Waiting patterns for a printer (Extended Abstract)

D. Merlini, R. Sprugnoli, M. C. Verri
Dipartimento di Sistemi e Informatica
via Lombroso 6/17, 50134, Firenze, Italia

Abstract

We introduce a model based on some combinatorial objects, which we call 1-histograms, to study the behaviour of devices like printers and use the combinatorial properties of these objects to study some important distributions such as the waiting time for a job and the length of the device queue. This study is based on an important relation between 1-histograms, generating trees and binary trees.

Keywords: binary trees; 1-histograms; generating trees; generating functions, FCFS.

1 Introduction

In this paper we present a combinatorial model for studying the characteristics of job scheduling in a slow device, which does not operate in real-time, neither uses pre-emption, i.e., when a job is started it occupies the device up to its end. A typical device of this kind is the printer in a local network: it is shared by several users, who send it their print-outs. The policy used to produce print-outs is called *First Come First Served* (FCFS) and can be realised by queuing the processes according to their arrival time and by using a FIFO algorithm. Obviously, when the printer has begun to output a file, it should print it out completely, otherwise the pages of different users are mixed.

As it is well-known, FCFS is not the best strategy for this sort of scheduling, and the average waiting time of the users can be reduced by using the *Shortest Job First* (SJF) policy (see, e.g., [1, 7]). This consists in assigning a priority to every job, and this priority is inversely proportional to the length of the print-out. In this way, when a user has been served and several jobs are present in the waiting queue, the one with the highest priority (i.e., the shortest job) is performed. Although this method is advantageous for the users as a whole, it creates some practical and psychological problems for the single user, because a job with a long output can have to wait for a long time, or also for ever if shorter outputs continuously arrive before it can be started. This phenomenon is known as *starvation* and is very annoying, although it can be solved in various ways, for example by increasing the priority of long jobs as time goes by.

Because of this drawback of SJF, FCFS is the policy usually adopted by spooling systems and our aim is to study it from a combinatorial point of view. The combinatorial approach to this kind of problems is rather new and deep results on the behaviour of these systems can be obtained in a very interesting (and funny!) way. In fact, we use both specific and classical combinatorial objects (1-histograms, binary trees, generating trees) and their enumerative properties obtained by means of generating functions and asymptotic analysis. A more classical approach to this kind of problems uses probabilistic reasoning to produce analysis based on *queuing theory* (see, e.g., [11]).

In order to explain our model, let us proceed in the following way. We suppose that a job consists in a finite number of *actions*, each of which takes constant time to be performed; this constant quantity is called *time slot*. In our favorite example, a job consists in printing a file, and an action is just the print-out of a single page. For a normal laser printer, working at 12 pages per minute (ppm), the time slot is just 5 seconds long. Actual times are obviously irrelevant, but the important fact is that every action should take the same time, so that an almost empty or a full page are both printed in 5 seconds.

The device has a buffer in which jobs are stored; we will suppose that this buffer has infinite capacity, so it can store any number of jobs and no *saturation problem* is created. Because of this hypothesis, we study the jobs queue, instead of the action queue; in any case there is not much difference between the two cases. If we fix a period of time, say n time slots, and suppose that at the end of the period the queue becomes empty, while it was never empty before, the successive states of the jobs queue are described by a combinatorial structure which will be described in Section 3. This structure has already been considered in the literature, where it is known as *staircase polyominoes* or *1-histograms* (see e.g., [3, 5]). These objects are counted by the Catalan numbers, so we have a solid base for our investigations.

In order to study all the possible schedules, we should assign a job number to every action (the job to which the action belong), so that different *configurations* correspond to a single histogram. This is modeled by a new combinatorial object, which we call *labeled* 1-*histograms*, and will be studied in Section 3. These objects are related to another well-known sequence, i.e., the Schröder numbers. Putting together known and new results, we study the average behaviour of our model and, in Section 4, we compare it against the MIN and MAX cases, i.e., the two extreme cases when, in the considered time period (n time slots, for a given n) the minimal and maximal number of jobs is served.

2 The 1-histograms

In order to introduce the concept of 1-*histograms* let us give some definitions: a *cell* is a 1×1 square and a *column* is a finite number of cells arranged one on top of the other. If we juxtapose several columns in such a way that their lowest cells are at the same level, we obtain what we call a *histogram*.

Definition 2.1 *A* 1*-histogram of length n is a histogram whose last column only contains* 1 *cell and, whenever a column is composed by k cells, then the next column contains at least $k-1$ cells. Formally, a histogram of length n can be seen as a sequence of integers $H = (h_1, h_2, \ldots, h_n)$ such that $h_n = 1$ and $\forall i \in [1..n-1]$ $h_{i+1} \geq h_i - 1$.*

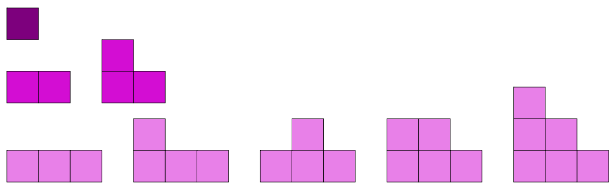

Figure 2.1: The 1-histograms with at most 3 columns.

An important concept, useful in our developments, is the concept of generating trees. It has been used from various points of view and has been introduced in the literature in [2] (without any specific name). This technique has been successively applied to other classes of permutations and the main references on the subject are due to West [9, 10]. A generating tree is a rooted labeled tree with the property that if v_1 and v_2 are any two nodes with the same label then, for each label l, v_1 and v_2 have exactly the same number of children with label l. To specify a generating tree it therefore suffices to specify: i) the label of the root; ii) a set of rules explaining how to derive from the label of a parent the labels of all of its children.

For example, Figure 2.2 illustrates the upper part of the generating tree which corresponds to the following specification:

$$\left\{\begin{array}{lll} root: & (1) & \\ rule: & (k) & \rightarrow (1)\cdots(k)(k+1) \end{array}\right. \tag{2.1}$$

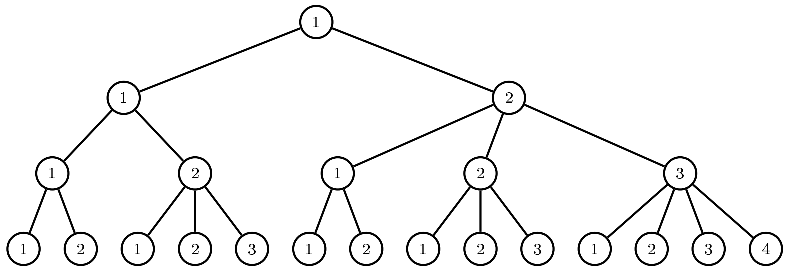

Figure 2.2: The partial generating tree for the specification (2.1).

It is at all obvious that a 1-histogram corresponds to a path in the generating tree produced by the specification (2.1). In fact, it is sufficient to associate the sequence of column heights of the 1-histogram (from right to left) to the labels in a particular path in the tree. Figure 2.1 illustrates the 1-histograms with at most 3 columns which correspond to the generating tree in Figure 2.2.

We now show a bijection between the generating tree specification (2.1) and binary trees which will be very useful in our development; to this purpose, we introduce a labeling technique by giving the definition of *e-labeled binary trees* (the *e* comes from the term *embedding* below):

Definition 2.2 *An e-labeled binary tree is a binary tree whose nodes are labeled as follows:*

1) if the binary tree is complete (that is, no subtree is empty) then we label the nodes in the following way: i) we label the root by 1; *ii) recursively, if a node is labeled k, then we label its children by $k, k+1$ proceeding from right to left (see Figure 2.3).*

2) if the binary tree is not complete we label its nodes by embedding it in a complete binary tree labeled as in 1).

Since this labeling is unique, the number of e-labeled binary trees with n nodes coincides with the number C_n of binary trees with n nodes. It is well-known that C_n corresponds to the n-th Catalan number:

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

The generating function $C(t)$ counting the number C_n of binary trees with n nodes satisfies equation:

$$C(t) = 1 + tC(t)^2, \quad \text{or} \quad C(t) = \frac{1-\sqrt{1-4t}}{2t}.$$

Theorem 2.3 *There exists a $1-1$ correspondence between the set of e-labeled binary trees with $n+1$ nodes and $(n+1)$-paths in the generating tree corresponding to the specification (2.1).*

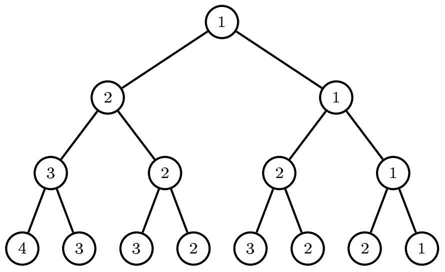

Figure 2.3: A complete e-labeled binary tree.

Proof: Let us define the procedure to go from an $(n+1)$-path $(h_0, h_1, \dots, h_n)$ in the generating tree to the corresponding e-labeled binary tree:

- the first element h_0 in the $(n+1)$-path is always 1; let it correspond to the root of the tree, labeled 1. Connect the root by means of dotted lines to its children, labeled 1 and 2 from right to left;
- after proceeding as far in the path as an element with value h_{i-1}, we will have constructed an e-labeled tree in which some of the edges have only tentative status (denoted by dotted lines instead of solid ones). Now, if h_i is the next element in the path we should have $1 \leq h_i \leq h_{i-1} + 1$: then we change the dotted line arriving to h_i (not necessarily coming from h_{i-1}) into a solid line, delete the eventual dotted line on its left and finally connect h_i by means of dotted lines to its children, labeled h_i and $h_i + 1$ from right to left.
- as a final step, when the $(n+1)$-path is exhausted, we eliminate all the dotted lines and the nodes at their ends.

The inverse correspondence is now rather obvious; suppose we have an e-labeled binary tree with $n+1$ nodes and simply perform a pre-order visit to the tree. What we get is an $(n+1)$-path in the generating tree because, by construction, its elements satisfy the condition for $(n+1)$-paths, i.e., an element h_{i-1} can only be followed by a h_i such that $1 \leq h_i \leq h_{i-1} + 1$. Obviously, different $(n+1)$-paths correspond to different e-labeled trees, and the proof is complete. ■

Some important results follow from this theorem:

Theorem 2.4 *There exists a $1-1$ correspondence between the 1-histograms with n columns and binary trees with n nodes.*

Theorem 2.5 *The 1-histograms with n-columns are counted by C_n, if we count as 1 the empty 1-histogram, i.e., the 1-histogram containing no cell.*

Theorem 2.6 *Let $S_{n,k}$ be the number of 1-histograms with n-columns having area equal to k and let $S(t,w) = \sum_{n,k} S_{n,k} t^n w^k$ the corresponding bivariate generating function. Then we have:*

$$S(t,w) = 1 + twS(t,w)S(tw,w).$$

As a consequence, we can determine the generating function $S(t)$ counting the total area S_n of 1-histograms of length n :

$$S(t) = \left.\frac{\partial S(t,w)}{\partial w}\right|_{w=1} = \frac{tC(t)}{1-4t}, \qquad S_n = \frac{1}{2}4^n - \frac{1}{2}\binom{2n}{n}.$$

Proof: From Theorem 2.4 we have that the number of 1-histograms with n-columns having area k coincides with the number of e-labeled binary trees with n nodes in which the sum of the labels is equal to k. If $S(t,w)$ is the bivariate generating function for this last quantity we have:

$$S(t,w) = 1 + twS(t,w)S(tw,w).$$

In fact, an e-labeled binary tree is empty or can be decomposed in a root (which counts as tw), and two subtrees which are an e-labeled binary tree (this contributes as $S(t,w)$) and a labeled binary tree in which each node has a label greater than one with respect to an e-labeled binary tree (this contributes as $S(tw,t)$). The results concerning $S(t)$ can be found by differentiating with respect to w the equation for $S(t,w)$ and then putting $w=1$. The value for S_n can be easily computed by extracting the n^{th} coefficient from $S(t)$. ■

In order to prove our next result, we mark the right external nodes of binary trees:

Theorem 2.7 *The number $N_{n,k}$ of 1-histograms of length n having k rises, i.e., the number of left edges in the perimeter of the 1-histograms, is equal to the number of binary trees with n internal nodes and k marked (right) external nodes. If we set $N(t,w) = \sum N_{n,k}t^n w^k$ then we have:*

$$N(t,w) = 1 + tN(t,w)(N(t,w) - 1 + w)$$

hence

$$N(t,w) = \frac{1 + t - tw - \sqrt{1 - 2t - 2tw + t^2 - 2t^2w + t^2w^2}}{2t} =$$
$$= 1 + wt + w(1+w)t^2 + w(1+3w+w^2)t^3 + w(1+6w^2+6w^3+w^4)t^4 + O(t^5),$$

i.e., $N_{n,k}$ conforms to the Narayana distribution:

$$N_{n,k} = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}, \quad n,k > 0.$$

Proof: We know that there is a bijection between histograms of length n and binary trees with n nodes, in particular with e-labeled binary trees. The same bijection distributes the histograms of length n with total number of rises $= k$ in the same way as binary trees with n nodes and k marked (right) external nodes. To prove this fact it suffices to consider the procedure to go from a 1-histogram to the corresponding e-labeled binary tree and observe that when we add an internal node at right, we don't increase the number of marked external nodes (in this case we haven't an increment of rises in the 1-histogram); on the contrary, when we add an internal node at left, we increase by one both the number of marked external nodes and the number of rises in the corresponding 1-histogram (in this case we have a "growth" of the label's value in the internal node with respect to the label's value of the father node). Now, if $N(t,w)$ is the generating function for both an external node and a non empty binary tree then $N(t,w) - 1 + w$ is the generating function for both a marked external node and a non empty binary tree and each binary tree can be decomposed by means of these configurations. ■

Corollary 2.8 *Let $N_n(w) = \sum_{k=1}^{n} N_{n,k}w^{k-1}$ be the n^{th} Narayana polynomial; then*

$$N(t,w) = 1 + \frac{1}{w}\sum_{n\geq 1} N_n(w)t^n.$$

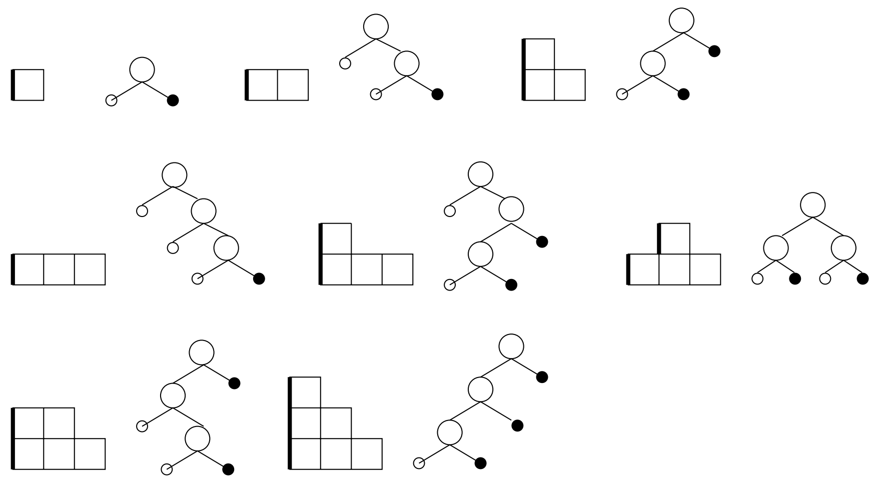

Figure 2.4: The correspondence between rises and marked (right) external nodes for $n \leq 3$.

3 The model

In order to explain our model, let us proceed in the following way. We suppose that a job consists in a finite number of *actions*, each of which takes constant time (a time slot) to be performed.

The device has a buffer of infinite capacity so it can store any number of jobs and no *saturation problem* is created. Because of this hypothesis, we study the jobs queue, instead of the action queue, although there is not much difference between the two cases. If we fix a period of time, say n time slots, and suppose that at the end of the period the queue becomes empty, while it was never empty before, the successive states of the jobs queue are described by the following combinatorial structure:

1. let us represent a job by a *cell*, i.e., a square with unitary dimensions;
2. the state of the queue at a given moment (i.e., at a given time slot) is a column of cells, containing as many cells as there are jobs in the queue;
3. the complete *history* of the queue is a sequence of columns having these two properties:
   - the last column contains a single cell, corresponding to the last job to be served;
   - if a column contains k cells, the next column contains h cells with $h \geq k-1$ ($h = k-1$ if the first job in the queue has been completely served and no new job is arrived; $h \geq k$ otherwise).

We observe explicitly that when $h \geq k$ two possibilities exist:

1. the first job in the queue has been completely served and at least a new job has been added to the queue, or
2. the first job continues to be served and
   - no new job has been inserted into the queue ($h = k$), or
   - at least one job has been inserted into the queue ($h > k$).

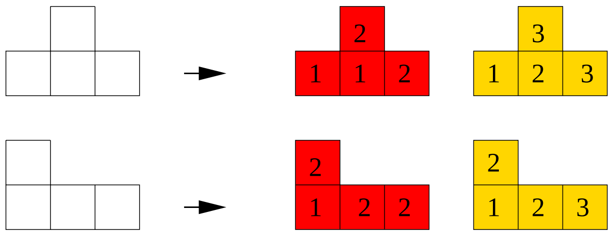

Figure 3.1: The schedules corresponding to two particular 1-histograms.

This means that, given a particular 1-histogram, it can correspond to the schedules of the printer relative to different numbers of jobs. In particular, we fill the 1-histogram with numbers that indicate in which order the jobs' requests are treated:

Definition 3.1 *A labeled* 1*-histograms of length* n *is a* 1*-histogram in which we label each cell with a number according to the following rule:*

- *the numbers in a column are consecutive and increasing from bottom up;*
- *the first column begins with a cell labeled* 1 *(jobs* $1, 2, \dots$ *are initially in the queue);*
- *if column* j *contains* k *cells and begins with a cell labeled* p*, then:*
  - *if column* $j+1$ *contains* $k-1$ *cells then its first cell is labeled* $p+1$;
  - *otherwise the first cell in column* $j+1$ *is labeled by* p *or* $p+1$.

We observe explicitly that a single 1*-histogram corresponds to several labeled* 1*-histograms.*

Figure 3.1 illustrates the possible schedules for two particular 1-histograms of length 3:

- the first one, corresponds to i) a first job which consists in printing two pages and a second job, which starts at time slot 2, and corresponds to printing a page at time slot 3, and ii) three different jobs which consists in printing a single page, the first at time slot 1, the second at time slot 2 and the third at time slot 3, after queuing at time slot 2.
- the second one, corresponds to i) a first job which consists in printing one page and a second job, which starts at time slot 1, and corresponds to printing two pages at time slots 2 and 3, and ii) three different jobs which consists in printing a single page, the first at time slot 1, the second at time slot 2, after queuing at time slot 1, and the third at time slot 3.

If we fix the number of time slots, the minimal number of jobs occurs when no job exits the printer queue at the same time as new jobs arrive and, obviously, the queue does not remain empty. The maximal number of jobs occurs when every job corresponds to printing a single page. These two particular situations will be discussed in Section 4.

Given a printer which operates for n time slots, we consider the set of labeled 1-histograms of length n and suppose that the schedules they represents are uniformly distributed. We are interested in studying the average number of jobs, $AveU_n$ the average waiting time for a job, $AveW_n$, and the average length of the queue, $AveQ_n$, among all the possible configurations of length n, that is, among all possible schedules corresponding to n time slots.

To do so, we need to know the number $\mathcal{C}_n$ of labeled 1-histograms of length n, the total number $\mathcal{U}_n$ of jobs and the total area $\mathcal{A}_n$ among all these possible configurations of length n :

$$AveU_n = \frac{(total\ \#\ of\ jobs)_n}{(\#\ of\ configurations)_n} = \frac{\mathcal{U}_n}{\mathcal{C}_n}$$

$$AveW_n = \frac{(total\ area)_n}{(total\ \#\ of\ jobs)_n} = \frac{\mathcal{A}_n}{\mathcal{U}_n}$$

$$AveQ_n = \frac{(total\ area)_n}{n \times (\#\ of\ configurations)_n} = \frac{\mathcal{A}_n}{n\mathcal{C}_n}$$

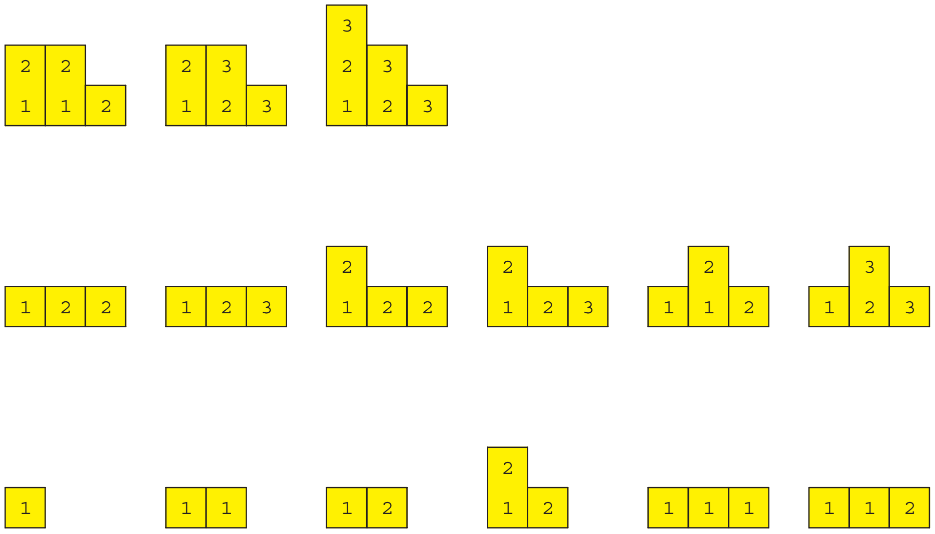

Figure 3.2: The labeled 1-histograms with $n \leq 3$.

We have now to introduce from the literature (see, e.g. [4, 8]) some concepts related to Riordan arrays. A *proper Riordan array* is an infinite lower triangular array $D = \{d_{n,k}\}_{n,k\in\mathbf{N}}$ for which two sequences $A = \{a_0, a_1, a_2, \ldots\}$ and $Z = \{z_0, z_1, z_2, \ldots\}$ exist such that:

$$d_{n+1,k+1} = a_0 d_{n,k} + a_1 d_{n,k+1} + a_2 d_{n,k+2} + \cdots = \sum_{j=0}^{\infty} a_j d_{n,k+j} \qquad \forall n, k \in \mathbf{N},$$

$$d_{n+1,0} = z_0 d_{n,0} + z_1 d_{n,1} + z_2 d_{n,2} + \cdots = \sum_{j=0}^{\infty} z_j d_{n,j} \qquad \forall n \in \mathbf{N}.$$

These sequences, and their generating functions $A(t) = \sum_k a_k t^k$ and $Z(t) = \sum_k z_k t^k$, are called the *A-sequence* and the *Z-sequence* of the proper Riordan array. Another way to characterize a proper Riordan array is through two formal power series $D = (d(t), h(t))$, for which we have:

$$d_{n,k} = [t^n] d(t)(th(t))^k \qquad \forall n, k \in \mathbf{N}.$$

There exists a simple relation between the two formal power series $h(t)$ and $A(t)$: $h(t) = A(th(t))$. Instead, $d(t)$ is simply the generating function for column 0 and is independent of $h(t)$ and $A(t)$. An important results is that the generating function for the sum $\sum_{k=0}^{n} d_{n,k} f_k$, involving the Riordan array D, is given by $d(t)f(th(t))$, $f(t)$ being the generating function for the generic sequence f_k.

Moreover, in [6] an important connection between proper Riordan arrays and generating trees is pointed out. In particular, it is proven that, under suitable conditions, a proper Riordan array

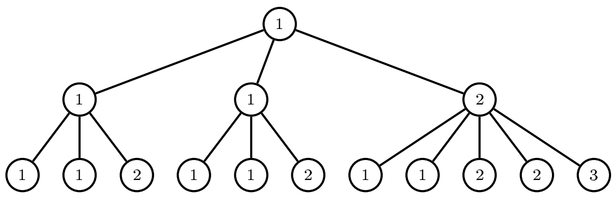

Figure 3.3: The partial generating tree for all scheduling

corresponds to a generating tree and vice versa ($d_{n,k}$ counts the number of nodes at level n with label k.).

By using the previous results and both algebraic and combinatorial reasoning we prove the following results (we omit all the proofs in this extended abstract):

Theorem 3.2 *Let $\mathcal{C}_{n,k\in\mathbf{N}}$ be the number of schedules of length n with k jobs' requests at the first time slot; then we have:*

$$\mathcal{C}_{n+1,k+1} = \mathcal{C}_{n,k} + 2\mathcal{C}_{n,k+1} + 2\mathcal{C}_{n+1,k+2} + \dots$$

and

$$\mathcal{C}_{n+1,1} = 2\mathcal{C}_{n,1} + 2\mathcal{C}_{n+1,2} + \dots,$$

that is, $\{\mathcal{C}_{n,k}\}_{n,k}$ represents a proper Riordan Array having A-sequence $A = \{1, 2, 2, \dots\}$ and Z-sequence $Z = \{2, 2, \dots\}$.

Theorem 3.3 *The possible schedules can be represented by the specification (3.2) (we associate the labels of a particular path in the tree to the sequence of column heights of the 1-histogram). Hence $\mathcal{C}_{n,k}$ represents the number of nodes at level n in the generating tree (3.2) having label k.*

$$\begin{cases} root: & (1) \\ rule: & (k) \quad \rightarrow (1)(1)\cdots(k)(k)(k+1) \end{cases} \tag{3.2}$$

In what follows, we denote by ρ the quantity $3 + 2\sqrt{2}$.

Theorem 3.4 *The number $\mathcal{C}_n$ of possible schedules are counted by the following generating function:*

$$\mathcal{C}(t) = \frac{1 - 3t - \sqrt{1 - 6t + t^2}}{4t} = t + 3t^2 + 11t^3 + 45t^4 + 197t^5 + 903t^6 + O(t^7),$$

i.e., $\mathcal{C}_n$ is the n^{th} small Schröder number. We also have:

$$\mathcal{C}_n = \sum_{k\geq 1} N_{n,k} 2^{n-k} = \sum_{k\geq 1} N_{n,k} 2^{k-1} = N_n(2), \quad \text{and} \quad \mathcal{C}_n \approx 1.435499973 \, \frac{\rho^n}{\sqrt{\pi n}(2n-1)}.$$

Theorem 3.5 *We have:*

$$\mathcal{U}_n = \sum_k N_{n,k} \sum_{j=0}^{n-k} \binom{n-k}{j} (k+j) = nN(2) - N'(2),$$

$$\mathcal{U}(t) = \frac{1 - 2t + t^2 - (1+t)\sqrt{1 - 6t + t^2}}{8t\sqrt{1 - 6t + t^2}} = t + 5t^2 + 26t^3 + 138t^4 + 743t^5 + 4043t^6 + O(t^7)$$

and

$$\mathcal{U}_n \approx \frac{\rho^n}{\sqrt{\pi n}}\left(.5075258827 - \frac{.7100533345}{(2n-1)}\right).$$

Theorem 3.6 *We have:*

$$\mathcal{A}(t) = \frac{1+t-\sqrt{1-6t+t^2}}{4(1-6t+t^2)} = t + 7t^2 + 44t^3 + 268t^4 + 1609t^5 + 9583t^6 + O(t^7),$$

and

$$\mathcal{A}_n \approx \rho^n\left(.3017766953 - \frac{.2537629413}{\sqrt{\pi n}}\right).$$

Theorem 3.7 *We have:*

$$AveU_n \approx \sqrt{2}n/2 \approx .7071067810\, n$$

$$AveW_n \approx .5946035573\sqrt{\pi n}$$

$$AveQ_n \approx .4204482076\sqrt{\pi n}$$

4 The MIN and MAX cases

In order to obtain a more accurate estimate of our quantities, we should consider minimal and maximal values relative to our model (see Figures 4.1 and 4.2). As we have already seen, the MIN and MAX cases are easily characterized and therefore we can compute the *extreme* average values assumed by our quantities and compare them against the general average values obtained in the previous section. Formally, since we are interested in studying the average number of jobs,

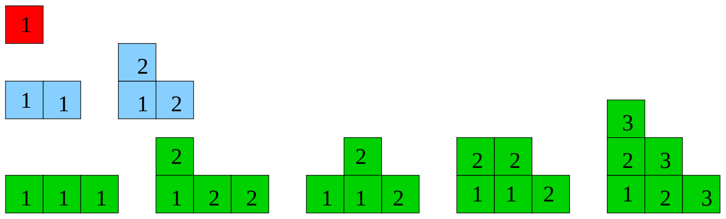

Figure 4.1: The MIN case, $n \leq 3$.

the average waiting time for a job, and the average length of the queue, among all the possible configurations of length n, in both MIN and MAX cases, let us consider a uniform distribution on 1-histograms: in both cases, at any 1-histogram corresponds a unique scheduling. The number of configurations are counted by C_n and the total area is counted by S_n.

$$AveU_n^{MIN} = \frac{U_n^{MIN}}{C_n}, \qquad AveU_n^{MAX} = \frac{U_n^{MAX}}{C_n}$$

$$AveW_n^{MIN} = \frac{S_n}{U_n^{MIN}}, \qquad AveW_n^{MAX} = \frac{S_n}{U_n^{MAX}}$$

$$AveQ_n^{MIN=MAX} = \frac{S_n}{nC_n}$$

We have the following results:

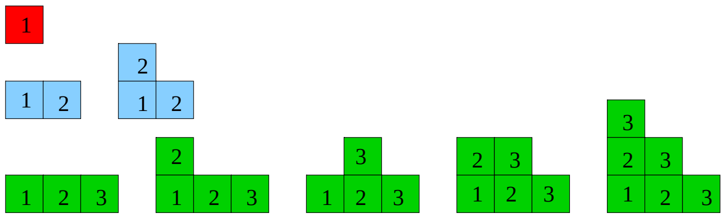

Figure 4.2: The MAX case, $n \leq 3$.

Theorem 4.1 *In the MIN case, we have* $U_n^{MIN} = \sum_k N_{n,k}$ *where* $N_{n,k}$ *is the number of* 1-*histograms corresponding to MIN number of jobs* $= k$ *(*k *is also the total number of rises). Hence:*

$$U_n^{MIN} = (2n-1)C_{n-1}.$$

Theorem 4.2 *In the MAX case, we have* $U_n^{MAX} = nC_n$.

Theorem 4.3 *We have:*

$$AveU_n^{MIN} \approx .5\,n \qquad AveU_n^{MAX} = n$$

$$AveW_n^{MIN} \approx \sqrt{\pi n}$$

$$AveW_n^{MAX} \approx .5\,\sqrt{\pi n}$$

$$AveQ_n^{MIN=MAX} \approx .5\,\sqrt{\pi n}.$$

We conclude that the following inequalities are asymptotically true:

$$AveU_n^{MIN} \leq AveU_n \leq AveU_n^{MAX}$$

$$AveW_n^{MAX} \leq AveW_n \leq AveW_n^{MIN}$$

$$AveQ_n \leq AveQ_n^{MIN=MAX}$$

Conclusions

We have proposed a combinatorial study of the FCFS policy for a slow device, typically a printer. In our model, we suppose that every sequence of job arrivals is equiprobable and that, in the period consider, the jobs queue is never empty (i.e., the device is always busy). Under these assumptions, if n is the length of the period under consideration (measured in time slots), we proved:

1. the average number of jobs served varies between $n/2$ and n, with a mean value of $\sqrt{2}n/2$; in other words, jobs served are $O(n)$;

2. the average waiting time for a job (before being completely served) varies between $\sqrt{\pi n}$ and $\sqrt{\pi n}/2$, with a mean value of $0.6\sqrt{\pi n}$; in other words, waiting time is $O(\sqrt{n})$;

3. the average length of the jobs queue has a mean value of $0.42\sqrt{\pi n}$, which is shorter than the average length in the MIN and MAX cases, where it is $\sqrt{\pi n}/2$; in any case the queue length is $O(\sqrt{n})$.

From a combinatorial point of view, it is important to observe that:

1. the objects involved in our model are histograms and labeled histograms, which are related to classical objects as binary and generating trees;
2. the quantities arising in our analysis are all related to the Catalan, Schröred and Narayana numbers.

References

[1] G. C. Buttazzo. *Sistemi in tempo reale.* Pitagora Editrice, Bologna, 1995.

[2] F. R. K. Chung, R. L. Graham, V. E. Hoggat, and M. Kleiman. The number of Baxter permutations. *Journal of Combinatorial Theory, Series A*, 24:382–394, 1978.

[3] M. P. Delest and S. Dulucq. Enumeration of directed column-convex animals with given perimeters and area. *Croat. Chem. Acta*, 66:59–80, 1993.

[4] D. Merlini. I Riordan Array nell'Analisi degli Algoritmi. Tesi di Dottorato, Università degli Studi di Firenze, 1996.

[5] D. Merlini. A generating tree approach to schedule printing devices, The Sixth Seminar on the Analysis of Algorithms, Krynica Morska, Danzica, July 2000.

[6] D. Merlini and M. C. Verri. Generating trees and proper Riordan Arrays. *Discrete Mathematics*, 218:167–183, 2000.

[7] A. Silberschat and P. Galvin. *Operating System Concepts (5th Edition).* Addison Wesley, 1998.

[8] R. Sprugnoli. Riordan arrays and combinatorial sums. *Discrete Mathematics*, 132:267–290, 1994.

[9] J. West. Generating trees and the Catalan and Schröder numbers. *Discrete Mathematics*, 146:247–262, 1995.

[10] J. West. Generating trees and forbidden subsequences. *Discrete Mathematics*, 157:363–374, 1996.

[11] R.W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice-Hall, 1989.