Generating Random Derangements*

Conrado Martínez† Alois Panholzer‡ Helmut Prodinger§

Abstract

In this short note, we propose a simple and efficient algorithm to generaterandom *derangements*, that is, permutations without fixed points. We discuss the algorithm correctness and its performance and compare it to other alternatives. We find that the algorithm has expected linear complexity, works in-place with little additional auxiliary memory and qualitatively behaves like the well-known Fisher-Yates shuffle for random permutations or Sattolo's algorithm for random cyclic permutations.

1 Introduction

Derangements are permutations without fixed points, i.e., a permutation $\pi : [1..n] \to [1..n]$ is a derangement if and only if it does not exist a value i, $1 \leq i \leq n$, such that $\pi(i) = i$. In other words, for the classical representation of a permutation as a set of cycles, a derangement is a permutation which does not contain any singleton cycle.

Derangements were first introduced by Pierre Raymond de Montmort. Their enumeration was also solved by de Montmort and about the same time by Nicholas Bernoulli [1]. Later L. Euler would give an independent proof (see [3, 4]).

The number D_n of derangements of size n is given by

(1.1)
$$D_n = n! \cdot \left[\frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!}\right] = \left\lfloor \frac{n! + 1}{e} \right\rfloor.$$

The number D_n is often written $!n$; these numbers are also called *subfactorials* [7, 8] and are a particular case of the so-called *rencontres numbers* [15]. Since $!n \approx n!/e$ the probability that a random permutation of size n is a derangement quickly converges to $1/e \approx 0.36788$.

The goal of this short note is to present a new algorithm for the generation of random derangements and provide a precise analysis of its performance. Our baseline for comparison is the combination of an efficient algorithm for the generation of random permutations, like the *Fisher-Yates shuffle* [5, pp. 26–27] (also known as *Knuth shuffle*), together with the rejection method. Since the probability that a random permutation is a derangement is roughly $1/e$, the average number of times that we have to generate a random permutation until we get a derangement is e. If we measure the complexity of our algorithms as the number of random numbers[1] that we need to generate, the algorithm sketched above has average cost $e \cdot n + o(n)$. An additional drawback of the approach using the rejection method is that it needs to check whether the generated permutation is a derangement or not.

2 The Algorithm

Our basis for random derangement generation are the modern implementation of the Fisher-Yates shuffle given by Durstenfeld [2] and later popularized by Knuth [10], and Sattolo's algorithm [16]. Sattolo's algorithm is a very simple modification of the Fisher-Yates shuffle that generates random cyclic permutations, that is, permutations that consist of a single cycle. The performance of this algorithm has been throughly analyzed in [12, 13, 18].

We modify Sattolo's algorithm so that, with appropriate probability, some elements get marked and will not be moved from their position afterwards. Each of these elements is part of a different cycle, but none of these cycles will be of length 1.

The algorithm starts filling the array A with the identity permutation, and with no marked elements. The variable u records the number of unmarked elements in the subarray $A[1..i]$.

Then we start a scanning of the array, from right to left. At iteration i, if $A[i]$ was marked in some previous iteration we just jump to the next position

*The first author was supported by the Spanish Min. of Science and Technology project TIN2006-11345 (ALINEX). The second author was supported by the Austrian Science Foundation FWF, grant S9608-N13. The third author was supported by the South African Science Foundation NRF, grant 2053748.

†Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya. E-08034 Barcelona, Spain. Email: conrado at lsi dot upc dot es.

‡Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien. Wiedner Hauptstraße 8-10/104. 1040 Wien, Austria. Email: Alois.Panholzer at tuwien dot ac dot at.

§Department of Mathematics, University of Stellenbosch. 7602 Stellenbosch, South Africa. Email: hproding at sun dot ac dot za.

[1] More precisely, random numbers of $\mathcal{O}(\log n)$ bits.

$i-1$. Otherwise, we choose at random one unmarked element, say $A[j]$, from $A[1..i-1]$. We use a simple loop to select such an element. Then $A[i]$ and $A[j]$ are swapped. But now, in order to be able to close the cycle to which j belongs, we might decide to mark the position j, so that no new element "joins" the cycle to which j belongs. We do so with some carefully chosen probability: we will show later that, indeed, $(u-1)D_{u-2}/D_u$ is the correct probability that will guarantee that any derangement is produced by the algorithm with the same probability. Then we move to the next element to the left. Since $A[i]$ was not marked, u must be decremented by one; furthermore, if during iteration i we decide to mark position $j < i$, then we must decrement u by an additional unit.

The algorithm just discussed is more formally presented as Algorithm 1.

Algorithm 1 Generation of random derangements.

```
1:  procedure RandomDerangement(n)
2:      for i ← 1 to n do A[i] ← i; mark[i] ← false
3:      i ← n; u ← n
4:      while u ≥ 2 do
5:          if ¬mark[i] then
6:              repeat j ← Random(1, i − 1)
7:              until ¬mark[j]
8:              A[i] ↔ A[j]
9:              p ← Uniform(0, 1)
10:             if p < (u − 1)D_{u−2}/D_u then
11:                 mark[j] ← true; u ← u − 1
12:             u ← u − 1
13:         i ← i − 1
14:     return A
```

2.1 Correctness By construction, Algorithm 1 produces always derangements. The first observation is that the elements in $A[i+1..n]$ never get involved in any further swap from iteration i downwards. Thus if a value $k > i$ has been moved to $A[1..i]$ during some previous iteration it will never be moved back to its original position. In this respect, our algorithm shares this important property with its close relatives Fisher-Yates' and Sattolo's: any given item can move some number of times to the left (i.e., to positions with lower index), but it can move to the right only once. The second observation is that a marked element is never sitting at its original position: we only mark elements once they have been swapped and then they never move again. Last but not least, as in Sattolo's algorithm, a swap involves $A[i]$ and $A[j]$ with $i \neq j$, thus we cannot create singleton cycles (compare to Fisher-Yates' algorithm, where a swap $A[i] \leftrightarrow A[i]$ in iteration i occurs with probability $1/i$).

To conclude the proof of correctness of Algorithm 1, we show that each derangement of size n has probability $1/D_n$ of being produced by the algorithm. The key identity to prove this is the recurrence for D_n:

$$D_n = (n-1)(D_{n-1} + D_{n-2}), \quad n \geq 2,$$

with $D_0 = 1$ and $D_1 = 0$.

We prove this by induction on n. Suppose $u = n = 2$. Then $A[1] = 1$ is swapped with $A[2] = 2$ and since $D_0 = D_2 = 1$, $A[1]$ is marked and we update $u := 0$, so nothing else happens and the algorithm has generated $A = [2, 1]$, the unique derangement of size 2—with probability 1.

For $u = n > 2$, the algorithm will probabilistically decide whether n is part of a cycle of two elements or not. In the former case, which has probability $(u-1)D_{u-2}/D_u$, the algorithm has choosen one of the remaining $u-1$ elements, say $A[j]$, and swaps $A[j]$ and $A[n]$ to construct the 2-cycle. Then position j is marked so that the element sitting there does not get involved in any further step of the algorithm, and we "recursively" generate a random derangement of size $u = n-2$ with the other elements. In the second case, which has probability $1 - \frac{(u-1)D_{u-2}}{D_s}$, we might think as if the algorithm has "reserved" uniformly at random one of the $u-1$ available slots, say j, and then recursively generates a random derangement of size $u = n-1$ with the remaining elements. The swap between $A[j]$ and $A[n]$ corresponds to the insertion of n into the slot we had "reserved" previously.

Thus, the probability that we generate a particular derangement is given either by

$$(u-1)\frac{D_{u-2}}{D_u}\frac{1}{u-1}\frac{1}{D_{u-2}} = \frac{1}{D_u},$$

or by

$$\left(1 - (u-1)\frac{D_{u-1}}{D_u}\right)\frac{1}{u-1}\frac{1}{D_{u-1}} = \frac{1}{D_u},$$

as we wanted to prove.

Algorithm 1 can be seen as a clever implementation of the straightforward random generation based on the recursive method (see for instance [6, 14, 17]). The recursive method needs to generate $2n$ random numbers, but it is only well suited if we represented a permutation as a set of cycles, using some linked data structure to store them.

Our algorithm is equivalent to the one stemming from the recursive method, but we have removed recursion, the derangement is stored in an array and it is generated *in-situ* without need of additional data structures other than the n bits for marks and some table to

store the D_n values (which the pure recursive method needs too). Also, if the elements which we wanted to "derange" were the numbers 1 to n, we could dispense the array of marks altogether; we could change the sign of $A[i]$ at iteration $j > i$ to indicate that it is marked, and multiply it by -1 again at iteration i (since we do not need that mark any longer).

The significant difference of our method with respect to the pure recursive method is that to find non-marked j's we are using a simple generate-and-reject loop (lines 6-7 of Algorithm 1). This loop will perform a single iteration most of the times, but it might generate many unusable values in a given round, so that the cost of our algorithm might be, in principle, very large.

3 Analysis of the performance

We measure the complexity of our algorithms by the number of times that we need to generate random numbers. Under this model, the complexity of Fisher-Yates shuffle and Sattolo's algorithm is obviously n, but the loop in lines 6 to 7 make the analysis of Algorithm 1 not entirely trivial. Additionally, the running time of any of these algorithms is clearly and directly related to the number of random numbers used.

Notice that if $u = 0$ and $i > 1$, all iterations from that point on would simply scan the leftmost i marked elements, had we not stopped the main loop of the algorithm. Hence, Algorithm 1 is equivalent to one where the outer loop stops when $i = 1$. The algorithm generates a uniform random number in $(0, 1)$ at each iteration such that the corresponding $A[i]$ is not marked (line 9 of Algorithm 1). Since the number of marked elements is the number of cycles in the derangement, the expected number of calls is $n - \mathbb{E}[C_n]$, where C_n denotes the number of cycles in a random derangement.

So we concentrate from now on in the number of random integers generated in line 6. Let us denote this number by G, and let G_i denote the number of random numbers generated during iteration i. Then we simply have

$$\mathbb{E}[G] = \sum_{i=2}^{n} \mathbb{E}[G_i]\,,$$

by linearity of expectations. On the other hand, if $A[i]$ were marked we just jump to the next iteration and $G_i = 0$. Let M_i be the indicator random variable such that $M_i = 1$ if $A[i]$ gets marked by the algorithm and $M_i = 0$ otherwise. Hence,

$$\mathbb{E}[G] = \sum_{i=2}^{n} \mathbb{E}[G_i \mid M_i = 0]\,\mathbb{P}[M_i = 0]\,.$$

Let U_i denote the value of u at the beginning of iteration i. If $A[i]$ is not marked then at iteration i we have to choose one of the $U_i - 1$ unmarked elements among the $i - 1$ elements to the left of $A[i]$. The number G_i is clearly geometrically distributed, with $(U_i - 1)/(i - 1)$ the probability of success. So we have

$$\mathbb{E}[G_i \mid M_i = 0] = \mathbb{E}\left[\frac{i-1}{U_i - 1} \mid M_i = 0\right].$$

Looking back again to Algorithm 1 we might say that we decrement u in every iteration and, in addition, we add to u some Δ_i which might be -1 (if some $A[j]$ gets marked), $+1$ (if $A[i]$ was already marked) or 0 (otherwise). That is, the dynamics of U_i is given by $U_i = U_{i+1} - 1 + \Delta_{i+1}$ if $i < n$ and $U_n = n$.

Unwinding the recursion we get

$$U_i = i + \Delta_{i+1} + \cdots + \Delta_n$$

and defining $B_i := -(\Delta_i + \cdots + \Delta_n)$ we finally obtain

$$U_i + B_{i+1} = i.$$

In other words, B_{i+1} denotes the number of marked elements in $A[1..i]$ when our algorithm reaches iteration i. The dynamics of B_i is also simple: $B_i = B_{i+1} - \Delta_i$, with $B_{n+1} = 0$. For this reason, we may also think of B_i as a "balance". Once we introduce B_i we can write $\mathbb{E}[G]$ as

$$\mathbb{E}[G] = \sum_{i=2}^{n} \mathbb{E}\left[\frac{i-1}{i-1-B_{i+1}} \mid M_i = 0\right] \mathbb{P}[M_i = 0]\,.$$

Why do we prefer to write $\mathbb{E}[G]$ in terms of B_i's rather than in terms of S_i's? To answer this question we must consider the canonical cycle representation of permutations. In this form, the cycles are listed in decreasing order of the cycle leaders, and each cycle is listed so that the leader is its first element. In order to understand better some important facts that relate Δ_i and M_i to the cycle decomposition of the generated derangement, it is useful to look at some particular example, such as the one shown in Figure 1. Each row shows the contents of the array at the beginning of the corresponding iteration, the number i of the iteration and the value of the variable u at the start of the iteration. We use a circle to indicate that a particular element is marked, and we show the usual representation of the generated derangement as a set of cycles below.

PROPOSITION 3.1. *For any derangement π of size n, and any i, $1 \le i \le n$, the following properties hold:*

1. *$M_i = 1$ if and only if i is the leader of some cycle in the generated permutation.*

Permutation												i	U_i	Δ_i	B_{i+1}
1	2	3	4	5	6	7	8	9	10	11	12	12	12	−1	0
1	2	3	4	5	6	7	12	9	10	11	8	11	10	0	1
1	2	3	4	11	6	7	12	9	10	5	8	10	9	−1	1
1	2	3	4	11	6	10	12	9	7	5	8	9	7	0	2
9	2	3	4	11	6	10	12	1	7	5	8	8	6	+1	2
9	2	3	4	11	6	10	12	1	7	5	8	7	6	+1	1
9	2	3	4	11	6	10	12	1	7	5	8	6	6	0	0
6	2	3	4	11	9	10	12	1	7	5	8	5	5	−1	0
11	2	3	4	6	9	10	12	1	7	5	8	4	3	0	1
11	2	4	3	6	9	10	12	1	7	5	8	3	2	−1	1
11	4	2	3	6	9	10	12	1	7	5	8	2	0	+1	2
11	4	2	3	6	9	10	12	1	7	5	8	1	0	+1	1

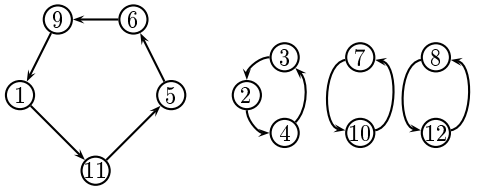

Figure 1: A sample execution of Algorithm 1 and its output.

2. $\Delta_i = 1$ *if* i *is the leader of some cycle in the permutation.*

3. $\Delta_i = -1$ *if* i *is the second smallest element of its cycle.*

4. $\Delta_i = 0$ *if* i *is neither the smallest nor the second smallest element of its cycle.*

5. $B_1 = 0$, $B_2 = 1$.

6. $B_i \leq C_n$, *where* C_n *denotes the number of cycles in a random derangement.*

7. $0 \leq B_{i+1} \leq i$.

8. *If* $M_i = 0$ *then* $B_{i+1} < i - 1$. *Moreover, for any* i, $1 \leq i \leq n$, $B_{i+1} \neq i - 1$.

Proof.

1. Suppose $M_i = 1$, that is, by definition, that $A[i]$ is marked. A moment's thought reveals that there can't be any $j < i$ such that $A[j] = i$. In other words, no element $j < i$ can become part of the cycle to which i belongs—recall that a given item can move several times to the left of the array, but only once to the right—. Conversely, if i is the leader of its cycle, then we have to "close" its cycle at iteration i, not before, not after. Suppose we mark the cycle at some iteration j (necessarily $j \neq i$), then $A[j]$ is swapped to some other place say $\ell < j$, so we mark $A[\ell]$. No element at a position smaller than ℓ can be included in the same cycle, and thus it must hold $\ell = i$. Hence, the element at position i must be marked. When the algorithm reaches position i, it skips the position and the cycle is finally "closed."

2. Trivial. Since $\Delta_i = 1$ if $M_i = 1$, the statement follows.

3. We close the cycle to which j belongs when reaching some marked element $A[j]$, so that j is the leader of that cycle. Position j was marked at some previous iteration k, hence the element u sitting at position k is moved to $A[j]$ and the element v formerly at $A[j]$ moves to position k. The cycle will be "closed" when the main loop reaches position j, but no other element may enter the cycle in the meanwhile. So what we want to proof is that k is part of the cycle, as no other value between j and k can be part of the cycle. And indeed it must be part of the cycle, as the element v that we moved to position k is part of the cycle.

4. By definition, $\Delta_i = 0$ when neither $A[i]$ is marked, nor any other element gets marked. So i is not the leader nor the second smallest element of the cycle, because otherwise at iteration i we would either mark the leader's position or would be at the leader's position.

5. Since 1 is a leader of its cycle, it must be marked when $i = 1$. Therefore the number of unmarked items U_1 in $A[1..1]$ is 0 and hence $B_2 = 1 - U_1 = 1$. As $B_1 = B_2 - \Delta_1$ and $\Delta_1 = 1$, we have $B_1 = 0$.

6. This follows from points 2.-4. of this proposition.

7. This follows directly from the identity $U_i + B_{i+1} = i$ and obvious bounds on U_i.

8. Recall that $U_i + B_{i+1} = i$. Hence $B_{i+1} = i - U_i$. If $U_i \geq 2$ then the statement is trivially true. The situation $U_i = 1$ cannot happen; otherwise, the algorithm would try to construct a derangement of size 1 and that's impossible. Incidentally, this proves that $B_{i+1} \neq i - 1$ for any i. Finally, we have only to show that happens when $U_i = 0$. This means that $A[1], \ldots, A[i]$ are marked and hence $M_j = 1$, for $1 \leq j \leq i$. I.e., $1, 2, \ldots, i$ are leaders of their respective cycles, and in particular $M_i = 1$, which contradicts the hypothesis of the proposition.

Now we can find a crude estimate of $\mathbb{E}[G]$ which is enough for all practical purposes. To begin with,

$$\begin{aligned}
\mathbb{E}[G] &= \sum_{i=2}^{n} \mathbb{E}\left[\frac{i-1}{i-1-B_{i+1}} \mid M_i = 0\right] \mathbb{P}[M_i = 0] \\
&\leq \sum_{i=2}^{n} \mathbb{E}\left[\left|\frac{i-1}{i-1-B_{i+1}}\right| \mid M_i = 0\right] \mathbb{P}[M_i = 0] \\
&\leq \sum_{i=2}^{n} \Big(\mathbb{E}\left[\left|\frac{i-1}{i-1-B_{i+1}}\right| \mid M_i = 0\right] \mathbb{P}[M_i = 0] \\
&+ \mathbb{E}\left[\left|\frac{i-1}{i-1-B_{i+1}}\right| \mid M_i \neq 0\right] \mathbb{P}[M_i \neq 0]\Big) \\
&= \sum_{i=2}^{n} \mathbb{E}\left[\left|\frac{i-1}{i-1-B_{i+1}}\right|\right].
\end{aligned}$$

Using the bounds given in Proposition 3.1(6-8) and conditioning in the number of cycles we have

$$\begin{aligned}
\mathbb{E}[G] &\leq \sum_{i=2}^{n} \mathbb{E}\left[\min\left\{i-1, \frac{i-1}{i-1-C_n}\right\}\right] \\
&\leq \sum_{k=1}^{\lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \cdot \left[\sum_{i=1}^{k+1}(i-1) + \sum_{i=k+2}^{n} \frac{i-1}{i-1-k}\right] \\
&\leq \sum_{k=1}^{\lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \cdot \left[\binom{k}{2} + n - k - 1 + kH_{n-1-k}\right] \\
&= \frac{1}{2}\,\mathbb{E}\big[C_n^{\underline{2}}\big] + n - 1 - \mathbb{E}[C_n] + \mathcal{O}(\mathbb{E}[C_n \ln(n - C_n)]) \\
&= n + \mathcal{O}(\mathbb{E}\big[C_n^{\underline{2}}\big]) + \mathcal{O}(\log n \cdot \mathbb{E}[C_n]),
\end{aligned}$$

where $H_n = \sum_{1 \leq j \leq n}(1/j) \sim \ln n + \mathcal{O}(1)$ denotes the n-th harmonic number.

The required results for the r.v. C_n can be obtained easily via a generating functions approach studying

$$C(z, v) = \sum_{n \geq 0} \mathbb{E}\big[v^{C_n}\big]\, D_n \frac{z^n}{n!}.$$

By using the natural decomposition of a derangement as a product of cycles of length ≥ 2, we get

$$C(z, v) = e^{\left(\log\left(\frac{1}{1-z}\right) - z\right) v} = e^{-vz} \frac{1}{(1-z)^v}.$$

Basic singularity analysis leads then to the following asymptotic expansion of the probability generating function $\mathbb{E}\big[v^{C_n}\big]$, which holds uniformly in a complex neighbourhood of $v = 1$, with an arbitrary $\epsilon > 0$:

$$\begin{aligned}
\mathbb{E}\big[v^{C_n}\big] &= \frac{n!}{D_n}[z^n]C(z, v) \\
&= \frac{e^{1-v}}{(v-1)!} e^{(v-1)\log n}\left(1 + \mathcal{O}(n^{-1+\epsilon})\right).
\end{aligned}$$

A direct application of Hwang's quasi power theorem [9] shows then that the normalized r.v. $\frac{C_n - \mathbb{E}[C_n]}{\sqrt{\mathbb{V}[C_n]}}$ converges in distribution to a standard normal distributed r.v. together with the asymptotic expansions $\mathbb{E}[C_n] = \log n + \mathcal{O}(1)$ and $\mathbb{V}[C_n] = \log n + \mathcal{O}(1)$. It follows from these results that

$$\mathbb{E}[G] \leq n + \mathcal{O}(\log^2 n)$$

Since $\mathbb{E}[G] \geq n - \mathbb{E}[C_n]$, we further have that

$$\mathbb{E}[G] = n + \mathcal{O}(\log^2 n),$$

so that the following theorem holds.

THEOREM 3.1. *The expected total number of random numbers used by Algorithm 1 to generate a random derangement of size n is $2n + \mathcal{O}(\log^2 n)$.*

This compares favorably to the average complexity of random generation based upon a straightforward implementation of the recursive method.

3.2 The number of moves Even though the number of times a particular element is swapped by Algorithm 1 has not direct impact on its performance, its analysis raises interesting mathematical challenges, as in the corresponding analysis for Sattolo's algorithm and Fisher-Yates shuffle [12, 13, 18, 11].

Let us denote by $M_{n,p}$ the number of moves of element p when generating a random derangement of length n and by $M_n := M_{n,U_n}$ the number of moves made by a random element. Here U_n is uniformly distributed on $\{1, 2, \ldots, n\}$.

First we introduce the probability generating functions for the number of moves of specific elements:

$$\varphi_{n,p}(v) := \sum_{m\geq 0} \mathbb{P}[M_{n,p} = m]\, v^m, \qquad n \geq 2, \quad 1 \leq p \leq n.$$

We obtain then the following recurrences for the functions $\varphi_{n,p}(v)$, if we additionally define $\varphi_{n,p}(v) = 0$ if $n \leq 1$ or $p < 1$ or $p > n$:

$$\varphi_{n,n}(v) = \frac{(n-1)D_{n-2}}{D_n} v + \frac{D_{n-1}}{D_n} v \sum_{k=1}^{n-1} \varphi_{n-1,k}(v), n \geq 2, \tag{3.2}$$

$$\varphi_{n,p}(v) = \frac{D_{n-2}}{D_n}\Big(v + (p-1)\varphi_{n-2,p-1}(v) + (n-1-p)\varphi_{n-2,p}(v)\Big) + \frac{D_{n-1}}{D_n}\big(v + (n-2)\varphi_{n-1,p}(v)\big), \quad 1 \leq p < n. \tag{3.3}$$

These recurrences are obtained when distinguishing the cases, where element n is contained in a cycle of length 2 (which appears with probability $\frac{(n-1)D_{n-2}}{D_n}$) or in a cycle of length ≥ 3 (which appears with probability $\frac{(n-1)D_{n-1}}{D_n}$).

We will use the recurrences (3.2) and (3.3) to study the r.v. M_n and thus the number of moves of a random element and introduce the probability generating function

$$\psi_n(v) := \sum_{m\geq 0} \mathbb{P}[M_n = m]\, v^m = \frac{1}{n}\sum_{k=1}^{n} \varphi_{n,k}(v)$$

and the abbreviation $\tilde{\psi}_n(v) := n\psi_n(v)$.

Equation (3.2) can then be rewritten as

$$\varphi_{n,n}(v) = \frac{(n-1)D_{n-2}}{D_n} v + \frac{D_{n-1}}{D_n} v\tilde{\psi}_{n-1}(v), \tag{3.4}$$

whereas we obtain by summation for p from 1 up to n from (3.3):

$$\tilde{\psi}_n(v) - \varphi_{n,n}(v) = v + \frac{D_{n-2}(n-1)}{D_n}\tilde{\psi}_{n-2}(v) + \frac{D_{n-1}(n-2)}{D_n}\tilde{\psi}_{n-1}(v). \tag{3.5}$$

After combining (3.4) and (3.5) this gives the following linear recurrence for $\tilde{\psi}_n(v)$:

$$\tilde{\psi}_n(v) = \left(1 + \frac{(n-1)D_{n-2}}{D_n}\right) v + \frac{(n-2+v)D_{n-1}}{D_n}\tilde{\psi}_{n-1}(v) + \frac{(n-1)D_{n-2}}{D_n}\tilde{\psi}_{n-2}(v), \quad n \geq 2, \tag{3.6}$$

with initial values $\tilde{\psi}_0(v) = \tilde{\psi}_1(v) = 0$. To treat this recurrence we introduce the bivariate generating function

$$\psi(z, v) := \sum_{n\geq 2} D_n \tilde{\psi}_n(v)\frac{z^n}{n!} = \sum_{n\geq 2} D_n \psi_n(v)\frac{z^n}{(n-1)!}.$$

Recurrence (3.6) can then be translated into the following first order linear differential equation for $\psi(z, v)$:

$$(1-z)\frac{\partial}{\partial z}\psi(z, v) + (1-z-v)\psi(z, v) = \frac{vz(2-z)e^{-z}}{(1-z)^2}, \tag{3.7}$$

with initial condition $\psi(0, v) = 0$. This differential equation has the following exact solution:

$$\psi(z, v) = e^{-z}\left(1 + \frac{v}{(2-v)(1-z)^2} - \frac{2}{(2-v)(1-z)^v}\right). \tag{3.8}$$

Extracting coefficients leads then from (3.8) to the following exact solution for the probability generating function $\psi_n(v)$ (with $n \geq 2$):

$$\psi_n(v) = \frac{(n-1)!}{D_n}[z^n]\psi(z, v) = \frac{(-1)^n}{nD_n} + \frac{v}{2-v}\left(1 + \frac{1}{n} + \frac{D_{n-1}}{D_n}\right) - \frac{(n-1)!}{D_n}\frac{2}{2-v}\sum_{k=0}^{n}\frac{(-1)^k}{k!}\binom{n-k+v-1}{n-k}. \tag{3.9}$$

Additionally one easily obtains via singularity analysis the asymptotic expansion

$$\psi_n(v) = \frac{v}{2-v} + \mathcal{O}(n^{-1}) + \mathcal{O}(n^{v-2}), \tag{3.10}$$

which holds uniformly for $|v| \leq 2 - \epsilon$ and $\epsilon > 0$. Thus the sequence of probability generating functions $\psi_n(v)$ converges uniformly in a complex neighbourhood of $v = 1$ to $\frac{v}{2-v}$, which is the probability generating function of a geometric r.v. with parameter $\frac{1}{2}$. Therefore M_n converges in distribution to a geometric distribution with parameter $\frac{1}{2}$.

References

[1] P.R. de Montmort. *Essay d'analyse sur les jeux de hazard. Seconde Edition, Revue & augmentée de plusieurs Lettres.* Jacque Quillau, Paris, 1713.

[2] R. Durstenfeld. Algorithm 235: Random permutation. *Communications of the ACM*, 7(7):420, July 1964.

[3] L. Euler. Calcul de la probabilité dans le jeu de rencontre. *Memoires de l'Academie des Sciences de Berlin*, (7):255–270, 1753.

[4] L. Euler. Solutio quaestionis curiosae ex doctrina combinationum. *Memoires de l'Academie des Sciences de St. Petersbourg*, 3:57–64, 1811.

[5] R.A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research.* Oliver & Boyd, London, 3rd edition, 1948.

[6] Ph. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

[7] I. Goulden and D. Jackson. *Combinatorial Enumerations.* John Wiley & Sons, 1983.

[8] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison–Wesley, 2nd edition, 1994.

[9] Hwang H.-K. On convergence rates in the central limit theorems for combinatorial structures. *European J. Combin.*, 19:329–343, 1998.

[10] D.E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms (volume 2).* Addison-Wesley, 3rd edition, 1997.

[11] G. Louchard, H. Prodinger, and S. Wagner. Joint distributions for movements of elements in Sattolo's and Fisher-Yates' algorithm. Preprint. Available at `http://www.ulb.ac.be/di/mcs/louchard/`, 2007.

[12] H.M. Mahmoud. Mixed distributions in Sattolo's algorithm for cyclic permutations via randomization and derandomization. *J. Appl. Probability*, 40:790–796, 2003.

[13] H. Prodinger. On the analysis of an algorithm to generate a random cyclic permutation. *Ars Combinatoria*, 65:75–78, 2002.

[14] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, 1977.

[15] J. Riordan. *An Introduction to Combinatorial Analysis.* Wiley, New York, 1958.

[16] S. Sattolo. An algorithm to generate a random cyclic permutation. *Information Processing Letters*, 22(6):315–317, 1986.

[17] H. S. Wilf and A. Nijenhuis. *Combinatorial Algorithms.* Academic Press, 2nd edition, 1978.

[18] M. C. Wilson. Probability generating functions for Sattolo's algorithm. *J. Iranian Statistics Soc.*, 3(2):297–308, 2004.