ASM Sci. J., Special Issue 6, 2019 for SKSM26, 157-161

Implementation of Circular Operation for Generating Starter Sets under Iterative Procedure

Sharmila Karim* and Haslinda Ibrahim

Pusat Pengajian Sains Kuantitatif, Universiti Utara Malaysia, 06010 Sintok, Kedah

Factorization of linear array of permutations is hard. The process of listing the permutations becomes simple via distinct starter sets. However, a problem arises when the equivalence starter sets will generate the similar permutation and needed to be discarded. Then a new iterative strategy is proposed to generate starter sets without generating equivalence starter sets by circular based. In order to list all permutations after the starter sets are obtained, the circular and reverse of circular operation are used on each starter set. Computational advantages are presented comparing the results obtained by the new algorithms with two other methods. The result indicates a new algorithm is better than others two in term of time execution.

Keywords: starter sets; circular; permutation; iterative procedure

I. INTRODUCTION

Permutation is an interesting topic in combinatorial problem and has a long history. In literature, there are a lot of work has been done in finding method to generate permutation. The recent methods are using factorial numbers (Borisenko *et al.*, 2008), Lexicographic order with a fixed number of inversion (Kuo, 2009) and starter sets (Ibrahim *et al.*, 2010).

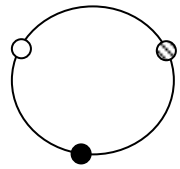
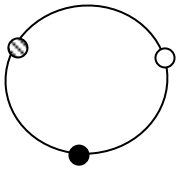
Permutation generation based on starter sets was proposed by Ibrahim *et. al*, (2010) by employing circular and reversing operation. The crucial task in their method was eliminating the equivalence starter sets after the distinct starter sets were obtained. Although this technique was simple and easy to use, unfortunately eliminating the equivalence starter sets was a quite tedious process when the number of elements increased. Thus Karim *et al.* (2010) proposed a circular strategy to generate starter set without generating equivalence starter sets. Since generating starter sets method is not unique, Karim *et al.* (2011) introduced new strategy based on exchanging two elements to generate starter sets also without discarding the equivalence starter sets. The advantages of distinct starter sets are redundancy of listing permutation can be avoided and can be applied for generating n order Latin squares. To see the listing of permutation by exploiting the $\frac{(n-1)!}{2}$ distinct starter sets, refer Karim *et al.* (2010). In spite of listing permutation, starter sets can be applied to find the distinct circuit in complete graph (Karim *et al.*, 2017). However, in this paper we highlight the implementation of circular operation for generating starter sets under iterative procedure and show the numerical result in term of time computation.

II. ILLUSTRATION OF STARTER SETS AND ITS EQUIVALENCE

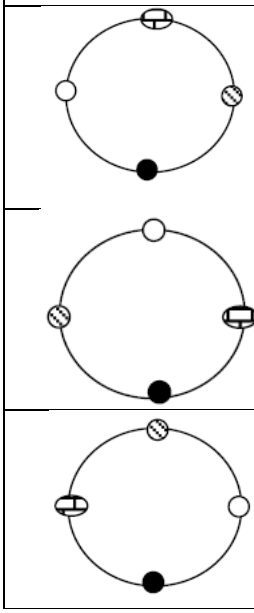
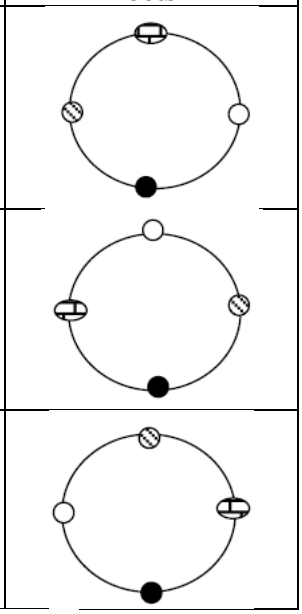
Let consider bead arrangement. There are $\frac{(n-1)!}{2}$ the number of ways that the other $n-1$ bead can be arranged in circular as starter sets. The second half $\frac{(n-1)!}{2}$ produced a similar bead arrangement in circular as equivalence starter sets. See the following figure for n = 3 and 4.

*Corresponding author's e-mail: mila@uum.edu.my

$n = 3, 2!/2 = 1$

Starter Sets	Equivalence Starter Sets
	

$n = 4, 3!/2 = 3$

Starter Sets	Equivalence Starter Sets
	

● = 1, ○ = 2, ⊘ = 3, ⊖ = 4.

Figure 1: Starter sets and its equivalence in beads arrangement

From graphic illustration in Figure 1, the starter sets and its equivalence are similar which the latter need to be discarded in order to avoid redundancy permutation generation.

III. STARTER SETS GENERATION UNDER CIRCULAR STRATEGY

Let S = (1,2,3,4,5)be the set of five elements. Step 1: Set (1, 2, 3, 4, 5) as initial permutation and without loss of generality, the first element is fixed.

Step 2: Identify the last three elements of initial permutation from step 1. By employing CP to the last three elements on initial permutation from step 1 will produce other three distinct starter sets as shown below:

1, 2, 3, 4, 5

1, 2, 4, 5, 3

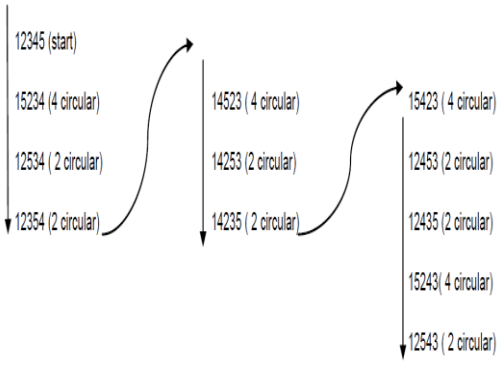
1, 2, 5, 3, 4

Step 3: Identify the last four elements of each starter sets in step 1. By employing CP to last four elements on each starter sets in step 2, the 12 distinct starter sets are obtained as shown below:

1, 2, 4, 5, 3	**1, 2, 5, 3, 4**	**1, 2, 3, 4, 5**
1, 4, 5, 3, 2	1, 5, 3, 4, 2	1, 3, 4, 5, 2
1, 5, 3, 2, 4	1, 3, 4, 2, 5	1, 4, 5, 2, 3
1, 3, 2, 4, 5	1, 4, 2, 5, 3	1, 5, 2, 3, 4

The process determining the starter sets is done under recursively. However, we had to modify our old recursive sequential algorithm in order to turn it to iterative sequential algorithm by combining some function from Langdon algorithm (1967). Its successful and our iterative algorithm did not generate the equivalence starter sets.

Our result in iterative way for n = 5 is as follows:



Our algorithm is as follows:
PERMUTIT (int n)
k=n;
WHILE k>2; DO

```
temp = num[1];
for (i=1, i<k, i++) DO
num[i]=num[i+1]
ENDFOR
num[k]=temp;
if k==2 || num[k] !=k,
BREAK
k--
ENDIF
ENDWHILE
```

Algorithm 1*: Iterative permutation generation pseudo code*

The input of this algorithm is a integer number of *n*.

The following table is a listing down all n! permutation.

CP	RoCP
12543	34521
25431	13452
54312	21345
43125	52134
31254	45213
15243	34251
52431	13425
24315	51342

CP	RoCP
14253	35241
42531	13524
25314	41352
53142	24135
31425	52413
14523	32541
45231	13254
52314	41325
23145	54132
31452	25413
12354	45231
23541	14532
35412	21453
54123	32145
41235	53214
12534	43521
25341	14352
53412	21435
34125	52143
41253	35214
15234	43251
52341	14325
23415	51432
34152	25143
41523	32514
12345	54321
23451	15432
34512	21543
45123	32154
51234	43215
43152	25134
31524	42513
12435	53421
24351	15342
43512	21534
35124	42153
51243	34215
15243	34251
52431	13425
24315	51342
43152	25134
31524	42513
12543	34521
25431	13452
54312	21345
43125	52134
31254	45213
14235	53241
42351	15324
23514	41532
35142	24153
51423	32415

From Table 1, the purpose of ROCP is a reversing the output from CP in order to generate another half of *n*!.

Remark 2: The **bold** mark of the permutation represent 12 starter sets for case $n = 5$.

IV. NUMERICAL RESULT

In this section, this new algorithm for listing permutation is compared to Langdon (1967) algorithm and Thongchiew (2007) algorithm. Langdon (1967) and Thongchiew (2007) were selected to compare because their algorithm both generated permutations using circular strategy under iterative procedure. Moreover, Langdon algorithm may run very fast on computer with hardware rotation capability (Sedgewick, 1977). The comparison over time computation among new algorithms, Langdon (1967) and Thongchiew (2007) are given in Table 1. The results are given in seconds without printing statement. The sequential algorithm of the developed and existing methods is written in C language and run on HP Computer with Intel Xeon CPU E5504, 2.0GHz processor and 4.00 GB Random Access Memory (RAM).

Table 2. The comparison over time computation among a new algorithm, Langdon (1967) and Thongchiew (2007) under iterative procedure (in seconds)

n	New algorithm	Langdon (1967)	Thongchiew (2007)
8	0.000	0.000	0.000
9	0.015	0.015	0.063
10	0.109	0.171	0.687
11	0.983	2.012	7.488
12	12.839	26.520	90.106
13	173.270	365.213	1672.510
14	2590.946	5423.443	22448.205
15	41885.652	85173.825	246139.962

From Table 2, we observed that our algorithm running lesser than Langdon (1967) and Thongchiew (2007). A time of our algorithms is about a half from Langdon (1967). From our point of view, we employed RoCP operation on CP result where we only reverse copied. In other word, $\frac{n!}{2}$ permutation resulted from CP operation over $\frac{(n-1)!}{2}$ starter sets. Then the second $\frac{n!}{2}$ is produced by reverse copied using RoCP operation over the first $\frac{n!}{2}$. On the other hand, Langdon (1967) and Thongchiew (2007) algorithm required more steps in listing all permutations. This factor might affect the computational time. In term of order of complexity, permutation generation algorithm fall under non polynomial time where equal O(n!).

Remark: The check our algorithm, Langdon (1967) and Thongchiew (2007) results are reliable, we compared their application in determining determinant result and also compared their results with other mathematical software i.e MATLAB.

V. CONCLUSION

The main idea of work for listing permutation is a starter sets generation. A new method to generate the starter sets are presented based on circular operation. Then this method is exploited to generate all n! permutations using CP and RoCP operation. The major difference of our strategies from other conventional permutation methods is that we exploit the starter sets to list all permutations. Our future work is to apply this method for generating distinct Hamiltonian Circuit from complete graph.

VI. ACKNOWLEDGEMENTS

The authors gratefully acknowledged the financial support received in the form of a Fundamental Research Grant Scheme (FRGS) (Code: 13046) from Universiti Utara Malaysia and Ministry of Education.

VII. REFERENCES

Borisenko, A. A., Kalashnikov, V. V., Kulik, I. A., & Goryachev, O. E. (2008). Generation of permutations based upon factorial numbers, IEEE, pp. 57-61.

Ibrahim, H., Omar, Z., & Rohni, A. M. (2010). New algorithm for listing all permutations. Modern Sciences , pp. 89-94.

Karim, S., Omar, Z., Ibrahim, H., Othman, K. I., & Suleiman, M. (2010). New Recursive Circular Algorithm for Listing All Permutations , Discovering Matematik, Vol. 32, no. 2, pp. 51-56.

Karim, S., Omar, Z. & Ibrahim, H. (2011). Integrated strategy for generating permutation. International Journal of Contemporary Mathematical Sciences, Vol. 6, no. 24, pp. 1167 - 1174.

Karim, S, Ibrahim, H., & Mohd Darus, M. (2017). Determining distinct circuit in complete graphs using permutation, AIP Conference Proceeding, Vol. 1905, Issue 1, 1-5.

Kuo, T. (2009). A New Method forGenerating Permutations in Lexicographic Order, Journal of Science and Engineering Technology, Vol. 5, No. 4, pp. 21-29.

Langdon, G. (1967). An Algorithm for generating permutations. Communication of ACM , 298-299.

Sedgewick, R. (1977). Permutation generation methods. Computing surveys , pp. 137-164.

Thongchiew, K. (2007). A computerize algorithm for generating permutation and it's application in determining a determinant. Proc.of World Academy of Science, Engineering and Technology, 21, pp. 178-183.