3*y_7
+ x^2*y_1*y_3 - x^2*y_2*y_5 - x^2*y_4*y_6 + x^2*y_1*y_7
+ x^2*y_3*y_7 - x*y_1 - x*y_3 - x*y_7 + 1)]
sage: F(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7)=
1/(1-8*x-C(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7));F
(x, y_1, y_2, y_3, y_4, y_5, y_6, y_7) |--> 1/(-8*x +
[(3*x^4*y_2*y_3*y_5 + 3*x^4*y_1*y_4*y_6 - 3*x^4*y_1*y_3*y_7
+ x^3*y_2*y_3*y_5 + x^3*y_1*y_4*y_6 - x^3*y_1*y_3*y_7
+ 2*x^3*y_1*y_3 + x^3*y_2*y_3 + x^3*y_1*y_4 - 2*x^3*y_2*y_5
+ x^3*y_3*y_5 - x^3*y_4*y_5 + x^3*y_1*y_6 - x^3*y_2*y_6
- 2*x^3*y_4*y_6 + 2*x^3*y_1*y_7 + 2*x^3*y_3*y_7 + x^2*y_1*y_3
- x^2*y_2*y_5 - x^2*y_4*y_6 + x^2*y_1*y_7 + x^2*y_3*y_7
- x^2*y_1 - x^2*y_2 - x^2*y_3 - x^2*y_4 - x^2*y_5
- x^2*y_6 - x^2*y_7 - x*y_1 - x*y_3 - x*y_7 + 1)
/(x^3*y_2*y_3*y_5 + x^3*y_1*y_4*y_6 - x^3*y_1*y_3*y_7
+ x^2*y_1*y_3 - x^2*y_2*y_5 - x^2*y_4*y_6 + x^2*y_1*y_7
+ x^2*y_3*y_7 - x*y_1 - x*y_3 - x*y_7 + 1)])
```

Now using the combinatorial interpretetions we can either avoid some patterns by substituting $y_i = 0$ *or ignore counting them as patterns by substituting* $y_i = 1$. *For example, suppose that we want to count the blocks that come from the pattern* 77 *without avoiding the rest of the patterns.*

```
sage: G(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7)=1/(-8*x + ((3*x^4*y_2*y_3*y_5
+ 3*x^4*y_1*y_4*y_6 - 3*x^4*y_1*y_3*y_7 + x^3*y_2*y_3*y_5
+ x^3*y_1*y_4*y_6 - x^3*y_1*y_3*y_7 + 2*x^3*y_1*y_3 + x^3*y_2*y_3
+ x^3*y_1*y_4 - 2*x^3*y_2*y_5 + x^3*y_3*y_5 - x^3*y_4*y_5
+ x^3*y_1*y_6 - x^3*y_2*y_6 - 2*x^3*y_4*y_6 + 2*x^3*y_1*y_7
+ 2*x^3*y_3*y_7 + x^2*y_1*y_3 - x^2*y_2*y_5 - x^2*y_4*y_6
+ x^2*y_1*y_7 + x^2*y_3*y_7 - x^2*y_1 - x^2*y_2 - x^2*y_3
- x^2*y_4 - x^2*y_5 - x^2*y_6 - x^2*y_7 - x*y_1 - x*y_3
- x*y_7 + 1)/(x^3*y_2*y_3*y_5 + x^3*y_1*y_4*y_6 - x^3*y_1*y_3*y_7
+ x^2*y_1*y_3 - x^2*y_2*y_5 - x^2*y_4*y_6 + x^2*y_1*y_7 + x^2*y_3*y_7
- x*y_1 - x*y_3 - x*y_7 + 1)))
sage: P(x,y_7)=G(x,1,1,1,1,1,1,y_7)
sage: P
(x, y_7) |--> -1/(8*x - (3*x^4*y_7 - 6*x^4 - 3*x^3*y_7 - 2*x^3
- x^2*y_7 + 7*x^2 + x*y_7 + 2*x - 1)/(x^3*y_7 - 2*x^3 - 2*x^2*y_7
+ x^2 + x*y_7 + 2*x - 1))
```

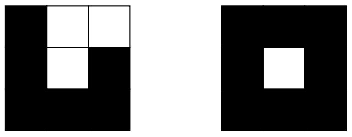
Note that, in this specific case the coefficients $[x^n {y_7}^k]F(x,1,1,1,1,1,1,y_7)$ of this generating function count the number of occurrences of the pattern 77 and not the exact number of blocks of four 1's that come from this pattern. In the pattern 77 the 2×2 block of four 1's appears twice and in every cluster string of the form $7^m, m \geq 3$ it appears $2(m-1)$ times. Therefore, the exact number will be $2k[x^n {y_7}^k]F(x,1,1,1,1,1,1,y_7)$.

3.2 Polyominoes

The polyominoes, as described by Golomb in [14], are shapes made by connecting certain numbers of equal-sized squares, each joined together with at least one other square along an edge. In this section, we will enumerate two special cases of polyominoes with their height fixed at 3 and n columns. Similarly to the previous section, each possible column will be encoded by a letter where the squares will be depicted by black squares and empty positions will be depicted by white squares.

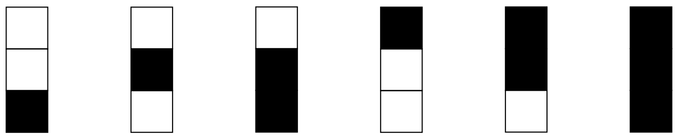
3.2.1 Column convex polyominoes

A polyomino is called column convex *if its squares that are in the same column are always connected to each other. For example:*
The left one is a column convex polyomino whereas the right one is not because of its second


(a) (b)

column.

In the $3 \times n$ polyominoes its sufficient to avoid the second row of (b) in order for it to be column convex. Therefore the alphabet in this case will be:



From which we yield the 10 forbidden patterns:

12,14,15,21,24,34,41,42,43,51

Alternating the $patterns$ array and computing in sage:

```
patterns=['12','14','15',
'21','24',
'34',
'41','42','43',
'51']
```

```
sage: R.<x> = PolynomialRing(ZZ);M = MatrixSpace(R,10,sparse=True)
sage: k=[0,0,0,x^1,0,0,x^1,0,0,x^1,0,0,0,x^1,0,0,x^1,0,0,x^1,0,
0,0,x^1,0,0,x^1,0,0,x^1,x^1,0,0,0,0,0,0,x^1,0,0,x^1,0,0,0,0,0,0,
x^1,0,0,0,0,0,0,0,0,0,0,x^1,0,0,x^1,0,0,x^1,x^1,0,0,0,0,0,x^1,0,
0,x^1,x^1,0,0,0,0,0,x^1,0,0,x^1,x^1,0,0,0,0,0,0,x^1,0,0,0,0,0,0,0]
sage: l=M(k)
sage: i = matrix.identity(10,sparse=True)
sage: v=l+i
sage: z=matrix(R,1,88,[-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,
-x^2,-x^2])
sage: z=matrix(R,1,10,[-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,
-x^2,-x^2])
```

```
sage: q=matrix(R,10,1,[1,1,1,1,1,1,1,1,1,1])
sage: a=v.adjoint()
sage: d=v.det();
sage: d
3*x^4 + 2*x^3 - 5*x^2 + 1
sage: C(x)=z*(1/d)*a*q
sage: F(x)=1/(1-6*x-C(x));F
x |--> 1/(-6*x + [(14*x^3 - 7*x^2 - x - 1)/(3*x^2 - x - 1)])
```

The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$.

Number of rows n:	*Number of valid column convex polyominoes:*
1	6
2	26
3	120
4	550
5	2526
6	11600
7	53274
8	244666
9	1123656
10	5160518
11	23700270
12	108846208
13	499888698
14	2295796202
15	10543707480

As an alternative of this enumeration problem, suppose that F*'s basic parameter does not count the length of the string, but for every letter in the string it sums the number of black squares in each letter's associated column, e.g.* $16 \to x^4$.

The generating function F now would be:

$$F(x,0,\ldots,0) = \frac{1}{1-3x-2x^2-x^3-C(x,-1,\ldots,-1)}$$

In order to compute it, the python script for forbidden patterns was alternated as follows:

```
alphabet_numsquares=[['1','1'],['2','1'],['3','2'],['4','1'],
['5','2'],['6','3']]

patterns=['12','14','15',
'21','24',
'34',
'41','42','43',
'51']



def correlations( a, b ):
        set=[]
        for i in range( 1, min( len(a), len(b) ) ):
            if a[-i:] == b[:i]:
              s=a + b[i:]
              set.append(s[:-len(b)])
        print('R{'+a+',' +b+'}=',set)

        if not set:
          return '0,'
        p=''
        if len(set)==1:
          for j in range(0,len(alphabet_numsquares)):
            if set[0]==alphabet_numsquares[j][0]:
              num=alphabet_numsquares[j][1]
          p+='x^' + num +','
          return p
        else:
          for i in range(0,len(set)):
            for j in range(0,len(alphabet_numsquares)):
              if set[i]==alphabet_numsquares[j][0]:
                num=alphabet_numsquares[j][1]
            p+='x^'+ num +'+'
          return p[:-1]+','

corr_matrix=''
for i in range(0,len(patterns)):
    for j in range(0,len(patterns)):
      corr_matrix+=correlations(patterns[j],patterns[i])
print('['+corr_matrix[:-1]+']')



matrix_1xn=''
for i in range(0,len(patterns)):
  num=0
  for letter in patterns[i]:
    for j in range(0,len(alphabet_numsquares)):
      if letter==alphabet_numsquares[j][0]:
        num+=int(alphabet_numsquares[j][1])
  matrix_1xn+='-x^'+ str(num) +' ,'
print('matrix_1xn:')
print('['+matrix_1xn[:-1]+']')

matrix_nx1=''
for i in range(0, len(patterns)):
  matrix_nx1+='1,'
print('nx1:')
print('['+matrix_nx1[:-1]+']')
```

Computing in sage:

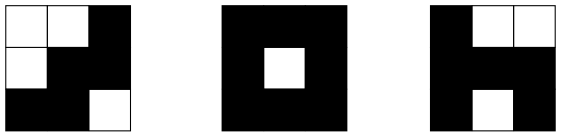
```
sage: R.<x> = PolynomialRing(ZZ);M = MatrixSpace(R,10,sparse=True)
sage: k=[0,0,0,x^1,0,0,x^1,0,0,x^2,0,0,0,x^1,0,0,x^1,0,0,x^2,0,0,0,
x^1,0,0,x^1,0,0,x^2,x^1,0,0,0,0,0,0,x^1,0,0,x^1,0,0,0,0,0,0,x^1,0,
0,0,0,0,0,0,0,0,0,x^1,0,0,x^1,0,0,x^1,x^2,0,0,0,0,0,x^1,0,0,x^1,x^2,
0,0,0,0,0,x^1,0,0,x^1,x^2,0,0,0,0,0,0,x^1,0,0,0,0,0,0,0]
sage: l=M(k)
sage: i = matrix.identity(10,sparse=True)
sage: v=l+i
sage: z=matrix(R,1,10,[-x^2 ,-x^2 ,-x^3 ,-x^2 ,-x^2 ,-x^3 ,-x^2 ,
-x^2 ,-x^3 ,-x^3])
sage: q=matrix(R,10,1,[1,1,1,1,1,1,1,1,1,1])
sage: a=v.adjoint()
sage: d=v.det();
sage: d
x^6 + 2*x^5 - 3*x^2 + 1
sage: C(x)=z*(1/d)*a*q
sage: F(x)=1/(1-3*x-2*x^2-x^3-C(x));F
x |--> 1/(-x^3 - 2*x^2 - 3*x + [(2*x^5 + 6*x^4 + 3*x^3 - 4*x^2 - x
- 1)/(x^3 + 2*x^2 - x - 1)])
```

The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$ *, where* n *is the number of black squares. Note that these are counted with respect to their coordinates on a* $3 \times k$ *board.*

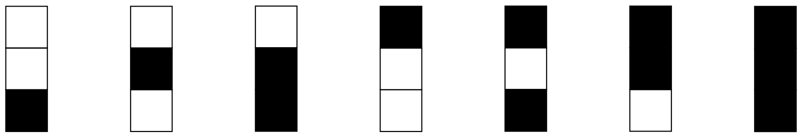
Number of squares n*:*	*Number of valid column convex polyominoes:*
1	*3*
2	*5*
3	*12*
4	*29*
5	*68*
6	*158*
7	*371*
8	*871*
9	*2043*
10	*4792*
11	*11243*
12	*26378*
13	*61886*
14	*145193*
15	*340645*

3.2.2 Directed polyominoes

A polyomino is called directed *if there is a path to every square while starting from the bottom left square and only moving towards up or right. For example:*



The polyominoes (a),(b) are directed whereas (c) isn't because any path to the bottom right square would have to move towards down.
The directed polyominoes don't have to be column convex as well, therefore the alphabet in this case is:



From which we yield the 24 forbidden patterns:

12,14,15,16,21,23,24,25,27,31,34,35,41,42,43,45,46,47,52,56,61,63,65,67

Notice that, the patterns $51, 53, 54, 55, 57$ *are excluded from the above list even though themselves are not associated to a valid directed polyomino. Notice also that, for example,* 7753 *is a valid directed polyomino. If you consider these five patterns as whole words the bottom left square has no way of reaching the upper left whereas when concatenated to a word that already is a valid directed polyomino (and the concatenation does not produce any other forbidden pattern) both squares of* 5 *will be reachable through different paths. Therefore, its required that any word* w *that is associated to a valid directed polyomino does not start with* 5.

Alternating the $patterns$ *variable and computing the generating function (it does not exclude the words that start with* 5*) in sage:*

```
patterns=['12','14','15','16',
'21','23','24','25','27',
'31','34','35',
'41','42','43','45','46','47',
'52','56',
'61','63','65','67']
```

```
sage: R.<x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,
y_10,y_11,y_12,y_13,y_14,y_15,y_16,y_17,y_18,
y_19,y_20,y_21,y_22,y_23,y_24> = PolynomialRing(ZZ)
sage: M = MatrixSpace(R,24,sparse=True)
sage: k="correlation matrix string"
sage: l=M(k)
sage: i = matrix.identity(24,sparse=True)
sage: v=i+l
sage: z=matrix(R,1,24,[-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,
-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,
-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2,-x^2])
sage: q=matrix(R,24,1,[1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1])
sage: a=v.adjoint()
sage: d=v.det();d
x^5 - 7*x^4 + 13*x^3 - 7*x^2 + 1
sage: C(x)=z*(1/d)*a*q;C
x |--> [(-7*x^6 + 52*x^5 - 107*x^4 + 80*x^3 - 24*x^2)
/(x^5 - 7*x^4 + 13*x^3 - 7*x^2 + 1)]
sage: F(x)=1/(1-7*x - C(x))
sage: F
x |--> 1/(-7*x + [(7*x^6 - 51*x^5 + 100*x^4 - 67*x^3
+ 17*x^2 + 1)/(x^5 - 7*x^4 + 13*x^3 - 7*x^2 + 1)])
```

Let W be the set of all words from the alphabet $A = \{1,2,3,4,5,6,7\}$ such that $F(x, y_1, \dots, y_{24}) = \sum_{w \in W} x^{|w|} {y_1}^{|w|_{12}} \dots {y_{24}}^{|w|_{67}}$ and $F(x,0,\dots,0)$ is the function computed in sage, where $|w|_{\tau_i}$ is the number of occurences of the i-th pattern τ_i in w. Let also W_k be the set of words from $A^*, k \in A$ such that every $w \in W_k$ starts with k.

Clearly, the desired generating function now is $G(x,0,\dots,0)$, where:

$$\begin{aligned}
G(x, y_1, \ldots, y_{24}) &= \sum_{w \in W \setminus W_5} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \\
&= \sum_{w \in W} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} - \sum_{w \in W_5} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \\
&= F(x, y_1, \ldots, y_{24}) - W_5(x, y_1, \ldots, y_{24}) \\
&= F - W_5
\end{aligned}$$

$$\begin{aligned}
W_5(x, y_1, \ldots, y_{24}) &= \sum_{w \in W_5} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \\
&= \sum_{w \in W} x^{|5w|} {y_1}^{|5w|_{12}} \ldots {y_{24}}^{|5w|_{67}} \\
&= \sum_{w \in W} x^{|w|+1} {y_1}^{|w|_{12}} \ldots {y_{18}}^{|w|_{47}} {y_{19}}^{|5w|_{52}} {y_{20}}^{|5w|_{56}} {y_{21}}^{|w|_{61}} \ldots {y_{24}}^{|w|_{67}} \\
&= x \sum_{w \in W} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{18}}^{|w|_{47}} {y_{19}}^{|w|_{52}+[w \in W_2]} {y_{20}}^{|w|_{56}+[w \in W_6]} {y_{21}}^{|w|_{61}} \ldots {y_{24}}^{|w|_{67}} \\
&= x \sum_{w \in W \setminus (W_2 \cup W_6)} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} + x y_{19} \sum_{w \in W_2} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \\
&+ x y_{20} \sum_{w \in W_6} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \\
&= x\big(\sum_{w \in W} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} - \sum_{w \in W_2} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \\
&- \sum_{w \in W_6} x^{|w|} {y_1}^{|w|_{12}} \ldots {y_{24}}^{|w|_{67}} \big) + x y_{19} W_2 + x y_{20} W_6 \\
&= xF - xW_2 - xW_6 + x y_{19} W_2 + x y_{20} W_6
\end{aligned}$$

Similarly, we get:

$$W_1 = xF - xW_2 - xW_4 - xW_5 - xW_6 + xy_1W_2 + xy_2W_4 + xy_3W_5 + xy_4W_6$$

$$W_2 = xF - xW_1 - xW_3 - xW_4 - xW_5 - xW_7 + xy_5W_1 + xy_6W_3 + xy_7W_4 + xy_8W_5 + xy_9W_7$$

$$W_3 = xF - xW_1 - xW_4 - xW_5 + xy_{10}W_1 + xy_{11}W_4 + xy_{12}W_5$$

$$W_4 = xF - xW_1 - xW_2 - xW_3 - xW_5 - xW_6 - xW_7 + xy_{13}W_1 + xy_{14}W_2 + xy_{15}W_3$$

$$+xy_{16}W_5 + xy_{17}W_6 + xy_{18}W_7$$

$$W_6 = xF - xW_1 - xW_3 - xW_5 - xW_7 + xy_{21}W_1 + xy_{22}W_3 + xy_{23}W_5 + xy_{24}W_7$$

$$W_7 = xF$$

Solving the linear system in sage:

```
sage: W_2(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_2')
sage: W_1(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_1')
sage:W_3(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_3')
sage: W_4(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_4')
sage: W_5(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_5')
sage: W_6(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_6')
sage: W_7(x,y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_10,y_11,y_12,
y_13,y_14,y_15,y_16,y_17,y_18,y_19,y_20,y_21,y_22,y_23,y_24)
=var('W_7')
```

```
sage: eqn1 = W_1== x * F - x * W_2 - x * W_4 - x * W_5 - x * W_6
+ x * y_1 * W_2 + x * y_2 * W_4 + x * y_3 * W_5 + x * y_4 * W_6
sage: eqn2 = W_2== x * F - x * W_1 - x * W_3 - x * W_4 - x * W_5
- x * W_7 + x * y_5 * W_1 + x * y_6 * W_3 + x * y_7 * W_4
+ x * y_8 * W_5 + x * y_9 * W_7
sage: eqn3 = W_3== x * F - x * W_1 - x * W_4 - x * W_5
```

```
+ x * y_10 * W_1 + x * y_11 * W_4 + x * y_12 * W_5
sage: eqn4 = W_4== x * F - x * W_1 - x * W_2 - x * W_3
- x * W_5 - x * W_6 - x * W_7 + x * y_13 * W_1 + x * y_14 * W_2
+ x * y_15 * W_3 + x * y_16 * W_5 + x * y_17 * W_6
+ x * y_18 * W_7
sage: eqn4= W_5== x * F - x * W_2 - x * W_6 + x * y_19 * W_2
+ x * y_20 * W_6
sage: eqn4= W_4== x * F - x * W_1 - x * W_2 - x * W_3 - x * W_5
- x * W_6 - x * W_7 + x * y_13 * W_1 + x * y_14 * W_2
+ x * y_15 * W_3 + x * y_16 * W_5 + x * y_17 * W_6 + x * y_18 * W_7
sage: eqn5= W_5== x * F - x * W_2 - x * W_6 + x * y_19 * W_2
+ x * y_20 * W_6
sage: eqn6= W_6== x * F - x * W_1 - x * W_3 - x * W_5 - x * W_7
+ x * y_21 * W_1 + x * y_22 * W_3 + x * y_23 * W_5 + x * y_24 * W_7
sage: eqn7= W_7== x * F

sage:solve([eqn1,eqn2,eqn3,eqn4,eqn5,eqn6,eqn7],W_1,W_2,W_3,
W_4,W_5,W_6,W_7)
```

Note that, in F the variable change $y_i \to 0$ has already taken place, nevertheless that does not affect the computations since the same variable change will be applied to the other generating functions as well in order to find $G(x, 0, \ldots, 0)$.
Now setting in sage the generating function W_5 to another function $g(x, y_1, \ldots, y_{24})$ and setting $H(x) = g(x, 0, \ldots, 0)$ we get:

```
sage: g(x,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
(2*x^10 - 17*x^9 + 45*x^8 - 37*x^7 - 20*x^6 + 49*x^5 - 30*x^4
+ 5*x^3 + 2*x^2 - x)/(2*x^10 - 23*x^9 + 107*x^8 - 274*x^7
+ 423*x^6 - 395*x^5 + 208*x^4 - 44*x^3 - 10*x^2 + 7*x - 1)
sage: h(x)=(2*x^10 - 17*x^9 + 45*x^8 - 37*x^7 - 20*x^6 + 49*x^5
- 30*x^4 + 5*x^3 + 2*x^2 - x)/(2*x^10 - 23*x^9 + 107*x^8
- 274*x^7 + 423*x^6 - 395*x^5 + 208*x^4 - 44*x^3 - 10*x^2
+ 7*x - 1)
sage: F
x |--> -1/(7*x - (7*x^6 - 51*x^5 + 100*x^4 - 67*x^3 + 17*x^2
+ 1)/(x^5 - 7*x^4 + 13*x^3 - 7*x^2 + 1))
sage: G=F-H
sage: G
x |--> -(2*x^10 - 17*x^9 + 45*x^8 - 37*x^7 - 20*x^6 + 49*x^5
- 30*x^4 + 5*x^3 + 2*x^2 - x)/(2*x^10 - 23*x^9 + 107*x^8
- 274*x^7 + 423*x^6 - 395*x^5 + 208*x^4 - 44*x^3 - 10*x^2 + 7*x
- 1) - 1/(7*x - (7*x^6 - 51*x^5 + 100*x^4 - 67*x^3 + 17*x^2
+ 1)/(x^5 - 7*x^4 + 13*x^3 - 7*x^2 + 1))
```

The following table shows the coefficients of the generating function for $n = 1, \ldots, 15$.

Number of rows n:	*Number of valid directed polyominoes:*
1	*6*
2	*20*
3	*67*
4	*218*
5	*698*
6	*2218*
7	*7021*
8	*22177*
9	*69960*
10	*220523*
11	*694774*
12	*2188256*
13	*6890762*
14	*21696127*
15	*68306623*

In the last part of this chapter we will briefly sketch two examples that apply the GJCM. In the first example the objective was achieved by trying to avoid infinite number of patterns.

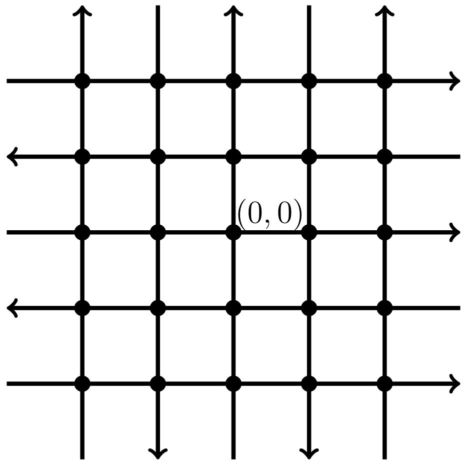
3.3 Enumeration of self-avoiding walks on the taxi-walk Manhattan lattice

In [25], Chen focuses on a certain quantity called the connective constant of the taxi-walk Manhattan lattice and one of the things he did was to improve its upper bound using the GJCM. This quantity is related to the rate at which the number of taxi-walks grows as the length of the walks increases.

It is also related in a known statistical physics model and therefore has a study interest. In order to further discuss his work and where GJCM takes place in, we need the following definitions:

Definition 3.3.1 (Self-Avoiding walks). *A self-avoiding walk on a lattice is a walk visiting any vertice of the lattice at most once.*

Definition 3.3.2 (Manhattan Lattice). *A Manhattan lattice is a directed lattice on $\mathbb{Z}^2$ where each two consecutive parallel lines alternate directions.*



An example of a Manhattan lattice.

Definition 3.3.3 (Straight and Turn). *A walk on the Manhattan lattice goes straight (resp. makes a turn) on the vertice u_i if the edges $u_{i-1}u_i$ and u_iu_{i+1} are parallel (resp. perpendicular).*

Definition 3.3.4 (Taxi-walk). *A taxi-walk is a self-avoiding walk on the Manhattan lattice that does not make two consecutive turns.*

Definition 3.3.5 (Mistake). *A mistake is a given word $w \in A^*, A = \{s, t\}$ that is to be avoided in order for a word $u \in A^*$ to describe a taxi-walk.*

Without loss of generality, suppose that the x-axis (resp. y-axis) of the Manhattan lattice is directed towards east (resp. north) and every taxi-walk starts at the origin. Now, some ideas with these notions are that the taxi-walks starting towards east and those starting towards north are symmetrical, therefore enumarating only those starting towards east is sufficient. Another idea is that, by the definition of the Manhattan lattice, every turn that can be made from a certain vertex *is already uniquely defined. Therefore all the taxi-walks that start towards east can be described now as words from the alphabet $A = \{s, t\}$. However, not every word $\in A^*$ describes a valid taxi-walk. By definition, any word that has tt as a subword does not describe a valid taxi-walk and therefore tt is a mistake.*

Proposition 11. *There are infinitely many mistakes.*

Proof. The mistake tt is sufficient in order to dissatisfy the two consecutive turns condition. The other condition is that the taxi-walk has to be a self-avoiding walk in the first place. Let w be a word $\in \{s, t\}^*$ that describes a walk on the Manhattan lattice starting towards east and let this walk end in a vertex u, visiting it for a second time while the other vertices have been visited at most once. The subword w' of w that describes the walk right after the first visit of u and until the second visit does also itself describe a same kind of walk that starts from the origin and ends on it. Therefore, any word w' that describes a walk on the Manhattan lattice that starts towards east, begins from the origin and ends in it is a mistake. It is trivial now that there are infinitely many words that describe these walks that begin and end on the origin as one instance of them would be all those that depict a square on the lattice, for all integer values the length of the side of the square can hold. $\square$

The main idea now is that the GJCM can only work for a finite set of forbidden patterns, or mistakes in this case and thus it will be overcounting the number of taxi-walks setting an upper bound. This upper bound will improve as the number of mistakes used in the method grows. Finally, after using mistakes only up to length 44 (there are 1,721,372 of them) to get the number of taxi-walks with length 802 the GJCM gave a better result for the upper bound of the connective constant than any other method had given so far.

3.4 Counting strings in DNA sequences

In the second example we present an application of pattern enumeration on how to discover statistically significant parts in DNA sequences.

3.4.1 Introduction to DNA and mutations

It is known from Campbell's biology [23] that the molecural structure of DNA, or deoxyribonucleic acid, is accounted for the ability of information storage. The DNA molecules consist of two long chains, called strands, which are arranged in a double helix. These chains are made of four types of nucleotides called adenine (A), cytosine (C), guanine (G) and thymine (T). Nucleotides are chemical building blocks [23]. The information in genes is encoded by specific sequential arrangements of these four nucleotide letters and the length of these arrangements is usually hundreds or thousands of nucleotides long [23].

Mutations are changes to the genetic information of a cell (or virus) and are accounted for the huge diversity of genes found among organisms [23]. There are large-scale and small-scale mutations. Large-scale mutations are chromosomal rearrangements from which long segments of DNA are affected [23]. Furthermore, small-scale mutations are the changes of one or a few nucleotide pairs and more specifically, point mutations are the changes in a single nucleotide pair of the gene [23]. There are two general categories for point mutations withing a gene: (1) single nucleotide-pair substitutions and (2) nucleotide-pair insertions or deletions [23]. Moreover, a point mutation can cause a heart condition, called familial cardiomyopathy, which is related to sudden death in young athletes [23]. However, point mutations leading to this disorder have already been identified in several genes [23].

3.4.2 Expected versus Actual count of strings in DNA sequences

```
with open('filepath/dna_sequence.txt','r') as dnafile:
        string = dnafile.read().replace('\n','')
sub = ['CTAG','CTAAG','CGGTAG','CTACGG','CTTAG']
def occurrences(string, sub):
    count = start = 0
    while True:
        start = string.find(sub, start) + 1
        if start > 0:
            count+=1
        else:
            return count
for i in range(len(sub)):
        print(occurrences(string,sub[i]))
```

Definition 3.4.1 (Statistically significant strings)**.** *A string w is statistically significant if the number* *of it's occurrences on a given DNA sequence differs from the expected frequency.*

One can calculate the expected frequency of a string (see 3.2) after evaluating the generating function which counts its occurrences. For example, the $ctag$ *string is statistically significant as it is well known to be under-represented in most genomes. To give more insight, in the 4,639,221–base pair sequence of* Escherichia coli K-12 *[] the* $ctag$ *string's count is 886. The generating function which counts the occurrences of* $ctag$ *in* $\{a, c, g, t\}^*$ *is* $F_{ctag} = \frac{x^4y-x^4-1}{-4x^5y+4x^5+2x^4y-2x^4+4x-1}$ *and thus:*

$$\begin{aligned}
\left.\frac{\partial F}{\partial y}\right|_{y=1} &= \frac{x^4}{(4x-1)^2} \\
&= x^4(1-4x)^{-2} \\
&= x^4 \sum_{n=0} \binom{-2}{n} (-4x)^n \\
&= x^4 \sum_{n=0} \frac{(-2)(-3)\cdots(-n-1)(-1)^n 4^n x^n}{n!} \\
&= x^4 \sum_{n=0} (-1)^{2n} \frac{2\cdot 3\cdots(n+1)4^n x^n}{n!} \\
&= \sum_{n=0} (n+1)4^n x^{n+4} \\
&= \sum_{n=4} (n-3)4^{n-4} x^n
\end{aligned}$$

Substituting x *with* $\frac{x}{4}$ *leads to:*

$$\mu(n) = \frac{n-3}{256} \quad n \geq 4$$

The expected frequency of $ctag$ *is* $\mu(4639221) \approx 18121$ *which is a lot higher than the actual count, implying it is statistically significant. A statistical significance might as well have a biological significance, regarding a certain string. For* $ctag$ *various hypotheses about it's under-representation are given in [24].*

Bibliography

[1] *F. Bassino, J. Clément, P. Nicodéme, Counting occurrences for a finite set of words: Combinatorial methods,* ACM Transactions on Algorithms **8***(3) (2012), Article 31.*

[2] *F. Bassino, J. Clément, J. Fayolle, P. Nicodème. Counting occurrences for a finite set of words: an inclusion-exclusion approach. Jacquet, Philippe. 2007 Conference on Analysis of Al- gorithms, AofA 07, 2007, Juan les Pins, France. Discrete Mathematics and Theoretical Computer Science, DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07), pp.31- 46, 2007, DMTCS Proceedings*

[3] *I. P. Goulden, D. M. Jackson, An inversion theorem for cluster decompositions of sequences with distiguished subsequences,* J. London Math. Soc. **20** *(1979), 567–576.*

[4] *I. P. Goulden, D. M. Jackson,* Combinatorial enumeration*, John Wiley & Sons, (1983).*

[5] *R. M. Low and A. Kapbasov, Non-attacking bishop and king positions on regular and cylindrical chessboards,* J. Integer Seq. **20** *(2017), Article 17.6.1, 26 pp.*

[6] *J. Noonan, D. Zeilberger, The Goulden-Jackson cluster method: Extensions, applications, and implementations,* J. Difference Eq. **5** *(1999), 355–377.*

[7] *H. Wilf,* Generatingfunctionology*, A.K.Peters, (2005).*

[8] *P. Flajolet and R. Sedgewick,* Analytic Combinatorics*, Cambridge University Press, (2013).*

[9] *N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. Published electronically at http://oeis.org, 2015.*

[10] *B. Hao, Fractals from genomes,* Mod. Phys. Lett. B*,* **14***, (2000), 871–875.*

[11] B. Hao, H. Xie, Z. Yu, and G. Chen, Avoided strings in bacterial complete genomes and a related combinatorial problem, Ann. Comb. Conference on Combinatorics and Physics (LosAlamos, NM, 1998) 4 (2000), pp. 247– 255.

[12] B. Hao, H. Xie, Z. Yu, and G.Y. Chen, Factorizable language: from dynamics to bacterial complete genomes, Physics A **288***, (2000), 10–20.*

[13] H. Xie and B. Hao, Visualization of k-tuple distribution in procaryote complete genomes and their randomized counterparts, in Computational Systems Bioinformatics Conference, International IEEE Computer Society, Los Alamitos, CA, 2002, p. 31.

*[14] S. W. Golomb,*Polyominoes: Puzzles, patterns, problems, and packings*, Princeton University Press, (1996)*

[15] L.J. Guibas and A.M. Odlyzko, String overlaps, pattern matching, and nontransitive games, J. Combin. Theory Ser. A 30 *(1981), pp. 183–208.*

[16] L.J. Guibas and A.M. Odlyzko, Periods in strings, J. Combin. Theory Ser. A 30 *(1981), pp. 19-42.*

[17] M. Régnier and Szpankowski , W. 1997. On the approximate pattern occurrences in a text. In Proceedings of the Compression and Complexity of Sequences (SEQUENCES'97) Conference. IEEE Computer Society, LOS Alamitos, CA, 253.

[18] M. Régnier and Szpankowski , W. 1998. On pattern frequency occurrences in a markovian sequence. Algorithmica 22, 4, 631–649.

[19] M. Lothaire (2005). Applied Combinatorics on Words. Encyclopedia of Mathematics, Cambridge University Press.

[20] Y. Kong (2005). Extension of Goulden-Jackson cluster method on pattern occurrences in random sequences and comparison with Régnier Szpankowski method. J. Diff. Equa. Appl. 11, 15, 1265–1271.

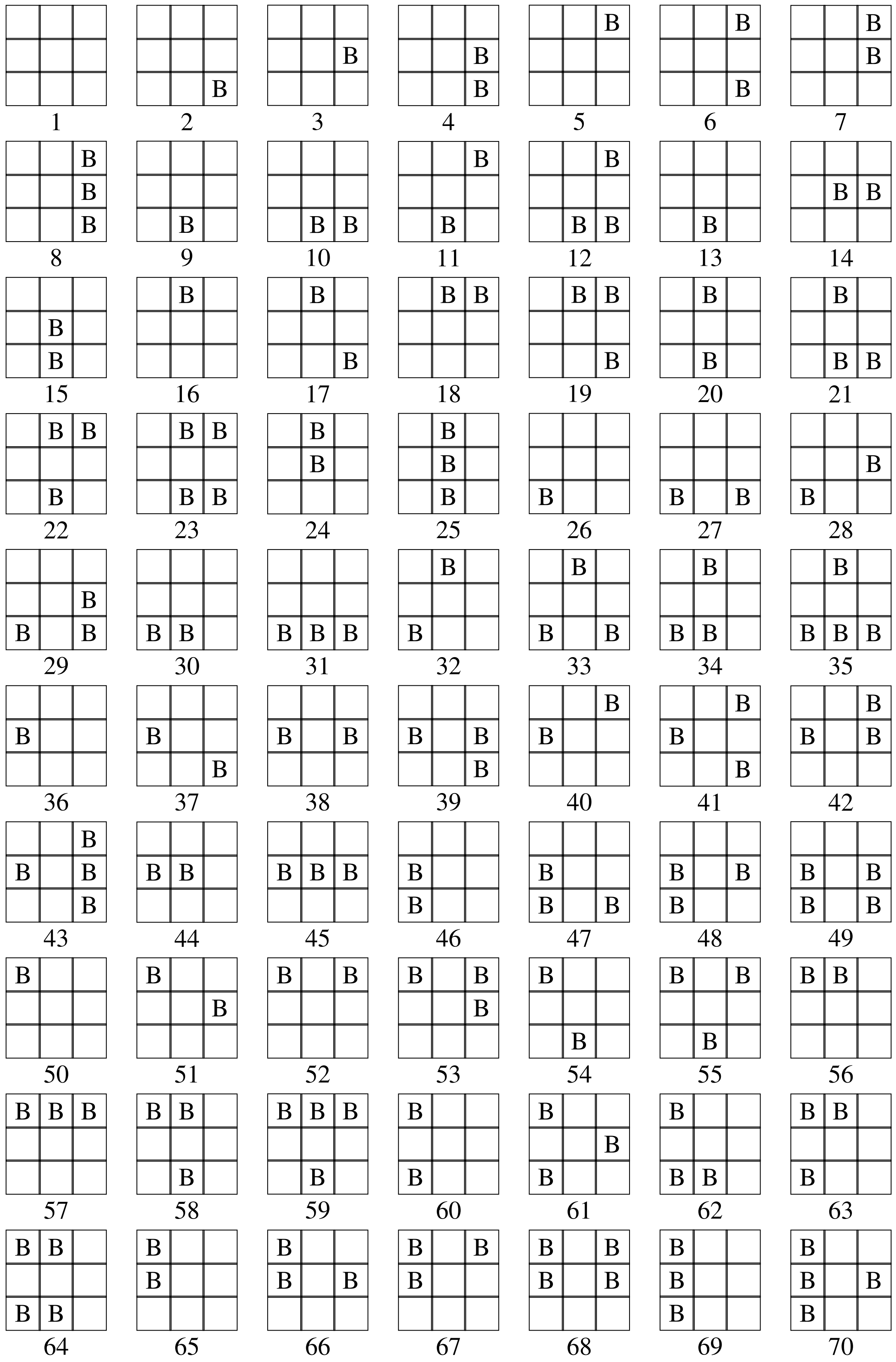
[21] A. Aho and M. Corasick (1975). Efficient string matching: An aid to bibliographic search. Comm. ACM 18, pp 333–340.

[22] R. P. Stanley.Enumerative Combinatorics, volume 1. Cambridge University Press, Cambridge, 2nd edition, 2011.

[23] J. B. Reece, N. A. Campbell (2011). Campbell biology. Boston. Benjamin Cummings/Pearson.

[24] F. R. Blattner, G. Plunkett III, C. A. Bloch, et. al. (1997). The Complete Genome Sequence of Escherichia coli K-12. Science, 277(5331), pp. 1453-1462.

[25] Y. Chen. Senior Thesis. https://www3.nd.edu/ dgalvin1/TD/EthanChenST.pdf



For $n = 3$ the 70 non-attacking Bishop positions.

1

		K

2

		K

3

		K
		K

4

		K

5

		K
		K

6

		K
		K

7

		K
		K
		K

8

	K	

9

	K	K

10

		K
	K	

11

		K
	K	K

12

	K	

13

	K	
		K

14

	K	K

15

	K	K
		K

16

		K
	K	

17

		K
	K	
		K

18

		K
	K	K

19

		K
	K	K
		K

20

	K	
	K	

21

	K	
	K	K

22

	K	K
	K	

23

	K	K
	K	K

24

	K	

25

	K	
		K

26

	K	K

27

	K	K
		K

28

	K	
	K	

29

	K	
		K
	K	

30

	K	
	K	

31

	K	
	K	K

32

	K	K
	K	

33

	K	K
	K	K

34

	K	
	K	
	K	

35

	K	
	K	K
	K	

36

K		

37

K		K

38

		K
K		

39

		K
K		K

40

K	K	

41

K	K	K

42

	K	
K		

43

	K	
K		K

44

		K
	K	
K		

45

		K
	K	
K		K

46

	K	
K	K	

47

	K	
K	K	K

48

K		

49

K		K

50

K		
	K	

51

K		K
	K	

52

K	K	

53

K	K	K

54

K	K	
	K	

55

K	K	K
	K	

56

	K	
K		

57

	K	
K		K

58

	K	
K		
	K	

59

	K	
K		K
	K	

60

	K	
K	K	

61

	K	
K	K	K

62

	K	
K	K	
	K	

63

	K	
K	K	K
	K	

64

K		
K		

65

K		
K	K	

66

K	K	
K		

67

K	K	
K	K	

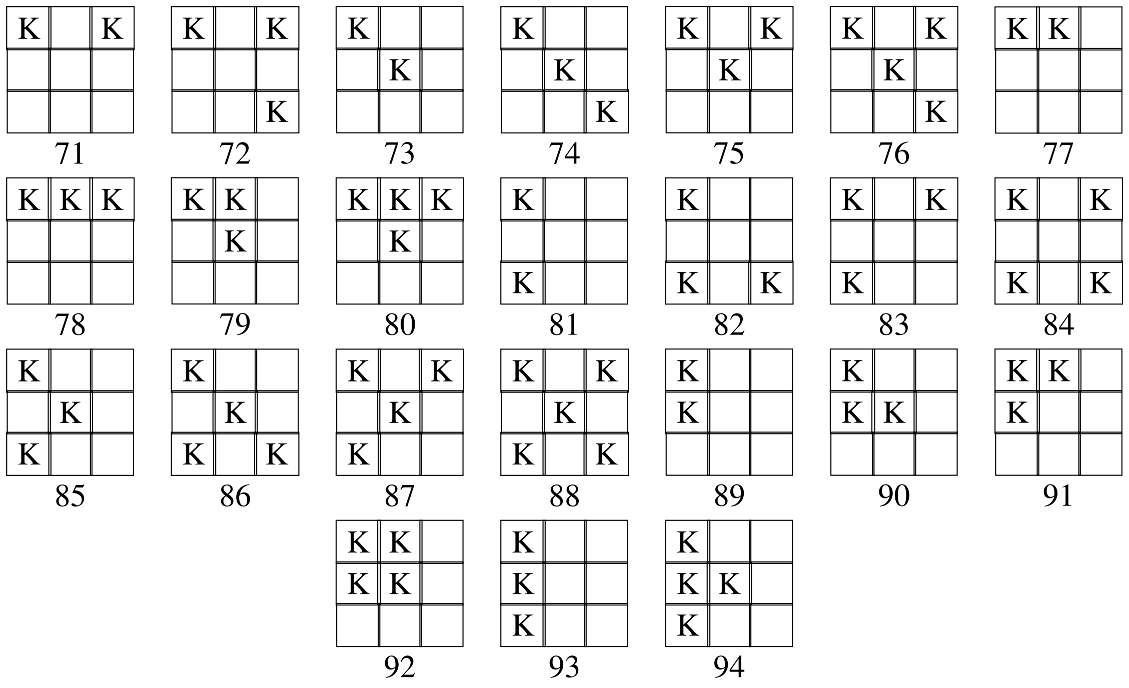
68

K		

69

K		
		K

70



For $n = 3$ the 94 non-attacking Knight positions.