EPiC Series in Computing
Volume 91, 2023, Pages 98–107
Proceedings of 38th International Conference on Computers and Their Applications

Alternative Approach to Achieve a Solution of Derangement Problems by Dynamic Programming

Thitivatr PatanasakPinyo[1] and Adel Sulaiman[2]

[1] Faculty of Information and Communication Technology, Mahidol University
Salaya, Nakhon Pathom, 73170, Thailand
thitivatr.pat@mahidol.edu
[2] Najran University
Najran, Saudi Arabia
aaalsulaiman@nu.edu.sa

Abstract

Derangement is one well-known problem in the filed of probability theory. An instance of a derangement problem contains a finite collection $\mathscr{C}$ of n paired objects, $\mathscr{C} = \{(x_1, y_1), ..., (x_n, y_n)\}$. The derangement problem asks how many ways to generate a new collection $\mathscr{C}' \neq \mathscr{C}$ such that for each $(x_i, y_j) \in \mathscr{C}', i \neq j$. We propose an efficient dynamic programming algorithm that divides an instance of the derangement problem into several subproblems. During a recursive process of unrolling a subproblem, there exists a repeated procedure that allows us to make a use of a subsolution that has already been computed. We present the methodology to formulate a concept of this subproblem as well as parts of designing and analyzing an efficiency of the proposed algorithm.

1 Introduction

In the theory of probability and statistics, a derangement problem is a problem that asks how many ways we can generate a collection of paired objects such that it is completely different from the given input collection [2, 3]. For instance, given that there were three customers would like to stop by a restaurant for lunch. Each customer had his own umbrella. They dropped their umbrellas in a box before getting in the restaurant. After all of them finished their food, they came out and randomly picked an umbrella from the box. The derangement problem asks how many ways it can be where every customer did not get his own umbrella.

For detailed clarification, let us formulate this problem. Let X be a finite set that contains n customers, i.e., $X = \{x_1, x_2, x_3\}$ where x_i represents Customer i. Clearly, $|X| = 3$ for this problem instance. Let $Y = \{y_1, y_2, y_3\}$ be another finite set where y_i represents an umbrella possessed by Customer i. Similarly, $|Y| = 3$ for this instance. Let $\mathscr{C}$ be an input collection, i.e., $\mathscr{C} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$ since Customer i is mapped with Umbrella i at the very beginning. Since there were only three pairs, using a brute force methodology (or exhaustive search like [6]) to generate all possible collections $\mathscr{C}'$ that satisfy the condition of derangement was feasible. Recall that a collection $\mathscr{C}'$ that qualifies must satisfy the property:

$$\forall (x_i, y_j) \in \mathscr{C}', i \neq j$$

As we all can verify, the were totally two ways that we can generate a collection that satisfies the desired objective. The first collection, denoted as $\mathscr{C}_1'$, was defined as

$$\mathscr{C}_1' = \{(x_1, y_2), (x_2, y_3), (x_3, y_1)\}$$

$\mathscr{C}_1'$ can simply be interpreted as Customer 1 took Umbrella 2, Customer 2 took Umbrella 3, and Customer 3 took Umbrella 1. The second collection, $\mathscr{C}_2'$, was defined as

$$\mathscr{C}_2' = \{(x_1, y_3), (x_2, y_1), (x_3, y_2)\}$$

$\mathscr{C}_2'$ can be interpreted as Customer 1 took Umbrella 3, Customer 2 took Umbrella 1, and Customer 3 took Umbrella 2. Hence, for the case that $n = 3$, there are only two possible ways to generate satisfied collection ($\mathscr{C}_1'$ and $\mathscr{C}_2'$). Therefore, the solution of this instance of derangement problem is two. Table 1 summarizes all solutions of derangement problems, as well as instances, of cases when $n =$ 2,3, and 4, respectively. In Table 1, the column $!n$ represents a total number of solutions to the case n. Table 2 shows solutions to cases of small n.

Table 1: Total Solutions and Instances of Solutions for Small n.

n	$!n$	Instances $\mathscr{C}'$ of Solution
1	0	$\emptyset$
2	1	$\mathscr{C}_1' = \{(x_1, y_2), (x_2, y_1)\}$
3	2	$\mathscr{C}_1' = \{(x_1, y_2), (x_2, y_3), (x_3, y_1)\}$ $\mathscr{C}_2' = \{(x_1, y_3), (x_2, y_1), (x_3, y_2)\}$
4	9	$\mathscr{C}_1' = \{(x_1, y_2), (x_2, y_1), (x_3, y_4), (x_4, y_3)\}$ $\mathscr{C}_2' = \{(x_1, y_2), (x_2, y_3), (x_3, y_4), (x_4, y_1)\}$ $\mathscr{C}_3' = \{(x_1, y_2), (x_2, y_4), (x_3, y_1), (x_4, y_3)\}$ $\mathscr{C}_4' = \{(x_1, y_3), (x_2, y_1), (x_3, y_4), (x_4, y_2)\}$ $\mathscr{C}_5' = \{(x_1, y_3), (x_2, y_4), (x_3, y_1), (x_4, y_2)\}$ $\mathscr{C}_6' = \{(x_1, y_3), (x_2, y_4), (x_3, y_2), (x_4, y_1)\}$ $\mathscr{C}_7' = \{(x_1, y_4), (x_2, y_1), (x_3, y_2), (x_4, y_3)\}$ $\mathscr{C}_8' = \{(x_1, y_4), (x_2, y_3), (x_3, y_1), (x_4, y_2)\}$ $\mathscr{C}_9' = \{(x_1, y_4), (x_2, y_3), (x_3, y_2), (x_4, y_1)\}$

Table 2: Total Solutions for Small n.

n	1	2	3	4	5	6	7	8	9	10	11	12
$!n$	0	1	2	9	44	265	1854	14833	133496	1334961	14684570	176214841

Even though there are mathematical formulas that can precisely compute a solution to a derangement problem with input size $n, n \in \mathbb{N}$, it is still worth to come up with a meaningful and fully logical procedure by the use of a fundamental well-known algorithm, a dynamic programming. The dynamic programming is a concept that utilize a recursive algorithm by thinking ahead of how many finite possible solutions to every valid subproblem they can be and memorizing those such subsolutions for future use. The dynamic programming is mostly used to find an optimal solution to a given problem such as finding an optimal RNA secondary structure of a given RNA string.

We organized this paper as follows: Section 2 examines existing methodologies that relate to both derangement problem and dynamic programming. Section 3 constructs a concept of how we extract a subproblem that lies in the procedure of computing a solution to the derangement problem. Section 4 describes the proposed dynamic programming algorithm and its tractable efficiency. Finally, Section 5 summarizes the contribution described in this paper as well as feasible future applications.

2 Preliminaries

2.1 Derangement

2.1.1 Mathematical Approach in Probability Theory

In probability and counting theory, the derangement problem can also be called a hat-check problem. The problem can be described by given n people with n hats. The problem asks how many ways that we can return n hats to n people where no hat is returned to its owner. Several mathematicians have come up with formulas. Here is a list of selected formula to solve the derangement problem.

1. The first one formulate the hat-check problem by proposing that there are totally two mutually-exclusive cases for any $n \in \mathbb{N}$. Given that there are n people. Let x_i represent Person i where $i \in \mathbb{N}$ and $i \leq n$. Similarly, let y_i represent a hat that belongs to Person i. Let $\mathscr{C}'$ be a valid instance of a solution.

   - The first case happens when $\{(x_1, y_j), (x_j, y_1)\} \subseteq \mathscr{C}', j \in \mathbb{N}, 1 < j \leq n$. Since $(x_1, y_j) \in \mathscr{C}'$ and $(x_j, y_1) \in \mathscr{C}'$, then the problem is reduced to solving an instance of $n - 2$.
   - The second case happens when $(x_1, y_j) \in \mathscr{C}'$ but $(x_j, y_1) \notin \mathscr{C}'$. This case creates a smaller instance of a problem where $X_{\text{new}} = X \setminus \{x_1\}$ and $Y_{\text{new}} = Y \setminus \{y_j\}$. Clearly, $|X_{\text{new}}| = |Y_{\text{new}}| = n - 1$.

   According to the two cases, a formula could be formulated as a recursive procedure:

   $$!n = (n-1)(!(n-1)+!(n-2)), n \geq 2$$

   where $!n$ represents the number of solutions of the problem with n people and $!1 = 0$ and $!0 = 1$ [15]. For the other two formulas, we will mention only the formula itself.

2. The second formula is formulated using the same base idea as the previous formula:

   $$!n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}, n \geq 0$$

3. The third formula is defined as:

   $$!n = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor$$

   where $\lfloor x \rfloor$ is a floor function on an input $x \in \mathbb{R}$ [16].

2.1.2 Bell's Number

A Bell's number [13], $\mathbb{B} = (B_1, B_2, B_3, ...)$, is a sequence of natural numbers where B_n represents a total number of valid partitions $\mathscr{P}$ over a finite set S where $|S| = n$. In order to compute an asymptotic expansion of $!n$, there is another formula of the upper bound of $!n$ that implementing a Bell's number B_n:

$$!n - \frac{n!}{e} - \sum_{k=1}^{m} (-1)^{n+k-1} \frac{B_k}{n^k} = O\left(\frac{1}{n^{m+1}}\right)$$

2.1.3 Concatenation of Permutation Functions

Let us define a concatenation AB of two finite set, $A = \{x_1, x_2, ..., x_n\}$ and $B = \{y_1, y_2, ..., y_n\}$ as:

$$AB = \{(x_i, y_j) \in A \times B\}$$

that comes with constraints:

1. $\forall x_i \in A, x_i$ must show up in exactly one ordered pair in AB.
2. $\forall y_j \in B, y_j$ must show up in exactly one ordered pair in AB.
3. $\forall (x_i, y_j) \in AB, i \neq j$.

We can view AB as a bijection from $A \to B$. Computing a total ways that we can create AB is equivalent to an objective of a derangement problem. In order to compute using this concept, we have to run an algorithm to generate all permutations $\mathscr{P}_B$ of B by using permutation functions $f_B : B \to B$. After we get all candidates, we construct all possible $AB, B \in \mathscr{P}_B$. We visit every candidate of AB and remove one that does not satisfy all three constraints of concatenation. The remaining AB candidates that qualify are precisely instances of solution of the derangement. We use the case of $n = 3$ as an example to simulate the algorithm. Let $A = \{x_1, x_2, x_3\}$ and $B = \{y_1, y_2, y_3\}$. All permutation functions of B, $\mathscr{P}_B$, are:

$$\begin{aligned} \mathscr{P}_B &= \{B_1, B_2, B_3, B_4, B_5, B_6\} \qquad\qquad , \text{ where} \\ B_1 &= \{y_1, y_2, y_3\} \\ B_2 &= \{y_1, y_3, y_2\} \\ B_3 &= \{y_2, y_1, y_3, \} \\ B_4 &= \{y_2, y_3, y_1\} \\ B_5 &= \{y_3, y_1, y_2\} \\ B_6 &= \{y_3, y_2, y_1\} \end{aligned}$$

Although $|\mathscr{P}_B| = 6$, not all of them can be a valid candidate of AB. For example, $AB_2 = \{(x_1, y_1), (x_2, y_3), (x_3, y_2)\}$ is not a valid concatenation since $(x_1, y_1) \in AB_2$. From the above work, we have two valid AB candidates left after filtering out all the unqualified sets, which are $AB_4 = \{(x_1, y_2), (x_2, y_3), (x_3, y_1)\}$ and $AB_5 = \{(x_1, y_3), (x_2, y_1), (x_3, y_2)\}$. Therefore, there are two valid solutions for $n = 3$. Even though this methodology returns both number of solutions and corresponded instances of solution, its running time is not efficient enough since each permutation requires $n!$ and it has been proven that $n! \notin \Theta(n^k), k \in \mathbb{Z}$ [4].

2.2 Dynamic Programming

A dynamic programming (DP) is an algorithm that can be implemented to solve many computer science problems. Most algorithm that implement DP are efficient. The concept of DP that makes it dissimilar to traditional recursive algorithm is that once DP successfully computes a subsolution of a subproblem, it does memorize the subsolution for further use. Memorizing subsolutions in a data storage, e.g., an array, help improving the running time when dealing with the same subproblem in the future. Retrieving a subsolution from the array costs us $O(n)$ where n is the size of the current array, which is considered efficient. Practical examples of problems that have a DP algorithm to solve can be found in [10].

3 Extracting a Subproblem of Derangement

We initiate our concept by thinking about the most simple case, which is the case that $n = 2$, i.e., $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2\}$. Let $\mathscr{S}_2$ denote all possible ways of permutation of Y where $|Y| = 2$. Clearly, $\mathscr{S}_2 = 2! = 2$. Particularly, all permutations of Y are $\{y_1, y_2\}$ and $\{y_2, y_1\}$. The only valid instance of the derangement of this case is $\{(x_1, y_2), (x_2, y_1)\}$. Therefore, the number of solutions of the case $n = 2$ is just one way. We define $C_{(i,j)}$ to denote the total number of ways that there are j out of i people that do not get their umbrellas. Clearly, $C_{(2,2)}$ is nothing but a total number of solutions of the derangement that we would like to know for this case. In general, for a given derangement problem of n people, the objective is just to compute $C_{(n,n)}$.

Let us list all possible permutations of Y and observe the result when making a collection of $(x_i, y_i), x_i \in X, y_i \in Y$. The first collection that we can create is $\{(x_1, y_1), (x_2, y_2)\}$ and the second collection is $\{(x_1, y_2), (x_2, y_1)\}$. The total number of collections that we get must equal $\mathscr{S}_2$. We can see that the first collection is the case that every umbrella is taken by its owner. Hence, $C_{(2,0)} = 1$. Similarly, the second collection is when no umbrella is taken by its owner, which is the instance of the solution of derangement that we want, and it makes $C_{(2,2)} = 1$.

Let examine one more case when $n = 3$ in order to reveal more solid pattern. For this round, $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$. $\mathscr{S}_3 = 3! = 6$. Thus, there are six collections that we can create:

$$\begin{aligned}
\mathscr{C}'_1 &= \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\} \\
\mathscr{C}'_2 &= \{(x_1, y_1), (x_2, y_3), (x_3, y_2)\} \\
\mathscr{C}'_3 &= \{(x_1, y_2), (x_2, y_1), (x_3, y_3)\} \\
\mathscr{C}'_4 &= \{(x_1, y_2), (x_2, y_3), (x_3, y_1)\} \\
\mathscr{C}'_5 &= \{(x_1, y_3), (x_2, y_1), (x_3, y_2)\} \\
\mathscr{C}'_6 &= \{(x_1, y_3), (x_2, y_2), (x_3, y_1)\}
\end{aligned}$$

$\mathscr{C}'_1$ is the only case that every umbrella is taken by its owner, so $C_{(3,0)} = 1$. There are two cases that nobody gets its own umbrella, which are $\mathscr{C}'_4$ and $\mathscr{C}'_5$. Hence, $C_{(3,3)} = 2$. The remaining collections ($\mathscr{C}'_2$, $\mathscr{C}'_3$, and $\mathscr{C}'_6$) represent cases when there is exactly one umbrella taken by its owner, thus $C_{(3,2)} = 3$.

From both cases, $n = 2$ and $n = 3$, we can see that the permutation of n itself equals a summation of all collections $\mathscr{C}'_i$ that could be created. Particularly, for $n = 2$:

$$\mathscr{S}_2 = |\{(x_1, y_1), (x_2, y_2)\}| + |\{(x_1, y_2), (x_2, y_1)\}| \tag{1}$$

$$\mathscr{S}_2 = C_{(2,0)} + C_{(2,2)} \tag{2}$$

And for $n = 3$:

$$\mathscr{S}_3 = |\mathscr{C}'_1| + |\mathscr{C}'_2 \cup \mathscr{C}'_3 \cup \mathscr{C}'_6| + |\mathscr{C}'_4 \cup \mathscr{C}'_5| \tag{3}$$
$$= C_{(3,0)} + C_{(3,2)} + C_{(3,3)} \tag{4}$$

Lemma 1 For $n \in \mathbb{N}$, a collection $\mathscr{C}'_i$ of valid permutations of Y must belong to exactly one set $S_k, 0 \le k \le n$, such that $|S_k| = C_{(n,k)}$.

Proof Suppose there exists a collection $\mathscr{C}'_i$ such that it belongs to S_k and $S_m, m \neq k$, at the same time. Since $\mathscr{C}'_i \in S_k$, there are k umbrellas that are not taken by their owners, i.e., there are k numbers of (x_a, y_b) pairs that $a \neq b$ in $\mathscr{C}'_i$. Since $\mathscr{C}'_i \in S_m$, there are m umbrellas that are not taken by their owners, i.e., there are m numbers of (x_a, y_b) pairs that $a \neq b$ in $\mathscr{C}'_i$. To have these two statements satisfy true, k must equal m. So there is a contradiction. Hence, proved ■

Theorem 2 For $n \in \mathbb{N}$, $!n = \mathscr{S}_n - \sum_{i=0}^{n-1} C_{(n,i)}$

Proof For $n \in \mathbb{N}$:

$$\mathscr{S}_n = C_{(n,0)} + C_{(n,1)} + C_{(n,2)} + \cdots + C_{(n,n)} \qquad \because \textbf{Lemma 1} \tag{5}$$
$$\mathscr{S}_n = C_{(n,0)} + C_{(n,1)} + C_{(n,2)} + \cdots + C_{(n,n-1)} + C_{(n,n)} \tag{6}$$
$$\mathscr{S}_n = \sum_{i=0}^{n-1} C_{(n,i)} + C_{(n,n)} \tag{7}$$
$$\mathscr{S}_n = \sum_{i=0}^{n-1} C_{(n,i)} + !n \tag{8}$$
$$!n = \mathscr{S}_n - \sum_{i=0}^{n-1} C_{(n,i)} \qquad \blacksquare \tag{9}$$

Lemma 3 For $n \in \mathbb{N}, C_{(n,1)} = 0$

Proof It is obvious that for any collection, there must be at least 2 umbrellas that are not taken by their owners or every umbrella is correctly taken by its owner. The case only one umbrella is left not taken by its owner cannot happen. Hence, proved ■

After we get Theorem 2, we are going to extract a subproblem. Again, we use the case $n = 3$ to initialize a concept. Recall Theorem 2on $n = 3$:

$$!3 = \mathscr{S}_3 - \sum_{i=0}^{3-1} C_{(3,i)} \tag{10}$$
$$!3 = 3! - \left(C_{(3,0)} + C_{(3,1)} + C_{(3,2)}\right) \tag{11}$$
$$!3 = 3! - \left(C_{(3,0)} + 0 + C_{(3,2)}\right) \qquad \because \textbf{Lemma 3} \tag{12}$$
$$!3 = 3! - \left(C_{(3,0)} + C_{(3,2)}\right) \tag{13}$$
$$!3 = 3! - \left(1 + C_{(3,2)}\right) \qquad \because C_{(3,0)=1} \tag{14}$$

From the last equation (Line 14), the term $C_{(3,2)}$ represents the case that there are two umbrellas (out of three umbrellas) that not taken by their owners. This term initiate a subproblem. In order to solve $C_{(3,2)}$, let us elaborate the situation. Having two umbrellas incorrectly taken means that there is one umbrella that is correctly taken by its owner. So we can recursively compute $C_{(3,2)}$ by:

$$C_{(3,2)} = \binom{3}{1} C_{(2,2)} \tag{15}$$

where $\binom{3}{1}$ represents total ways that there is only one umbrella correctly taken and $C_{(2,2)}$ represents the total number of derangement solutions for $n = 2$, i.e, $!2$, which is 1. Let us plug in every value we get back in the Line 14 to confirm the correctness:

$$
\begin{aligned}
!3 &= 3! - \left(1 + C_{(3,2)}\right) && (16)\\
&= 6 - \left(1 + \binom{3}{1} C_{(2,2)}\right) && (17)\\
&= 6 - (1 + (3)(1)) && (18)\\
&= 2 \qquad \blacksquare && (19)
\end{aligned}
$$

Next, let us observe the case of $n = 4$ using the same methodology:

$$
\begin{aligned}
!4 &= 4! - \left(C_{(4,0)} + C_{(4,1)} + C_{(4,2)} + C_{(4,3)}\right) && (20)\\
!4 &= 4! - \left(1 + C_{(4,2)} + C_{(4,3)}\right) && (21)
\end{aligned}
$$

Again, we can see that two terms, $C_{(4,2)}$ and $C_{(4,3)}$, are the parts that the subproblems (that we have already computed the subsolution) kick in. The equation can be re-written to:

$$
\begin{aligned}
!4 &= 4! - \left(1 + \binom{4}{2} C_{(2,2)} + \binom{4}{1} C_{(3,3)}\right) && (22)\\
!4 &= 4! - \left(1 + \binom{4}{2} !2 + \binom{4}{1} !3\right) && (23)\\
!4 &= 24 - (1 + (6)(1) + (4)(2)) && (24)\\
!4 &= 9 \qquad \blacksquare && (25)
\end{aligned}
$$

Lastly, we simulate the last example with the case $n = 5$. However, this time we show every subsolution being stored every time we compute a subproblem. Let $\mathbb{O}$ be an array of integers where $OPT[n]$ represents $!n$. At first, OPT is (0) since $!1 = 0$.

- Initialize

$$
\begin{aligned}
!5 &= 5! - \left(C_{(5,0)} + C_{(5,1)} + C_{(5,2)} + C_{(5,3)} + C_{(5,4)}\right) && (26)\\
\mathbb{O} &\leftarrow (0) && (27)
\end{aligned}
$$

- Step 1: Compute $C_{(5,0)}$

$$
\begin{aligned}
!5 &= 5! - \left(1 + C_{(5,1)} + C_{(5,2)} + C_{(5,3)} + C_{(5,4)}\right) && (28)\\
\mathbb{O} &\leftarrow (0) && (29)
\end{aligned}
$$

- Step 2: Compute $C_{(5,1)}$
  By **Lemma 3**

$$
\begin{aligned}
!5 &= 5! - \left(1 + 0 + C_{(5,2)} + C_{(5,3)} + C_{(5,4)}\right) && (30)\\
\mathbb{O} &\leftarrow (0) && (31)
\end{aligned}
$$

- Step 3: Compute $C_{(5,2)}$
  Since $C_{(5,2)} = \binom{5}{3} C_{(2,2)}$ and $C_{(2,2)} = 1$. We add $C_{(2,2)} = 1$ to $\mathbb{O}[2]$.

$$
\begin{aligned}
!5 &= 5! - \left(1 + 0 + 10 + C_{(5,3)} + C_{(5,4)}\right) && (32)\\
\mathbb{O} &\leftarrow (0, 1) && (33)
\end{aligned}
$$

- Step 4: Compute $C_{(5,3)}$
  Since $C_{(5,3)} = \binom{5}{2}C_{(3,3)}$, in order to compute $C_{(3,3)}$, we recall $C_{(2,2,)}$ from OPT (by Equation in Line 17). We add $C_{(3,3)} = 2$ to $\mathbb{O}[3]$.

$$!5 = 5! - \big(1 + 0 + 10 + 20 + C_{(5,4)}\big) \tag{34}$$
$$\mathbb{O} \leftarrow (0, 1, 2) \tag{35}$$

- Step 5: Compute $C_{(5,4)}$
  Since $C_{(5,4)} = \binom{5}{1}C_{(4,4)}$, in order to compute $C_{(4,4)}$, we recall $C_{(2,2)}$ and $C_{(3,3,)}$ from OPT (by Equation in Line 23). We add $C_{(4,4)} = 9$ to $\mathbb{O}[4]$.

$$!5 = 5! - (1 + 0 + 10 + 20 + 45) \tag{36}$$
$$\mathbb{O} \leftarrow (0, 1, 2, 9) \tag{37}$$

- Step 5: Compute !5

$$!5 = 5! - (1 + 0 + 10 + 20 + 45) \tag{38}$$
$$!5 = 44 \qquad \blacksquare \tag{39}$$

4 Design and Analysis of the Proposed Dynamic Programming Algorithm

4.1 Design an Algorithm

Algorithms 1 and 2 completely describes the procedure to compute the number of solutions of the derangement problem with input size n that we have described in Section 3.

Algorithm 1 Dynamic Programming Algorithm for Derangement Problem.

```
function TOTAL-DERANGEMENT(n)
    return REC-DERANGE(n, n)
end function
```

4.2 Analyze the Algorithm

Algorithm 2 contains two parts that involves in the total running time calculation. The first part is the iteration in Lines 13 to 15. Since the iteration repeats at most n times, this part has $O(n)$ running time. Another part is the part that does recursive call in Line 9. However, with the implementation of the dynamic programming concept, the time consumed in this part equals the time that we do a linear search on array O (which has possibly maximum size of n), which is $O(n)$. Therefore, the overall running time of Algorithm 2 is $O(n^2)$. Since the algorithm has a polynomial running time, it is theoretically considered efficient.

5 Conclusion and Feasible Future Applications

We propose an algorithm that can precisely return the total number of solutions of a derangement problem with an input $n \in \mathbb{N}$ by using a concept of dynamic programming, which known

Algorithm 2 Algorithm for Subproblem of Derangement.

```
function Rec-Derange(n, i)
    if i = 0 then
        return 1
    end if
    if i = 1 then
        return 0
    end if
    if 1 < i ≤ n − 1 then
        return (n choose n−i) REC-DERANGE(i, i)        ▷ Get the value from the array 𝕆
    end if
    if i = n then
        sum ← 0
        for j = 0 : n − 1 do
            sum ← sum + REC-DERANGE(n, j)
        end for
        return n! − sum
    end if
end function
```

to be efficient most of the time. The derangement problem with inputs $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$ simply asks how many ways we can generate a collection $\mathscr{C}' = \{(x_i, y_j) \in X \times Y\}$ such that $i \neq j$ and every $x_i \in X$ shows up exactly once as well as every $y_j \in Y$. We start by simulate a methodology to compute total ways for the cases of small n to extract a recursive procedure that is hidden as a subproblem. We discover that we can reuse what we have already computed in the previous recursive call, which guides us to recall the concept of dynamic programming. The output of the proposed algorithm is shown to be precise and the algorithm itself comes with a polynomial running time that guarantees its efficiency.

The derangement problem can lead to several applications that need to rearrange a finite set such that the original arrangement is expected not to be returned. The example of such applications is designing a sequence of addresses for the address the verification task given the specific starting address [1, 5, 7, 8]. Another good example is designing a matching problems for the paper folding test, which is a test of individual spatial visualization (VZ) [9, 11, 12, 14].

References

[1] Georgi Batinov, Michelle Rusch, Tianyu Meng, Kofi Whitney, Thitivatr Patanasakpinyo, Les Miller, and Sarah Nusser. Understanding map operations in location-based surveys. In *Eighth International Conference on Advances in Computer-Human Interactions (ACHI 2015)*, pages 144–149, Lisbon, Portugal, 2015. International Academy, Research, and Industry Association (IARIA).

[2] Mehdi Hassani. Derangements and applications. *Journal of Integer Sequences*, 6(1):03–1, 2003.

[3] Mehdi Hassani. Derangements and alternating sum of permutations by integration. *Journal of Integer Sequences*, 23(2):3, 2020.

[4] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson Education India, 2006.

[5] Thitivatr PatanasakPinyo. *Flattening methods for adaptive location-based software to user abilities*. Graduate Theses and Dissertations, Iowa State University, 2017.

[6] Thitivatr Patanasakpinyo. Model checking approach for deadlock detection in an operating system process-resource graph using dynamic model generating and computation tree logic specification. In Gordon Lee and Ying Jin, editors, *Proceedings of 34th International Conference on Computers and Their Applications*, volume 58 of *EPiC Series in Computing*, pages 55–64. EasyChair, 2019.

[7] Thitivatr Patanasakpinyo. Ameliorating accuracy of a map navigation when dealing with different altitude traffics that share exact geolocation. In Alex Redei, Rui Wu, and Frederick Harris, editors, *SEDE 2020. 29th International Conference on Software Engineering and Data Engineering*, volume 76 of *EPiC Series in Computing*, pages 95–104. EasyChair, 2021.

[8] Thitivatr PatanasakPinyo. Exploiting a real-time non-geolocation data to classify a road type with different altitudes for strengthening accuracy in navigation. *International Journal of Computers and Their Applications*, 28(1):55–64, 2021.

[9] Thitivatr PatanasakPinyo, Georgi Batinov, Kofi Whitney, and Les Miller. Methods that flatten the user space for individual differences in location-based surveys on portable devices. In *31st International Conference on Computers and Their Applications (CATA 2016)*, pages 65–70, Las Vegas, Nevada, 2016. International Society for Computers and their Applications (ISCA).

[10] Thitivatr Patanasakpinyo, Georgi Batinov, Kofi Whitney, Adel Sulaiman, and Les Miller. Object-indexing: A solution to grant accessibility to a traditional raster map in location-based application to accomplish a location-based task. *International Journal of Computing, Communication and Instrumentation Engineering (IJCCIE)*, 5(1), 2018.

[11] Thitivatr Patanasakpinyo, Georgi Batinov, Kofi Whitney, Adel Sulaiman, and Les Miller. Enhanced prediction models for predicting spatial visualization (vz) in address verification task. In Gordon Lee and Ying Jin, editors, *Proceedings of 34th International Conference on Computers and Their Applications*, volume 58 of *EPiC Series in Computing*, pages 247–256. EasyChair, 2019.

[12] Thitivatr Patanasakpinyo and Les Miller. Ui error reduction for high spatial visualization users when using adaptive software to verify addresses. In Gordon Lee and Ying Jin, editors, *Proceedings of 35th International Conference on Computers and Their Applications*, volume 69 of *EPiC Series in Computing*, pages 22–31. EasyChair, 2020.

[13] Stefano Pironio, Antonio Acín, Serge Massar, A Boyer de La Giroday, Dzmitry N Matsukevich, Peter Maunz, Steven Olmschenk, David Hayes, Le Luo, T Andrew Manning, et al. Random numbers certified by bell's theorem. *Nature*, 464(7291):1021–1024, 2010.

[14] Timothy A Salthouse, Renee L Babcock, Debora RD Mitchell, Roni Palmon, and Eric Skovronek. Sources of individual differences in spatial visualization ability. *Intelligence*, 14(2):187–230, 1990.

[15] Richard P Stanley. What is enumerative combinatorics? In *Enumerative combinatorics*, pages 1–63. Springer, 1986.

[16] Eric W Weisstein. Derangement. *https://mathworld. wolfram. com/*, 2002.