Directed Column-Convex Polyominoes by Recurrence Relations

Elena Barcucci Renzo Pinzani Renzo Sprugnoli

Dipartimento di Sistemi e Informatica
Firenze, Italy

Abstract. In the literature most counting problems for directed column-convex polyominoes are solved by using Schutzenberger's method. In this paper, we use the traditional recurrence relation approach in order to count the number of directed column-convex polyominoes with a given area, the number of their columns and the number of directed column-convex polyominoes having at most k cells in the first column. This approach allows us to state a very simple algorithm for the random generation of directed column-convex polyominoes. Furthermore, directed column-convex polyominoes are considered to be structures for storing and retrieving information in a computer, and their average internal path length is then evaluated.

1 Introduction

In the vast literature concerning polyominoes, attention has been given to the so-called *directed column-convex polyominoes*, *dcc-polyominoes* or *dcc-animals* for short. Many counting problems (see Viennot [7] for an exhaustive survey) for these structures were solved by Delest and Dulucq [4] and by Barcucci, Pinzani, Rodella and Sprugnoli [1]. In the general setting of polyominoes, the qualification *directed column-convex* refers to the following characteristics: i) they are *directed* in the sense that they can be built by starting with a single cell (the *origin*) and then by adding new cells on the right or on the top of an existing cell; ii) in this construction, every column must be formed by contiguous cells.

The *area* of a dcc-polyomino is the number of its cells. In fig. 1 all the dcc-polyominoes having area $n = 1, 2, 3, 4$ are shown.

Undoubtedly, the dcc-polyominoes are one of the simplest and easiest subclasses of polyominoes to study. They are also an interesting combinatorial object and their study can constitute a first step in the analysis of many properties of polyominoes. Furthermore, they can serve as a structure for storing and retrieving information in a computer. The above-mentioned authors used Schutzenberger's method; this consists in looking for an unambiguous, context-free language, whose words have a 1–1 correspondence with dcc-polyominoes; it is then possible to derive their counting generating functions from the formal grammar. For n-area dcc-polyominoes, a very simple language can be defined by:

$$\begin{aligned} A &::= \epsilon \mid aA \\ B &::= \epsilon \mid bB \\ C &::= AcB \\ M &::= \epsilon \mid CM \\ P &::= MaA \end{aligned}$$

It is a simple matter to show that the language $L(P)$ has a 1–1 correspondence with the set of dcc-polyominoes. Every column in a dcc-polyomino (except the last one) corresponds to a (possibly empty) sequence of a, followed by a c, followed by a (possibly empty) sequence of b. Every letter corresponds to a cell and the letter c denotes the cell from which the next column starts. The last column is only composed by a. Therefore, the 13 dcc-polyominoes of area 4 correspond to the words $\{aaaa,\ cbba,\ acba,\ aaca,\ cbaa,\ acaa,\ cbca,\ acca,\ caaa,\ ccba,\ caca,\ ccaa,\ ccca\}$ (according to the ordering in Fig. 1). From this grammar, several counting problems can be easily solved.

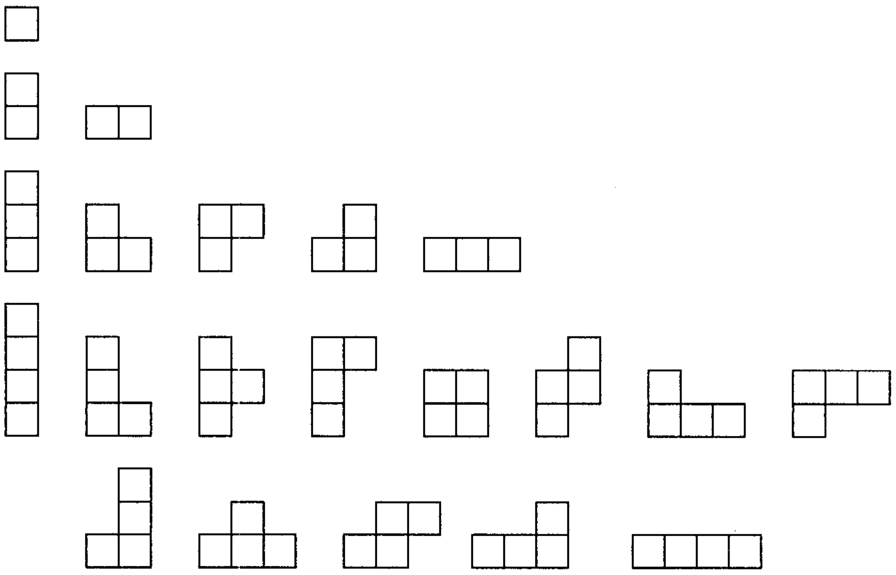

Fig. 1 - Dcc-polyominoes with area n = 1, 2, 3, 4.

In this paper, we follow a different approach. We only deal with dcc-polyominoes of a given area and derive recurrence relations in order to solve our problems. In particular these relations allow us to state a very simple linear time random generation algorithm for this kind of polyominoes. This algorithm is much simpler and more direct than the ones which can be obtained by the general algorithm proposed for regular grammars [5] or by the methods of [2, 3]. The first two results are already known, but all the others seem to be new. In Section 1, we find recurrence relations for the number of dcc-polyominoes, for the number of their columns and for the number of dcc-polyominoes having area n and k columns. In Section 2, we find the number of dcc-polyominoes having area n and k cells in the first column; this allows us to state a simple algorithm to generate random dcc-polyominoes in linear time. Finally in Section 3, we find the average internal path length of the dcc-polyominoes with area n.

2 How to Count Dcc-Polyominoes

As we shall see, the number of dcc-polyominoes and many other related quantities are strictly connected to the Fibonacci numbers {0, 1, 1, 2, 3, 5, 8, . . .}. We denote by F_n the n^{th} Fibonacci number whose generating function is:

$$\mathcal{G}\{F_n\} = F(t) = \frac{t}{1-t-t^2}$$

Note that by applying the bisection rules we find:

$$\mathcal{G}\{F_{2n}\} = \frac{t}{1-3\,t+t^2} \qquad\qquad \mathcal{G}\{F_{2n+1}\} = \frac{1-t}{1-3\,t+t^2}$$

The basic result of Delest and Dulucq [4] concerns the number V_n of dcc-polyominoes with area n. We get the same result by using the following recurrence relation:

Theorem 2.1: The number V_n of dcc-polyominoes with area n is given by the recurrence relation:

$$V_n = 1 + \sum\nolimits_{k=1}^{n-1} k\ V_{n-k}$$

having the initial condition $V_0 = 1$, given by the empty polyomino. Hence, we obtain the generating function:

$$V(t) = \frac{1-2\,t}{1-3\,t+t^2}$$

and the closed formula $V_n = F_{2n-1}$, for every $n > 0$.

Proof: Since it is obvious that $V_1 = 1$, let us suppose that $n > 1$. There is only one dcc-polyomino having a single column with n cells. For every $1 \leq k < n$, let k be the number of cells in the first column. The other $n-k$ cells constitute a dcc-polyomino which can be attached to every cell in the first column. Therefore, we have:

$$V_n = 1 + \sum\nolimits_{k=1}^{n-1} k\ V_{n-k}$$

dcc-polyominoes, and this is our recurrence. The sum can be extended from 0 to n, provided we subtract the n resulting from $k = n$ (and hence $V_{n-k}=V_0=1$). So we have:

$$V_n = 1 + \sum_{k=0}^{n} k\ V_{n-k} - n$$

and for the generating functions:

$$V(t) = \frac{1}{1-t} + \frac{t}{(1-t)^2}\ V(t) - \frac{t}{(1-t)^2}$$

This can be solved in $V(\mathrm{t})$

$$V(t) = \frac{1-2t}{1-3t+t^2}$$

Finally, we observe that $1+t\mathcal{G}\{F_{2n+1}\} = V(t)$ and this means that $V_n = F_{2n-1}$, for every $n > 0$. □

In much the same way we can count the number of dcc-polyominoes having area n and k columns:

Theorem 2.2: The number of n-area dcc-polyominoes having exactly k columns is:

$$V_{n,k} = \binom{n+k-2}{n-k}$$

Proof: The numbers $\{V_{n,k} \mid n,\ k \in \mathbb{N},\ k \leq n\}$ constitute a triangle:

$n \mid k$	0	1	2	3	4
0	1				
1	0	1			
2	0	1	1		
3	0	1	3	1	
4	0	1	6	5	1
	. . . .				

If $V_k(t)$ denotes the generating function of column k, we obviously have $V_0(t) = 1$ and $V_1(t) = \frac{t}{(1-t)}$. As in the proof of Theorem 2.1, we can now show that for $k > 0$:

$$V_{n,k+1} = \sum_{j=1}^{n-1} j\ V_{n-j,k} = \sum_{j=0}^{n} j\ V_{n-j,k}$$

In terms of generating functions, this is equivalent to:

$$V_{k+1}(t) = \frac{t}{(1-t)^2} V_k(t)$$

By using $V_1(t)$ and the induction principle, we easily find:

$$V_k(t) = \frac{t^k}{(1-t)^{2k-1}}$$

Finally, we can extract the coefficient of t^n:

$$V_{n,k} = [t^n]\, V_k(t) = [t^n]\frac{t^k}{(1-t)^{2k-1}} = \binom{-2k+1}{n-k}(-1)^{n-k} = \binom{n+k-2}{n-k}$$

In fact, this formula is valid for any $n,\ k \in \mathbb{N}$, except when $n = 1,\ k = 0$. □

It is worth noting that Theorems 2.1 and 2.2 imply $\sum_{k=1}^{n}\binom{n+k-2}{n-k} = F_{2n-1}$; hence, by first setting $h = k-1$ and then $m = n-1$, we obtain a combinatorial proof of the identity:

$$\sum_h \binom{m+h}{m-h} = F_{2m+1}$$

As it will be illustrated in the next section, an important quantity is the average number of columns in the n-area polyominoes. To find this value, let us begin by counting the total number of columns in all the n-area polyominoes.

Lemma 2.3: The total number D_n of columns in all the n-area polyominoes is defined by the recurrence:

$$D_n = 1 + \sum_{k=1}^{n-1} k(D_{n-k} + V_{n-k})$$

having the initial condition $D_0 = 0$. Hence, we have the generating function:

$$D(t) = \frac{t(1-t)^3}{(1-3t+t^2)^2}$$

and the closed formula:

$$D_n = \frac{2n-1}{5}F_{2n} - \frac{n-5}{5}F_{2n-1}$$

Proof: It is obvious that $D_0 = 0$ and $D_1 = 1$. Since there is only one dcc-polyomino with a single column of n cells for $n > 1$, let us consider the dcc-polyominoes with the first column of k cells, for $1 \leq k < n$. We can attach a dcc-polyomino of area $n-k$ to every cell of the first column, for a total of

$k\,D_{n-k}$ columns. The contribution of the first column is $k\,V_{n-k}$, and so we obtain the recurrence:

$$D_n = 1 + \sum_{k=1}^{n-1} (k\,D_{n-k} + k\,V_{n-k})$$

As in Theorem 2.1, we can extend the sum from 0 to n, provided we take into account the extra quantities we introduce. So we have:

$$D_n = 1 + \sum_{k=0}^{n} k\,D_{n-k} + \sum_{k=0}^{n} k\,V_{n-k} - n - \delta_{n0}$$

and for the generating functions:

$$D(t) = \frac{1}{1-t} + \frac{t}{(1-t)^2} D(t) + \frac{t}{(1-t)^2} V(t) - \frac{t}{(1-t)^2} - 1$$

Since $V(t)$ is known, we obtain:

$$D(t) = \frac{t(1-t)^3}{(1-3\,t+t^2)^2}$$

and by means of very simple computations or by using a computer algebra system:

$$D_n = -\frac{n}{5} F_{2n+2} + \frac{4}{5}(n-1)\, F_{2n} + \frac{8}{5} F_{2n} - F_{2n-2} = \frac{2\,n-1}{5} F_{2n} - \frac{n-5}{5} F_{2n-1}$$

This is the closed formula for D_n. □

Finally, the average number of columns is given by the following:

Theorem 2.4: The average number d_n of columns in the n-area dcc-polyominoes is:

$$d_n \sim \frac{\sqrt{5}}{5} n + \frac{9-\sqrt{5}}{10}$$

Proof: For $n > 0$, we have:

$$d_n = \frac{D_n}{V_n} = \frac{2\,n-1}{5} \frac{F_{2n}}{F_{2n-1}} - \frac{n-5}{5} \sim \frac{2\,n-1}{5} \phi - \frac{n-5}{5} = \frac{\sqrt{5}}{5} n + \frac{9-\sqrt{5}}{10}$$

where $\phi = (\sqrt{5}+1)/2$ is the golden ratio. □

The value is accurate for a small n, too. For example, for $n = 4$ the true value is $d_4 = 32/13 = 2.46153846$ and the approximate value is 2.46524758, with an error of 0.15%.

3 The Random Generation of Dcc-Polyominoes

The aim of this section is to find a method of generating a random dcc-polyomino in linear time. In other words, we wish to find out an algorithm which receives an integer n as input and gives a dcc-polyomino of area n, selected at random (with probability $1/V_n$) among all the possible n-area dcc-polyominoes, as output. We begin by counting the number of n-area dcc-polyominoes with the first column containing at most k cells.

Theorem 3.1: The number of n-area dcc-polyominoes whose first column contains exactly k cells is $c_{n,k} = k\, V_{n-k}$ for $k = 1, 2, \dots, n-1$ and $c_{n,n} = 1$. The number of n-area dcc-polyominoes whose first column contains at most k cells is:

$$G_{n,k} = V_n - V_{n-k-1} - (k+1)\, F_{2n-2k-2}$$

for $k = 1, 2, \dots, n-1$ and $F_{n,n} = V_n$.

Proof: The first part is obvious, since, as we have already observed, if the first column contains k $(k < n)$ cells, we can attach a dcc-polyomino of area $n-k$ to every one of these cells and thus obtain a total of $k\, V_{n-k}$ different n-area dcc-polyominoes. For the second part, we have:

$$G_{n,k} = \sum_{j=1}^{k} c_{n,k} = \sum_{j=1}^{k} j\, V_{n-j} = \sum_{j=0}^{k} j\, V_{n-j} = \sum_{j=n-k}^{n} (n-j)\, V_j =$$

$$= \sum_{j=0}^{n} (n-j)\, V_j - n \sum_{j=0}^{n-k-1} V_j + \sum_{j=0}^{n-k-1} j\, V_j$$

We can now evaluate these three sums by using partial fraction expansion:

$$\sum_{j=0}^{n} (n-j)\, V_j = [t^n] \frac{t}{(1-t)^2} \frac{1-2t}{1-3t+t^2} = [t^n]\left(\frac{t}{(1-t)^2} - \frac{2}{1-t} + \frac{1-2t}{1-3t+t^2}\right) =$$

$$= V_n + n + 1 - 2 = V_n + n - 1$$

$$\sum_{j=0}^{n-k-1} V_j = [t^{n-k-1}] \frac{1}{1-t} \frac{1-2t}{1-3t+t^2} = [t^{n-k-1}]\left(\frac{1}{1-t} + \frac{t}{1-3t+t^2}\right) = 1 + F_{2n-2k-2}$$

$$\sum_{j=0}^{n-k-1} j\, V_j = [t^{n-k-1}] \frac{t}{1-t} \frac{\mathrm{d}}{\mathrm{d}t} \frac{1-2t}{1-3t+t^2} = [t^{n-k-1}] \frac{t-2t^2+2t^3}{(1-t)(1-3t+t^2)^2} =$$

$$= [t^{n-k-1}]\left(\frac{1}{1-t} - \frac{1-2t}{1-3t+t^2} + \frac{t(1-t^2)}{(1-3t+t^2)^2}\right) = 1 - V_{n-k-1} + (n-k-1)\, F_{2n-2k-2}$$

In the last sum, we used the fact that:

$$\mathcal{G}\{kF_{2k}\} = t\frac{\mathrm{d}}{\mathrm{d}t}\frac{t}{1-3t+t^2} = \frac{t(1-t^2)}{(1-3t+t^2)^2}$$

By putting everything together, we eventually find:

$$G_{n,k} = V_n+n-1-n-nF_{2n-2k-2}+1-V_{n-k-1}+(n-k-1)F_{2n-2k-2} =$$

$$= V_n-V_{n-k-1}-(k+1)F_{2n-2k-2}$$

For $k = n$, we obviously have $G_{n,n} = V_n = F_{2n-1} \quad (n > 0)$ □

It is easy for us to go from frequencies to probabilities:

$$p_{n,k} = 1-\frac{V_{n-k-1}}{V_n} - (k+1)\frac{F_{2n-2k-2}}{V_n} \qquad 1 \le k \le n$$

and $p_{n,0} = 0$, $p_{n,n} = 1$. This is the probability that an n-area dcc-polyomino's first column contains k or less cells. This result suggests a simple algorithm for generating random dcc-polyominoes. We begin by extracting a uniformly distributed random number p in the range $[0, 1)$, and decide that the first column of the dcc-polyomino to be generated contains exactly k cells iff $p_{n,k-1} \le p \le p_{n,k}$. If $k = n$, we have finished; otherwise, we extract a random integer number a such that $1 \le a \le k$, and this is the cell which the rest of the dcc-polyomino has to be attached to. We go on to apply the same procedure recursively to generate a random dcc-polyomino having area $n-k$. Obviously, the dcc-polyomino is uniquely determined by the list of pairs $\{(c_1, g_1), (c_2, g_2), \ldots, (c_{b-1}, g_{b-1}), (c_b, 0)\}$, where c_i is the number of cells in the i-th column, and g_i is the cell which the next column is attached to minus 1. In Fig. 2, we show a random dcc-polyomino with area 30 generated by the algorithm and corresponding to the list {(2, 0), (2, 0), (2, 0), (1, 0), (1, 0), (1, 0), (3, 2), (1, 0), (2, 1), (3, 1), (3, 0), (1, 0), (3, 0), (1, 0), (2, 1), (2, 0)}.

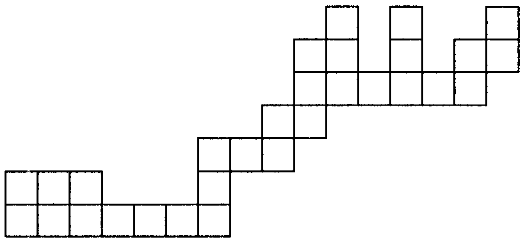

Fig. 2 - A random dcc-polyomino with area 30

To be more precise, we formulate the algorithm as a pseudo-Pascal procedure, to be called *column*(n) if we want to generate a random dcc-polyomino having area n:

```
procedure column (n: integer);
var g, c: integer; p: real;
begin   p := random;
        c := findlevel (n, p);
        if (c = n) or (c = 1) then append (c, 0)
        else begin  g := random (c); append (c, g) end;
        if c < n then column (n-c)
end {column};
```

In most Pascal compilers there is a *random* function which generates uniformly distributed pseudo-random numbers in the interval [0, 1); the same function, called with an integer argument k, generates a random integer in the range 0 . . k–1. The procedure *append* is a simple routine which appends the pair formed by its arguments to a global list; the list will eventually contain the result of our generation and is to be initialized before calling *column* for the first time. Finally, the *findlevel* function is used for finding the number of cells in the column to be generated. A possible formulation is as follows:

```
function findlevel (n: integer; p: real): integer;
var v, z: real; k: integer;
begin if n = 1 then k := 1 else begin
        v := dccpoly (n); q := v*p; k := 0;
        repeat    k := k+1;
                  if k = n then z := v else
                  z := v-dccpoly (n-k-1)-(k+1)*fibo (2*n-2*k-2)
        until z > q  end;
        findlevel := k
end {findlevel};
```

Obviously, the *fibo* and *dccpoly* functions compute the n^{th} Fibonacci number and the number of the n-area dcc-polyominoes, respectively. Actually, *dccpoly* is reduced to a call to *fibo*, except for $n = 0$ when the result is 1.

We now want to show that this program generates a dcc-polyomino in O(n) time. We must study the number of calls to three routines: *random, append* and *fibo*, which are all executed in constant time, say t_r, t_a, t_f time units, respectively. Let us observe that:

$$F_n = round\left(\frac{\phi^n}{\sqrt{5}}\right) = floor\left(\frac{1}{\sqrt{5}}\, exp(n\, log\phi)+0.5\right);$$

this takes a long time compared to *random* and *append*, but when we have fixed the precision, the evaluation time does not depend on n. If the three routine are called A_r, A_a, A_f times during the generation of a dcc-polyomino, then the total time will be:

$$T = A_r t_r + A_a t_a + A_f t_f + C$$

where C is the time taken by housekeeping operations and t_r, t_a, t_f may embody the time for loop control execution if necessary.

Let us begin by determining A_f, the number of calls to the routine *fibo*. According to the *findlevel* program above, the variable v is computed once for every column in the resulting dcc-polyomino. The variable z, instead, is computed once for every cell in a column, and then for every cell in the dcc-polyomino. So if the generated dcc-polyomino has area n and c columns, we have $A_f = c+2n$. Note that when the last column has only one cell, there is no call to *fibo* because of the initial condition in *findlevel*. The number V'_n of n-area dcc-polyominoes with the last column containing a single cell is easily evaluated. Since without this last cell we have a dcc-polyomino of area $n-1$, we find $V'_n = V_n - V_{n-1}$. By means of these facts and Lemma 2.3, we obtain the total number of calls to *fibo* for generating all the V_n dcc-polyominoes of area n: $D_n + 2n\,V_n - V_n + V_{n-1}$. Therefore, we have:

Theorem 3.2: The average number $\overline{A}_f$ of calls to the routine *fibo* to generate a random n-area dcc-polyomino is

$$\overline{A}_f \sim n\left(2+\frac{\sqrt{5}}{5}\right) - \frac{\phi}{5} + \frac{1}{\phi}$$

Proof: By using the expression for D_n found in Lemma 2.3 and dividing by V_n, we find:

$$\overline{A}_f \sim \frac{2n-1}{5}\phi - \frac{n-5}{5} + 2n-1 + \frac{1}{\phi} = n\left(2 + \frac{2\phi}{5} - \frac{1}{5}\right) - \frac{\phi}{5} + \frac{1}{\phi}$$

The constant multiplying n is about 2.4472135955. □

Theorem 3.3: The average number $\overline{A}_a$ of calls to the routine *append* to generate a random n-area dcc-polyomino is:

$$\overline{A}_a \sim \frac{\sqrt{5}}{5}n + \frac{9-\sqrt{5}}{10}$$

Proof: The *append* routine is exactly called once for every column in the generated dcc-polyomino. Therefore, we have $\overline{A}_a = d_n$ and the result follows from Theorem 2.4. □

Let us now determine the value of A_r, the number of calls to the routine *random*. The evaluation of the variable p in *column* is made once for every column in the generated dcc-polyomino. However, the evaluation of g is performed only when the column is not the last one and contains at least two cells. As a result, we have to determine the number $H_{n,k}$ of the k-cell columns in all the n-area dcc-polyominoes. We have the infinite triangle $\{H_{n,k} \mid n, k \in \mathbb{N}, k \leq n\}$:

	1	2	3	4	5	
1	1					
2	2	1				
3	6	3	1			
4	18	9	4	1		
5	53	28	12	5	1	
.	.	.	.	.	.	.

In order to determine a recurrence relation for $H_{n,k}$ $(k \leq n)$, let us consider for $j = 1, 2, \ldots, n$ the n-area dcc-polyominoes having j cells in the first column. We find a total of $\sum_{j=1}^{n-1} j\, H_{n-j,k}$ columns containing k cells; when $j = k$, we also have to count the first columns, which contribute for a total of $k\, V_{n-k}$. Hence:

$$H_{n,k} = \sum_{j=1}^{n-1} j\, H_{n-j,k} + k\, V_{n-k}$$

This completely defines the above triangle but we are only interested in the one cell columns:

Lemma 3.4: The total number H_n of one-cell columns in all the n-area dcc-polyominoes is:

$$H_n = \frac{3\,n-4}{5} F_{2n} - \frac{4\,n-10}{5} F_{2n\text{-}1}$$

Proof: By setting $k = 1$, the above recurrence becomes

$$H_n = \sum_{j=1}^{n-1} j\, H_{n-j} + V_{n\text{-}1}$$

Since $H_0 = 0$, the sum can be extended to $j = 0$ through n and we can go on to the generating functions:

$$H(t) = \frac{t}{(1-t)^2} H(t) + t\, V(t)$$

By solving in $H(t)$ and using partial fraction expansion:

$$H(t) = \frac{t(1-t)^2(1-2\,t)}{(1-3\,t+t^2)^2} = \frac{t}{5}\left(\frac{17-10\,t}{1-3\,t+t^2} - \frac{(4-11\,t)\,(3-2\,t)}{(1-3\,t+t^2)^2}\right)$$

By extracting the coefficient of t^n, after some easy computations, we find:

$$H_n = \frac{17}{5}F_{2n} - 2\,F_{2n-2} - \frac{4}{5}n\,F_{2n+2} + \frac{11}{5}(n-1)\,F_{2n} = \frac{3\,n-4}{5}F_{2n} - \frac{4\,n-10}{5}F_{2n-1}$$

□

We are now able to give the value for A_r:

Theorem 3.5: The average number $\overline{A}_r$ of calls to the *random* routine for generating a random n-area dcc-polyomino is:

$$\overline{A}_r \sim n\frac{\phi+2}{5} + \frac{2\phi}{5} - \frac{1}{\phi}$$

Proof: As we have already observed, the total number of calls to *random* for generating all the V_n n-area dcc-polyominoes is given by $D_n+(D_n-H_n-V_{n-1})$. In fact, H_n is the total number of one-cell columns and V_{n-1} represents the number of the last columns containing at least two cells. We divide by V_n and for $n > 0$ we find:

$$\overline{A}_r \sim \frac{4\,n-2}{5}\phi - \frac{2\,n-10}{5} - \frac{3\,n-4}{5}\phi + \frac{4\,n-10}{5} - \frac{1}{\phi} = n\frac{\phi+2}{5} + \frac{2\phi}{5} - \frac{1}{\phi} \quad \square$$

Obviously, the quantities t_r, t_a, t_f and C depend on the particular implementation of the algorithm, but we have now proved that the execution time is $O(n)$ and most time is spent in computing Fibonacci numbers. Some improvements can be easily conceived of. For example as predefined lists, we can code all the dcc-polyominoes having an area less than or equal to $n_0 = 5$ (say) and extract a random list whenever *column* is called with $n \leq n_0$. This and other "tricks" of the same kind, however, go beyond the aim of the present paper.

4 The Average Internal Path Length

Yuba and Hoshi [8] proposed directed polyominoes under the name of Binary Search Networks (BSN) as a structure for storing and retrieving information in a computer. The idea was to use VLSI hardware for searching in parallel along the directed, linear paths of the structure in order to minimize retrieval time. Parallelism is essential because in a traditional serial computer, BSN's cannot favourably compare with other well-known structures, such as binary search trees, with which BSN's share a common retrieving methodology. It is well-known (see, e.g., Knuth [6]) that binary search trees have an average retrieval time of order O(log n), if n is the total number of data contained in the tree. It is evident that for BSN's, the average retrieval time is much worse and is situated somewhere between $O(\sqrt{n})$ and $O(n)$. As far as we know, nobody has been able to find out the exact order, but computer experiments [2] show that it is about $O(n^{0.82})$. In the case of dcc-polyominoes, we are able to give the exact value of the average retrieval time when all the dcc-polyominoes are considered as equally probable. This value, however, cannot be extrapolated to all directed polyominoes.

We use the common terminology for binary search trees and define the *internal path length* (IPL) of a cell in a dcc-polyomino as the minimal number of steps necessary for reaching the cell starting at the origin and going from one cell to any one of the two adjacent cells. It is easy to show that there are several minimal paths of this kind, but every path contains the same number of steps. In Fig. 3, we give an example with the internal path length of every cell in the dcc-polyomino.

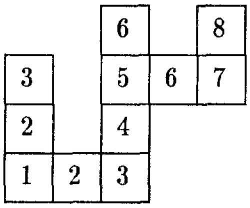

Fig. 3 - The internal path length in a dcc-polyomino

It is not very difficult to find a recurrence relation for the total IPL relative to all the n-area dcc-polyominoes. Let P_n be this quantity. By Fig. 1, we can easily find the first values: $P_1 = 1$, $P_2 = 6$, $P_3 = 29$, $P_4 = 122$ and state the following:

Theorem 4.1: The total internal path length P_n of all the V_n n-area dcc-polyominoes satisfies the recurrence relation:

$$P_n = \sum_{k=1}^{n-1} k P_{n-k} + n \sum_{k=1}^{n-1} \frac{k(k+1)}{2} V_{n-k} + \frac{n(n+1)}{2}$$

and then it is defined by the generating function:

$$P(t) = \frac{t}{(1-t)(1-3t+t^2)} + \frac{2t^2}{(1-t)(1-3t+t^2)^2} + \frac{3t^3-2t^4}{(1-t)(1-3t+t^2)^3}$$

Proof: First, let us observe that if the dcc-polyomino is reduced to a single column, then its total internal path length is $n(n+1)/2$. Let us assume that the first column contains k cells with $1 \leq k \leq n$, and let r ($1 \leq r \leq k$) be the position of the cell which the rest of the dcc-polyomino is attached to. Therefore:

i) the first column contributes for $k(k+1)/2$ to the total IPL of all the dcc-polyominoes, and it must be taken into account for each of the V_{n-k} dcc-polyominoes making up the rest of our dcc-polyomino;

ii) these V_{n-k} dcc-polyominoes have a total internal path length equal to P_{n-k};

iii) since every one of them is attached to the cell in position r, the IPL of each of their cells is increased by r, for a total contribution of $r(n-k)\, V_{n-k}$.

Therefore we have :

$$P_n = \frac{n(n+1)}{2} + \sum_{k=1}^{n-1} \sum_{r=1}^{k} \left(\frac{k(k+1)}{2} V_{n-k} + P_{n-k} + r(n-k)\, V_{n-k} \right) =$$

$$= \frac{n(n+1)}{2} + \sum_{k=1}^{n-1} \left(k P_{n-k} + \frac{k^2(k+1)}{2} V_{n-k} + (n-k)\, V_{n-k} \frac{k(k+1)}{2} \right) =$$

$$= \frac{n(n+1)}{2} + \sum_{k=1}^{n-1} k P_{n-k} + n \sum_{k=1}^{n-1} \frac{k(k+1)}{2} V_{n-k}$$

and this is the recurrence we are looking for. Obviously, $P_0 = 0$. At this point, it is not difficult to extend the sums from 0 through n, recalling that $V_0 = 1$. By adding and subtracting suitable quantities, we find:

$$P_n = \sum_{k=0}^{n} k P_{n-k} + n \sum_{k=}^{n} \frac{k(k+1)}{2} V_{n-k} + \frac{(n-1)\, n\, (n+1)}{2}$$

We can now go on to generating functions. We observe that:

$$\mathcal{G}\left\{ \sum_{k=0}^{n} k P_{n-k} \right\} = \frac{t}{(1-t)^2} P(t)$$

$$\mathcal{G}\left\{ \sum_{k=0}^{n} \binom{k+1}{2} V_{n-k} \right\} = \frac{t^2}{(1-t)^3} V(t) = \frac{t^2(1-2t)}{(1-t)^3(1-3t+t^2)}$$

$$\mathcal{G}\left\{ n \sum_{k=0}^{n} \binom{k+1}{2} V_{n-k} \right\} = t \frac{d}{dt} \frac{t^2}{(1-t)^3} V(t) =$$

$$= \frac{t(1+2t)(1-2t)}{(1-t)^4(1-3t+t^2)} + \frac{t^2(1-2t+2t^2)}{(1-t)^3(1-3t+t^2)^2}$$

$$\mathcal{G}\left\{ \binom{n+1}{3} \right\} = \frac{t^2}{(1-t)^4}$$

and these relations imply:

$$P(t) = \frac{t}{(1-t)^2} P(t) + \frac{t(1-4t^2)}{(1-t)^4(1-3t+t^2)} + \frac{t^2(1-2t+2t^2)}{(1-t)^3(1-3t+t^2)^2} - \frac{3t^2}{(1-t)^4}$$

By solving in $P(t)$ we find:

$$P(t) = \frac{t(1-4t^2)}{(1-t)^2(1-3t+t^2)^2} + \frac{t^2(1-2t+2t^2)}{(1-t)(1-3t+t^2)^3} - \frac{3t^2}{(1-t)^2(1-3t+t^2)}$$

Now we use partial fraction expansions to obtain:

$$\frac{t-4t^3}{(1-t)^2(1-3t+t^2)^2} = \frac{5-12t}{(1-3t+t^2)^2} - \frac{7-5t}{1-3t+t^2} + \frac{5}{1-t} - \frac{3}{(1-t)^2}$$

$$\frac{t^2-2t^3+2t^4}{(1-t)(1-3t+t^2)^3} = \frac{7-18t}{(1-3t+t^2)^3} - \frac{8+t}{(1-3t+t^2)^2} + \frac{2-t}{1-3t+t^2} - \frac{1}{1-t}$$

$$\frac{3t^2}{(1-t)^2(1-3t+t^2)} = \frac{3t}{1-3t+t^2} - \frac{3}{(1-t)^2} + \frac{3}{1-t}$$

Finally, by putting all these together:

$$P(t) = \frac{7-18t}{(1-3t+t^2)^3} - \frac{3+13t}{(1-3t+t^2)^2} - \frac{5-t}{1-3t+t^2} + \frac{1}{1-t}$$

which is the generating function we were looking for. □

We can now extract the coefficient of t^n from the generating function and thus obtain a closed form for the total IPL referring to all the n-area dcc-polyominoes. For $n > 0$, the number of cells in all these polyominoes is $n F_{2n-1}$, and by dividing by this quantity, we get the average IPL referring to n-area dcc-polyominoes.

Theorem 4.2: The total IPL relative to all the n-area dcc-polyominoes is:

$$P_n = \frac{n^2}{10}\left(F_{2n} + 2F_{2n-1}\right) + \frac{7n}{10}F_{2n} - F_{2n-1} + 1$$

and therefore the average internal path length is:

$$p_n \sim \frac{\phi+2}{10}n + \frac{7\phi}{10} - 1$$

Proof: The formula for P_n is a tedious exercise in coefficient extraction. In the proof of Lemma 2.3 we found:

$$[t^n]\,\frac{3-2t}{(1-3t+t^2)^2} = (n+1)\,F_{2n+4}$$

By differentiating, we obtain:

$$\frac{d^2}{dt^2}\frac{1}{1-3t+t^2} = \frac{16-18t+6t^2}{(1-3t+t^2)^3}$$

and therefore:

$$[t^n]\frac{16-18t+6t^2}{(1-3t+t^2)^3} = (n+1)(n+2)F_{2n+6}$$

By using these formulas, we can expand the terms of $P(t)$ into partial fractions and extract the coefficients of t^n:

$$\frac{7-18t}{(1-3t+t^2)^3} = \frac{1}{10}\frac{(7+18t)(16-18t+6t^2)}{(1-3t+t^2)^3} + \frac{1}{5}\frac{(9-24t)(3-2t)}{(1-3t+t^2)^2} - \frac{48}{5}\frac{1}{1-3t+t^2}$$

and then

$$[t^n]\frac{7-18t}{(1-3t+t^2)^3} = \frac{7}{10}(n+1)(n+2)F_{2n+6} - \frac{9}{5}n(n+1)F_{2n+4} +$$

$$+ \frac{9}{5}(n+1)F_{2n+4} - \frac{24}{5}nF_{2n+2} - \frac{48}{5}F_{2n+2}$$

Analogously:

$$\frac{3+13t}{(1-3t+t^2)^2} = \frac{(7-9t)(3-2t)}{(1-3t+t^2)^2} - \frac{18}{1-3t+t^2}$$

or:

$$[t^n]\frac{3+13t}{(1-3t+t^2)^2} = 7(n+1)F_{2n+4} - 9nF_{2n+2} - 18F_{2n+2}$$

Finally, we have:

$$[t^n]\frac{5-t}{1-3t+t^2} = 5F_{2n+2} - F_{2n}$$

At this point, by putting everything together and repeatedly using the recurrence relation for Fibonacci numbers $F_n = F_{n-1} + F_{n-2}$, we easily obtain the formula for P_n. The last formula is obtained dividing by nF_{2n-1} considering that $F_{2n}/F_{2n-1} \sim \phi$ and by ignoring the lower order terms. □

For dcc-polyominoes, the IPL is linear in n, which is worse than for directed polyominoes in general.

References

1. E. Barcucci, R. Pinzani, E. Rodella, R. Sprugnoli: A Characterization of Binary Search Networks. In: L. Budach (ed.): Foundamentals of Computation Theory. Lecture Notes in Computer Science 529. Berlin: Springer 1991, pp. 126-135
2. E. Barcucci, R. Pinzani, R. Sprugnoli: Génération Aléatoire des Animaux Dirigés. In: J. Labelle, J.-G. Penaud (eds.): Atelier de Combinatoire Franco-Québecois. Publications du LACIM. Montréal 1991, pp.17-25
3. E. Barcucci, R. Pinzani, R. Sprugnoli: The Random Generation of Underdiagonal Walks. In: P. Leroux, C. Reutenauer (eds.): Séries Formelles et Combinatoire Algébrique. Publications du LACIM 11. Montréal 1992, pp. 17-32
4. M. P. Delest, S. Dulucq: Enumeration of Directed Column-convex Animals with Given Perimeter and Area. Rapport LaBRI 87-15. Université de Bordeaux I 1987
5. T. Hickey, J. Cohen: Uniform Random Generation of Strings in a Context-Free Language. SIAM J. Comput. 12, 645-655 (1983)
6. D. E. Knuth: The Art of Computer Programming, Vol. I-III. Addison-Wesley, Reading, Ma, 1968-1973
7. X. G. Viennot: A Survey of Polyomino Enumeration. In: P. Leroux, C. Reutenauer (eds.): Séries Formelles et Combinatoire Algébrique. Publications du LACIM 11. Montréal 1992, pp. 399-420
8. T. Yuba, M. Hoshi: Binary Search Networks: a New Method for Key Searching. Information Processing Letters 24, 59-65 (1987)