Programming the Permutation Method by proving two Components By Using Matlab

Lugen M. Zake Sheet

Received: 31/8/2020 Accepted: 20/9/2020 Published: 2020

Programming the Permutation Method by proving two Components By Using Matlab

Lugen M. Zake Sheet
University of Mosul/ Basic Education Faculty
lujaenalsufar@yahoo.com

ABSTRCT:

We will provide programming performance for the proposed ideal algorithm for generating permutations by installing two components, the first and the last component, by configuring all permutations using the periodic use of the remaining elements. In this proposed method, the effort expended whether using the paper or using the calculator to! (n-2)! compared to the effort exerted by the methods used previously and in this way we used two methods Processes for finding permutations (random and regular) in finding permutations. With this paper we presented the method on a computer only by applying the proposed algorithm in the language of MATLAB.

INTRODUCTION:

Combinatorial generation, or generation Permutation, account and seek are among basic problems in computer science [2,5]. In many application areas various classes of combinatorial things are complex as test or problem instances. Various generation algorithms have been calculated for such combinatorial things like combinations, compositions, permutations, derangements, variations, partitions, graphs, trees etc. In different representations. Not suitable and more flexible algorithms for variations have to be imaginary. In 1971 Ord-Smith [9] compared several permutation algorithms and concluded that Boothroyd's implementation of M.B. Well's

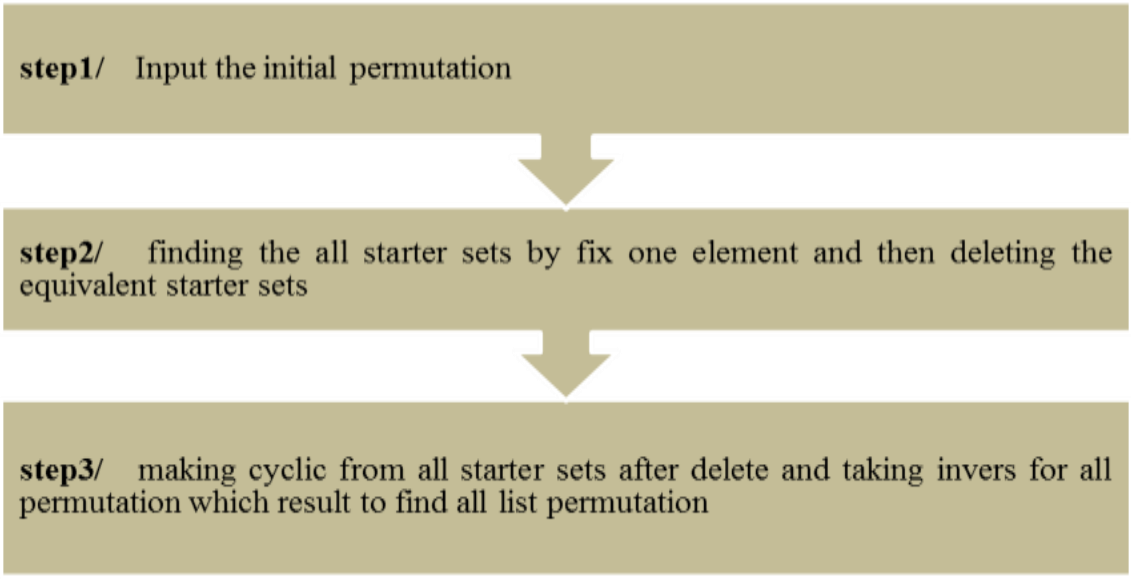
algorithm was the fastest published permutation algorithm. Generation of Well's sequence requires only a single exchange of two marks to generate a permutation from its predecessor in the sequence. More than half of the exchanges are with adjacent marks. In 1973, Ehrlich [3] coded an algorithm, PERMU, to generate the Johnson-Trotter permutation sequence. He claimed his implementation was twice as fast as Boothroyd's algorithm. Part of the speed of PERMU was attributed to its loop-free implementation [3, 4].

To permute the marks 1, 2, . . . , n, the Johnson- Trotter algorithm moves mark n one position at a time to the left until it reaches the left end at which time a new permutation of the marks 1, 2 , . . . , n - 1 is obtained. Mark n then moves to the right. Each time a boundary is reached, a new permutation of the lesser marks is obtained and mark n switches direction. The same algorithm is used to generate permutations of the lesser marks. Three implementation difficulties are associated with this algorithm: (1) the direction of movement of each mark must be recorded; (2) a decision on which mark to move must be made at each invocation of the algorithm؛ and (3) the position of the lesser marks within the total set of marks varies between permutations of the lesser marks. The Johnson-Trotter sequence has the advantage that a permutation differs from its predecessor by a single exchange of two adjacent marks. Four new algorithms for generating permutation sequences are presented in this article [6]. These algorithms are called a, b, c, and d in the following discussion. Each generates a unique permutation sequence. Algorithm a and c are simpler than the Johnson-Trotter sequence in that they avoid implementation difficulties 1, 2, and 3 listed previously. Algorithms a and c occasionally require exchanging more than one pair of marks. This occurs once in every n(n - 1) permutations.

Algorithms b and d require exchanging only one pair of marks at each entry, but both share implementation difficulties 1and 2 with the Johnson-Trotter algorithm. Algorithms b and d avoid difficulty 3. Combinatorial properties of variations and variations with repetitions are explained in [2,5]. position and un Ranking schemes for variations without repetitions are given in [1]. In the present paper the presentation of programming algorithm from a representation of permutations in fixing two elements in permutation.

ALGORITHM STARTER SETS:

We had presented some papers about the studying finding permutations by using starter sets algorithm which is fixing one element, it appears in Lugen (2012)[7]. It was easy compressing with other previous algorithms which was presented in the last. The starter algorithm presented in the following schema.



Schema A (Algorithm 1 by stabilization one element only)

This algorithm had good results on side the time and the speed to finding all permutations

THE PROPOSED ALGORITHM

We have developed the algorithm Permutation by starter sets by fixing one element which presented in last studies [7]. The presented technique by fixing two elements has reduced the time although it is had more steps for finding all permutations. We have presented this algorithm in the following steps:-

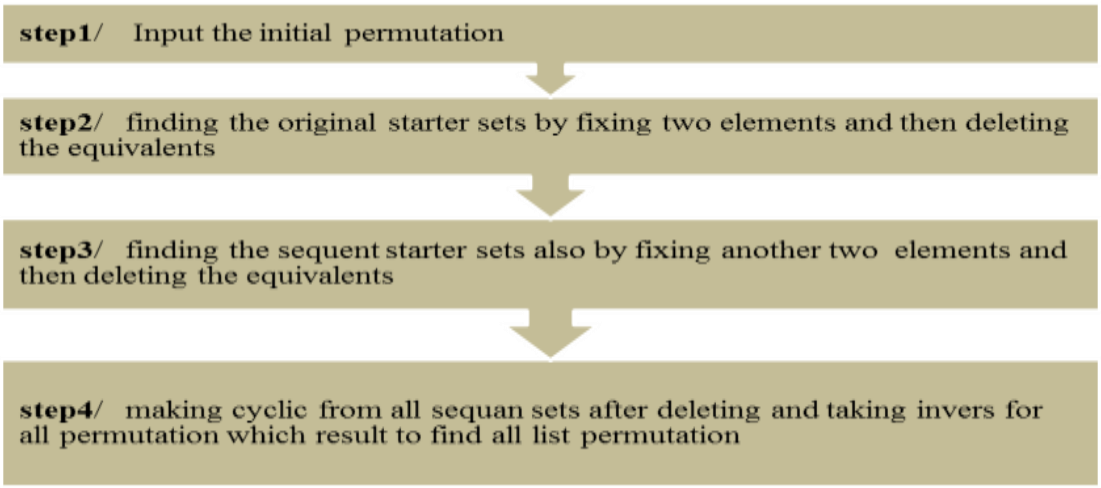
Step1: Generating original starter sets by fixing two elements without equivalence

Step 2: Generating sequent starter sets by fixing two elements also without equivalence

Step 3: Cycle for each sequent starter sets which resulted from step2 to get n!/2 permutation

Step 4: Reverse all n!/2 permutations to get inverse another n!/2 permutations

The schema is in the following:-



Schema B (Algorithm 2 by stabilization two-element)

PROGRAMMING THE PROPOSED ALGORITHM BY MATLAB

In this section, we present the suggested method for switching after installing two components by computer account using a Matlab. In the following code we perform the first and second steps in Matlab format. After inserting original permutation, we will install the first and second components. The remainder of the key is rotated by repetition and then we take the inverse as in the following code:-

```
Input ('n=');
 d=basicselect (1:n,[1 n]);
 a=[1*ones(size(d,1),1) d n*ones(size(d,1),1)];
```

After we find permutations from the previous code, we repeat the code with the same previous step, but by choosing to install two different elements the first itself and the last is different from the element in the previous step and also we find the permutations and the equivalent represented in a matrix and then delete the corresponding permutations using the following code: -

```
if size(a,1)==1
d=basicsellect(a(1,:),[1 i]);
 b=[[1*ones(size(d,1),1) d i*ones(size(d,1),1)]];
else
```

To generate all permutations from permutations from the previous step using Cyclic for all permutations with the following code, we deduce all permutations (n! / 2) from the required permutations.

```
b=[];
for i=n-1:-1:2
 d=basicsellect(a(n-i,:),[1 i]);
 b=[b;[1*ones(size(d,1),1) d i*ones(size(d,1),1)]];
end
```

Finally, the code provided by this step

```
makecycle.m
 c=makecycle([a;b]);
 d=[c;fliplr(c)];
```

works by taking the inverse of all permutations resulting from the previous code, whose number was divided by 2, so the resulting permutations are the number of permutations required.

With these steps and codes, the ideal suggested method is programmed.

RESULTS AND DISCUSSIONS:

In the codes provided is a complete programming for generating permutations in a way that uses two formats to find permutations (see APPENDIX). In the first and second step, random coordination was used, while the last step used regular periodic coordination. Thus, the programming of the steps for the proposed method was presented in a simplified manner and with complete accuracy to achieve the program for the proposed method.

CONCLUSION:

We have computed the proposed algorithm into the computer with MATLAB to insert n! Permutations based on fixing two components. The results showed that this proposed algorithm has a high accuracy in finding permutations and also a little calculation time. This study will assist in the use of this algorithm in other fields such as finding the determinant and

solving the small and large differential equation. Therefore, we can use this study in other fields in applied mathematics.

REFERENCES:

[1] P.G. Dobrev, Y.O. Minov, V.T. Trifonov, Encoding and decoding of variations without repetitions, Cybern. Syst. Anal. 32,741–744, http://dx.doi.org/10.1007/BF02367777. 1996.

[2] D.L. Kreher, D.R. Stinson, Combinatorial Algorithms: Generation, Enumeration, and Search, ACM, NY, 1999.

[3] Ehrlich, G. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J ACM, 20*, 3 (July 1973), 500-513.

[4] Ehrlich, G. Algorithm 466, Four combinatorial algorithms *Comm. ACM 16*, 11 (Nov. 1973), 690-691.

[5] F. Ruskey, Combinatorial generation, Working version (1j-CSC 425/ 520), http://www.1stworks.com/ref/ruskeycombgen. pdf, 2003.

[6] Lehmer, D.H. Teaching combinatorial tricks to a computer In *Combinatorial Analysis: Proceedings of Symposia in Applied Mathematics, Vol.* X, R. Bellman and M. Hall Jr,, Eds., Amer. Math. Soc., Providence, R. I., 1960

[7] L.M. Zake, New Algorithm for Determinant of Matrices Via Combinatorial Approach, Thesis PhD, Universiti Utara Malaysia, 2012.

[8] L. M. Zake, Zurn Omer, H. Ibrahim, The Perfect Algorithm for Generating Permutation, International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463 Vol. 2 Issue 11, November-2013, pp: (109-114),

[9] Ord-Smith, P.J. Generation of permutation sequences: Part 2. *Computer J. 14*, 2 (May 1971), 136-139.

APPENDIX:-

Step0: start

Step1: input the string with minimum Nummbers is :5

[1,2,3,4,5]

step2: fix two numbers first (1&5)

step3: loop

Position of [2 & 3 & 4] reverse and get the equivalents and save the output

step4: go step2 and fixing another two element

step5: Make cyclic for all permutation generating from step 2

step6: Take inverse for all permutation produced from step 3 to find all permutation from [1 2 3 4 5]

step7:end

المستخلص:

سنقدم أداء برمجة للخوارزمية المثالية المقترحة لتوليد التباديل بتثبيت عنصريين الاول والعنصر الاخير بتكوين جميع التباديل بالاستخدام الدوري للعناصر الباقية وبهذه الطريقة المقترحة يقلص الجهد المبذول سواء باستخدام الورقة او استخدام الحاسبة الى !(n-2) قياسا بالجهد المبذول بالطرق المستخدمة سابقا وبهذه الطريقة استخدمنا اسلوبين من العمليات لإيجاد التباديل المتداولة (العشوائية و المنتظمة) في ايجاد التباديل و بهذه الورقة قدمنا الطريقة على الحاسوب فقط بتطبيق الخوارزمية المقترحة بلغة MATLAB.