SERIAL PERMUTATION METHOD

GLAUCIO G. DE M. MELO AND EMERSON A. DE O. LIMA

Abstract. This article presents the *Serial Permutation Method* (SPM). The SPM proposal is to obtain a permutation vector from its serial number. The algorithm that does the SPM's inverted method is also showed, getting the serial number from the permutation vector.

Introduction

The permutation algorithms are classified in two groups: the one that creates a set of permutations from the identity permutation and the other that produces a set of permutations by means of simple changes between the vector's elements, creating a new permutation from the previous one. The algorithm *Next Permutation for N letters* [1] belongs to the second group, creating a complete set of permutations through successive algorithm invocation, getting the next permutation until reaches the last vector of the list. The algorithm *Next Permutation* is powerful creating the next permutation vector using only local information. This guarantees that the next vector will be different from anyone else that was defined before.

There is a specific problem on this creating process:

> *Is it possible to get directly a permutation vector located on a specific position on the list of vectors, excluding the alternative of the Next Permutation algorithm's successive execution until gets the desired vector?*

The *Serial Permutation Method* (SPM) answers this question, being able to process the same list of permutation that the *Next Permutation* algorithm does, with a difference: the SPM only needs the serial number and the permutation vector's size to process the desired output. It is also possible to invert this process from the SPM, getting the serial number through the permutation vector.

SPM Construction

The SPM was developed from the observation of the *Next Permutation's* data outputs. This algorithm has an auxiliary variable called offset vector, which is defined below [1].

Consider p the permutation vector with n elements and let d be the offset vector with $n-1$ elements, the vector d is defined by:

$$d_i = |j : j \leq i,\ p_j > p_{i+1}|\ i = 1, 2, ..., n \tag{1}$$

That is explained:

Key words and phrases. Combinatorial Algorithms, Complexity, Combinatorial Optimization, Permutation.

How many numbers are there biggest than p_{i+1} between the beginning of the permutation vector and the i index of the d vector?

Being $n!$ the complete list of the created vectors, we have in Table 1 an example of output from the p permutation vector related to its serial, for $n = 5$ and the $serial = 32$.

TABLE 1. Input and output processing that are common in the *Next Permutation* algorithm.

Input	Output
$n = 5$	$p = (3, 5, 1, 2, 4)$
$Serial = 32$	$d = (0, 2, 2, 1)$

The SPM is subdivided in two steps, and the data input of the second step corresponds to the data output of the first step:

(1) Given a serial number and the size of the permutation vector, determine the offset vector;
(2) Given the offset vector, determine the permutation vector.

Algorithm to Attain the Offset Vector. To solve the first SPM step it is necessary an algorithm to get the offset vector, which is called *Serial Offset Algorithm* (SOA). The algorithm does a calculus that reflects the pattern of the offset vectors' creation throughout the whole list of permutation, using only local information. In Table 2, we can visualize the pattern of the offset vector in the whole list of permutation, where the detached column will serve as a guide to the determination that the SOA decodes. Let d be the offset vector with n elements. Each vector's d column showed in Table 2 has the following properties:

- Let k be the value of the current column ($k = 1, 2, ..., n$), the elements of the d_k column ($0 \leq d_k \leq k$) repeat $k!$ times until the element of the column gets the k value. This can be visualized on the first part of the detached column in Table 2, illustrated in blue, which the number of detached elements in blue or red is defined by $(k + 1)!$
- When $d_k = k$ in $k!$ times, the following elements from the list are put on the inverted form, like showed in red in Table 2. The direct and inverted list intercalate themselves until complete all the positions of the column.

Exists throughout each offset vector column an intercalation between the two kinds of lists (direct and inverted), we can consider this fact as an element of parity in the list, considering as an even the list in its direct form and as odd the list in its inverted form. To determinate the relation between the serial number and the parity of the list, follows:

$$f = \left\lfloor \frac{s-1}{(k+1)!} \right\rfloor \ mod\ 2 \tag{2}$$

Where:

- f: Determines if the list is direct ($f = 0$) or inverted ($f = 1$);
- s: Serial number;
- k: Index of the offset vector;
- $\lfloor x \rfloor$: Floor function. It returns the biggest integer value smaller than x;
- *mod*: An operation that returns a division's rest.

TABLE 2. Full list of permutations, with $n = 4$

Serial	p	d
1	(1,2,3,4)	(0,0,0)
2	(2,1,3,4)	(1,0,0)
3	(3,1,2,4)	(1,1,0)
4	(1,3,2,4)	(0,1,0)
5	(2,3,1,4)	(0,2,0)
6	(3,2,1,4)	(1,2,0)
7	(4,2,1,3)	(1,2,1)
8	(2,4,1,3)	(0,2,1)
9	(1,4,2,3)	(0,1,1)
10	(4,1,2,3)	(1,1,1)
11	(2,1,4,3)	(1,0,1)
12	(1,2,4,3)	(0,0,1)
13	(1,3,4,2)	(0,0,2)
14	(3,1,4,2)	(1,0,2)
15	(4,1,3,2)	(1,1,2)
16	(1,4,3,2)	(0,1,2)
17	(3,4,1,2)	(0,2,2)
18	(4,3,1,2)	(1,2,2)
19	(4,3,2,1)	(1,2,3)
20	(3,4,2,1)	(0,2,3)
21	(2,4,3,1)	(0,1,3)
22	(4,2,3,1)	(1,1,3)
23	(3,2,4,1)	(1,0,3)
24	(2,3,4,1)	(0,0,3)

If the list had been direct, we attribute to the elements of the offset vector:

$$d_k = \left\lfloor \frac{[(s-1) \ mod \ (k+1)!]}{k!} \right\rfloor \tag{3}$$

If the list had been inverted, we attribute the complement which would be the direct list:

$$d_k = k - \left\lfloor \frac{[(s-1) \ mod \ (k+1)!]}{k!} \right\rfloor \tag{4}$$

Each attribution is made through a loop that go through the offset vector.

Serial Offset Algorithm (SOA)
Routine Specifications:

- n: Size of the offset vector;
- i: Index of the offset vector;
- d: Offset vector, alternating its indexes from $0..n-1$;
- s: Permutation's serial.

Routine:
  For $i \leftarrow 1$ to n do
    If $\lfloor (s-1) \ / \ (i+1)! \rfloor \ mod \ 2 = 1$
      $d_{i-1} \leftarrow i - \lfloor ((s-1) \ mod \ (i+1)!) \ / \ i! \rfloor$
    Else
      $d_{i-1} \leftarrow \lfloor ((s-1) \ mod \ (i+1)!) \ / \ i! \rfloor$
  End For
  return d.

Algorithm to attain the permutation vector from the offset vector. After the SOA is computed, the SPM is concluded with the *Permutation Algorithm by Offset* (PAO). The PAO does the SPM's second step, returning the desired output. A relevant topic for the PAO construction is to find the decoding process of the permutation vector, with only the offset vector being the input. We know that the offset vector maps the elements of the permutation vector. In the decoding, we have:

Let p be the permutation vector and d the offset vector:

$$p = (p_1, p_2, ..., p_n) \quad d = (d_1, d_2, ..., d_{n-1}) \tag{5}$$

And d already has its values determining by the SOA. This is how the decoding process is made: we know that d_1 has its values discretely included between 0 and 1. Obviously, 0 and 1 are the unique possible elements for d_1. From this information, we can conclude:

$$d_1 = \begin{cases} 0, & \text{if } p_1 < p_2 \\ 1, & \text{if } p_2 < p_1 \end{cases}$$

We do not consider the possibility of equality between the elements of the permutation vector, because we know that there is no repeated elements on the vector, and if we organize it, the difference between them will be only one unit. Like d_1, the element d_2 has its values included between 0 and 2. In this and in other cases, we analyze the current inequality from the inequality that was created previously. For d_2, we have:

$$d_1 = \begin{cases} 0, \quad d_2 = \begin{cases} 0, & \text{if } p_1 < p_2 < p_3 \\ 1, & \text{if } p_1 < p_3 < p_2 \\ 2, & \text{if } p_3 < p_1 < p_2 \end{cases} \\ \\ 1, \quad d_2 = \begin{cases} 0, & \text{if } p_2 < p_1 < p_3 \\ 1, & \text{if } p_2 < p_3 < p_1 \\ 2, & \text{if } p_3 < p_2 < p_1 \end{cases} \end{cases}$$

We can conclude that as the value of d_2 increases, p_3 "slides" on the left through the inequality. In general, we have:

$$d_k \in \{0, 1, ..., k\} \quad \{p_i < p_j < ... < p_k < ... < p_t\} \quad i, j, t \neq k \tag{6}$$

With p_k in the inequality with d_k positions, counted from right to left, because the order of the elements is ascendent. Finished the offset vector's raster, we will

get a set of inequalities that informs the order of the permutation elements. We have, for example:

Given a offset vector $d = (0, 2, 2)$, the inequality for a p permutation vector is:

$$p_3 < p_4 < p_1 < p_2 \tag{7}$$

Being the last inequality at 7 the final disposition between the elements of the vector. After the mapping of the permutation's elements was done, we can say that each element of the vector ordered in 7 corresponds to the elements of the identity permutation. This fact classifies the SPM in the first group described in the introduction of this article. On the 7 inequality, we have:

$$(p_3 = 1) < (p_4 = 2) < (p_1 = 3) < (p_2 = 4) \tag{8}$$

The next step to get the final output is to arrange each one of the elements ordered on its own positions. As the p vector is ordered like $(p_1, p_2, ..., p_n)$, we have:

$$p = (p_1, p_2, p_3, p_4) \quad \Rightarrow \quad p = (3, 4, 1, 2) \tag{9}$$

So we can get the SPM's final output. In implementation terms, to compute the input built from the permutation vector (from right to left) the corresponding indexes of the vector were used in relation to its complement. We will notice at the implementation that in the insertion moment of the current element, if it replaces another, the elements of inequality located on the left will "slide" to the left side, allocating space for the current element. The "slides to the left" operation is implemented on the subroutine *push*.

Permutation Algorithm by Offset (PAO)

Routine Specifications:

- n: Size of the permutation vector;
- i, j: Indexes of the algorithm's vectors;
- p: Permutation vector, alternating its elements on $0..n-1$;
- r: Vector which will keep the element's position before being ordered.

Routine:

$r_{n-1} \leftarrow 1$
For $i \leftarrow 1$ to $n-1$ do
  If $r_{n-1-d_{i-1}} = 0$
    $r_{n-1-d_{i-1}} \leftarrow i + 1$
  Else
    $r_{n-1-d_{i-1}} \leftarrow push$
End For
For $i \leftarrow 0$ to $n-1$ do
  $p_{r_i - 1} \leftarrow i + 1$
End For
return p.

Subroutine Push

For $j \leftarrow (n-1) - i$ to $j < (n-1) - d_{i-1}$ do
  $p_j \leftarrow p_{j+1}$
End For
return $i + 1$.

SPM Inverted Process

We can describe now the inverted process of the SPM's. We have the permutation vector as the input, and the desired output is the correspondent serial number. The ingenuous process to get the serial number for the permutation vector is the raster of the permutation list, comparing the vectors one by one, until gets the equivalent vector computed on the input, being the returned value the loop's index that does this raster. However, we can find the serial value inverting the SPM's steps:

(1) Given the permutation vector, find the offset vector;
(2) Given the offset vector, find the serial number.

Like the SPM, the second process depends on the first, with the first step data output corresponding to the second step data input.

Algorithm to attain the offset vector from the permutation vector. We will call this algorithm as the *Offset Algorithm by Permutation* (OAP). It uses the definition of the offset vector (see equation 1). Two nested loops add the value of each element of the offset vector.

Offset Algorithm by Permutation (OAP)
Routine Specifications:

- n: Size of the permutation vector;
- i, j: Indexes of algorithms's vectors;
- p: Permutation vector alternating its elements on $0..n-1$;
- d: Offset vector alternating its elements on $0..n-2$.

Routine:

```
For i ← 0 to n − 2 do
    For j ← 0 to i do
        If p_j > p_{i+1}
            d_i ← d_i + 1
        End If
    End For
End For
return d.
```

Algorithm to attain the serial number from the offset vector. Like the OAP, the *Serial Algorithm by Offset* (SAO) also uses definitions already showed at this article. We know that the value attributed for the elements of the offset vector, defined on 3 and 4 can be ordered depending on the serial. Given the variables:

- $d = (d_1, d_2, ..., d_k, ..., d_n)$: Offset vector;
- d_k: Offset vector's element;
- s: Permutation's serial number;
- k: Index of the offset vector;
- q: Quotient of the division between $s-1$ and $(k+1)!$;
- $\lfloor x \rfloor$: It returns the biggest integer value smaller than x.

For direct list, we have:

$$s - 1 = d_k.k! + \lfloor q.(k+1)! \rfloor \tag{10}$$

For inverted lists, we have:

$$s - 1 = (k - d_k).k! + \lfloor q.(k+1)! \rfloor \tag{11}$$

An important question for the algorithm implementation is how to find the variable's q value. To solve this question we have to consider that d_n belongs to a column that has only one list on a direct disposition (see Table 2, last vector d column). With this information, we conclude that the quotient for this column corresponds to a value between zero (closed) and one (open), making:

$$\lfloor q.(k+1)! \rfloor \tag{12}$$

Corresponds to zero. With this, 10 and 11 is equivalent to:

$$s - 1 = d_k.k! \tag{13}$$

For direct lists, and

$$s - 1 = (k - d_k).k! \tag{14}$$

For inverted lists.

The strategy of implementation to get the serial is done by incremental mode, working the current information being based on past information, doing a raster on the offset vector from right to left. This is particularly useful on this case, considering the next value that will converge to the final serial that belongs to a group related with the previous serial elements. Each quotient q during the raster will corresponds to zero, because we are considering the division in relation to the gotten serial. Another relevant question is how to classify if the previous element of the offset vector it is contained in a direct or inverted list. Looking at Table 2, we can make easily an equivalence grade, showed in Table 3.

TABLE 3. Next element status list of the offset vector

Previous Value - Parity			Next Value - Parity
Element	List		List
Even	Direct	$\Longrightarrow$	Direct
Even	Inverted	$\Longrightarrow$	Inverted
Odd	Direct	$\Longrightarrow$	Inverted
Odd	Inverted	$\Longrightarrow$	Direct

Exemplifying the attainment serial process from the offset vector, we have:

Let $d = (1, 0, 3)$ be the offset vector and s the permutation serial. As the raster is done from right to left, we work first with the value 3. As was already said, we have as an initial input a direct list which 3 is contained. For direct lists, we use an equation showed on 13:

$$s \leftarrow 3.3! \Rightarrow 18 \tag{15}$$

After that, we observer the next element of the offset vector. As the previous element is in a direct list and it is odd, the element 0 will be in a inverted list. Using 14, we have:

$$s \leftarrow s + (2 - 0).2! \Rightarrow 18 + 4 \Rightarrow 22 \tag{16}$$

Finally, observing the last element of the offset vector, we have the previous element which is in a inverted list and it is even. Looking at Table 3, we evidence that the element 1 is in a inverted list. Thus, we have:

$$s \leftarrow s + (1 - 1).1! \Rightarrow 22 + 0 \Rightarrow 22. \tag{17}$$

After we have calculated the serial in a incremental mode, we add 1 to the serial, because the equations 13 and 14 depend on $s-1$. Thus, the serial value characterizes it between $(1, n!)$. Where n is the size of the permutation value given as an input:

$$s \leftarrow s+1 \Rightarrow 22+1 \Rightarrow 23. \tag{18}$$

The serial 23 is the corresponding serial to the offset vector $(1, 0, 3)$ which corresponds to the permutation vector $(3, 2, 4, 1)$. On the algorithm's implementation showed here, the increment of one unit to the serial was made in the beginning, before get into the loop. A boolean variable was specified to determine if the list which belongs the elements is direct or inverted. The conditions that determine if the list is direct or inverted is optimized from four to two conditions. The algorithm returns a non-negative integer value, corresponding to the serial number required.

Serial Algorithm by Offset (SAO)
Routine Specifications:

- s: Serial value;
- n: Size of the offset vector;
- i: Index of the offset vector;
- d: Offset vector, alternating its elements on $0..n-1$;
- *direct*: Boolean variable. It determines if the list is direct or not.

Routine:

$s \leftarrow [\ d_{n-1}.(n-1)!\ ] + 1$
$direct \leftarrow true$
For $i \leftarrow n-1$ to 1 do
  If $(d_i = even\ and\ direct = true)\ or\ (d_i = odd\ and\ direct = false)$
    $direct \leftarrow true$
    $s \leftarrow s + d_{i-1}.(i-1)!$
  End If
  Else
  If $(d_i = odd\ and\ direct = true)\ or\ (d_i = even\ and\ direct = false)$
    $direct \leftarrow false$
    $s \leftarrow s + (i - d_{i-1}).(i-1)!$
  End If
End For
return s.

Demonstration

(This section is not ready)

References

[1] WILF, Herbert S., NIJENHUIS, A. Combinatorial Algorithms for computers and calculators. Academic Press, INC, 1978.

(Melo) Departamento de Estatística e Informática - UNICAP
E-mail address, Melo: glaucio@dei.unicap.br

(Oliveira-Lima) Departamento de Estatística e Informática - UNICAP
E-mail address, Oliveira-Lima: eal@dei.unicap.br