Toward an Extension of Efficient Algorithm to Solve Derangement Problems by Dynamic Programming Approach

Thitivatr PatanasakPinyo*
Mahidol University, Salaya, Nakhon Pathom 73170, THAILAND
Adel Sulaiman†
Najran University, Najran, SAUDI ARABIA

Abstract

In statistics, probability theory, and computer science, a derangement, $!n$, is known to be a basic problem that computes total ways of rearranging $n \in \mathbb{N}$ items such that a result contains no item i that stands in the same position as it did in the input. Formally, the derangement problem has a problem instance of a finite collection $\mathscr{C} = \{(x_1,y_1),....,(x_n,y_n)\}, |\mathscr{C}| = n$. With this formulation, $!n$ is a total number of qualified collections $\mathscr{C}'$ where $\mathscr{C}'$ contains n members of (x_i,y_j) where $i \neq j$ and every x_i and y_j $(1 \leq i,j \leq n)$ must show up exactly once in $\mathscr{C}'$. In this article, we present a dynamic programming algorithm that computes $!n$ and its justification. We also provide a discussion about extending the limitation of the problem with an objective to cover a general case where we have a finite set of variables $a_1,a_2,....,a_k$ rather than the traditional scenario that has only two variables: x and y.

Key Words: derangement, dynamic programming, recursion, algorithm, complete derangement

1 Introduction

The total ways of an arrangement of $n \in \mathbb{N}$ distinct items is a fundamental statement that has been brought to introduce the concept of factorial ($n!$). A derangement problem ($!n$) has the same input instance as $n!$, which is a finite set that contains n distinct items, i.e., $\{x_1,x_2,....,x_n\}$. The derangement problem asks to compute how many ways of an arrangement of these n items such that for every item x_i, it is not located on location i [2, 3]. For the case of $n = 1$, it is impossible. For the case of $n = 2$, there is only one way to do, i.e., $!2 = 1$. Particularly, let us illustrate the case of $n = 2$. Let the input instance be $\{x_1,x_2\}$. There are two possible arrangements of this input. The first arrangement is (x_1,x_2), which is obviously not qualified for a derangement. The second arrangement is (x_2,x_1), which is a valid derangement. Thus, $!2 = 1$.

Without loss of generality, we create a new viewpoint to the derangement problem by modifying an instance of the problem. Particularly, the input instance contains a collection $\mathscr{C} = \{(x_1,y_1),....,(x_n,y_n)\}, |\mathscr{C}| = n$. The objective of the derangement problem will be treated as computing total ways that we can create a collection $\mathscr{C}' = \{(x_i,y_j)\}, 1 \leq i,j \leq n$ with the following constraints:

1. $|\mathscr{C}'| = n$.
2. $\forall i, 1 \leq i \leq n, x_i$ must show up in $\mathscr{C}'$ exactly once.
3. $\forall j, 1 \leq j \leq n, y_j$ must show up in $\mathscr{C}'$ exactly once.
4. $\forall (x_i,y_j) \in \mathscr{C}', i \neq j$

With the new input instance, we use an umbrella problem, which can be found as an exercise in several textbooks about probability theory and statistics, to illustrate how it represents the derangement problem. The umbrella problem states that there are $n \in \mathbb{N}$ customers, which are $x_1,x_2,....,x_n$. Each customer x_i has his own umbrella y_i. Every customer must drop his umbrella into the box in front of the restaurant. After every customer finishes his meal, he will take an umbrella from the box and leave. The umbrella problem asks us to compute how many ways that **no** customer x_i gets his umbrella y_i. This is obvious that it is equivalent to compute all possible collections $\mathscr{C}'$ that we have previously mentioned where $x_1,....,x_n$ represent n customers and $y_1,....,y_n$ represents n umbrellas where Umbrella y_k belongs to Customer x_k at the beginning. Table 1 shows $!n$ and its corresponded collections $\mathscr{C}'$ for $n = 1$ to 4. We also show results of $!n$ for small n in Table 2.

Even though it is impractical to list every valid $\mathscr{C}'$ when n is bigger, computing total number of every valid $\mathscr{C}'$ is still feasible. Several methodologies were suggested to compute $!n$, mostly were intricate mathematical formulas. In this paper, we have come up with an efficient algorithm that can solve $!n$ by implementing a dynamic programming approach, which is a technique that is widely used in the field of algorithm design. Many problems have been known to have an efficient dynamic programming algorithm, which is way better than a traditional recursive algorithm, e.g., the famous Fibonacci series where an

*Faculty of Information and Communication Technology. Email: thitivatr.pat@mahidol.edu.
†College of Computer Science and Information Systems. Email: aaalsulaiman@nu.edu.sa.

Table 1: Total Valid Collections $\mathscr{C}'$ for Different n.

n	$!n$	$\mathscr{C}'$
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: Results of $!n$ for Small n.

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

objective is to compute a member of the series at any index $i \in \mathbb{N}$ has an efficient dynamic programming algorithm.

This paper follows the following structure. Section 2 reveals related work, which consists of descriptions and past studies about the derangement problem and the dynamic programming approach, with an objective to provide readers a story of the topic and its challenge in an aspect of algorithm study and theoretical computer science. Section 3 explains how we came up with a conceptualization to solve the derangement problem by formulating it as a recurrence problem that each layer has another subproblem inside. Section 4 illustrates the proposed algorithm along with efficiency analysis. Section 5 extends the work by initiating an idea to design a universal algorithm that can effectively solve a generalized version of the derangement problem. Section 6 summarizes all the work being presented in this paper and shed a light to possible future work.

2 Preliminaries

2.1 Derangement

2.1.1 Solving Problem Using Mathematical Approach

The derangement problem hes been interchangeably known as the **hat-check** problem where the instance of the problem contains $n \in \mathbb{N}$ people and each person has his own hat. The problem asks how many ways to assign the hat to each person such that no one gets his own hat. Multiple mathematical methodologies have been suggested. Here are some of them that we select to mention.

1. The first methodology treats the hat-check problem as two mutual-exclusive incidents. Particularly, let $X = \{x_1,x_2,...,x_n\}$ be a set of all n people and $Y = \{y_1,y_2,...,y_n\}$ be a set of all hats where x_i owns y_i. With this setup, all the derangement results can be partitioned into:
   (a) For a certain x_i, x_i is assigned y_j and x_j is assigned y_i.
   (b) For a certain x_i, x_i is assigned y_j but x_j is assigned y_k where $i \neq k$.

   For the first incident, it is equivalent to the exchange of hats between x_i and x_j. This causes the problem to be reduce to a smaller instance of the hat-check problem with a size $n-2$. The second incident also leads to a smaller instance where $X_{\text{new}} = X \setminus \{x_i\}$ and $Y_{\text{new}} = Y \setminus \{y_j\}$. X_{new} is a set of all people for this new instance. Y_{new} is a set of hats for the new instance. Combining both incidents together, a formula for $!n$ shall be constructed as:
   $$!n = (n-1)(!(n-1)+!(n-2)), n \geq 2$$
   where $!1 = 0$ and $!0 = 1$ [17].
2. The second methodology is simply a formula that implementing a summation, particularly,
   $$!n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}, n \geq 0$$
3. Another formula that can also return a solution of $!n$ is defined using a floor function as:
   $$!n = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$$
   where e is a base of natural logarithm and $\lfloor x \rfloor$ is the maximum $z \in \mathbb{Z}$ subject to $z \leq x$ [18].

We are going to discuss in more detail about the hat-check problem later in this paper.

2.1.2 Formulating Problem Using Permutation Function

Let A and B be two finite sets where $A = \{x_1,x_2,...,x_n\}$ and $B = \{y_1,y_2,...,y_n\}$. We define a concatenation of A and B (denoted by $A \cdot B$ or AB) as
$$A \cdot B = \{(x_i,y_j) \in A \times B\}$$

subject to:

1. $\forall x_i \in A$, x_i must show up in exactly one member of $A \cdot B$
2. $\forall y_j \in A$, y_j must show up in exactly one member of $A \cdot B$
3. $\forall (x_i, y_j) \in A \cdot B, i \neq j$.

The concatenation $A \cdot B$ can be perceived as a bijection $A \rightarrow B$. The total ways that we can create $A \cdot B$ is equivalent to a solution of $!n$. To apply this methodology, we run an algorithm that generate all permutations on B. Next, we do a concatenate of A on each B returned from the algorithm. If the result of $A \cdot B$ satisfies all constraints of concatenation, we then keep it. Otherwise, we omit it. The total number concatenations qualified is nothing but $!n$. For instance, we use the case of $n = 3$. The instance of the problem are $A = \{x_1, x_2, x_3\}$ and $B = \{y_1, y_2, y_3\}$. All valid permutations of B are:

1. $B_1 = \{y_1, y_2, y_3\}$
2. $B_2 = \{y_1, y_3, y_2\}$
3. $B_3 = \{y_2, y_1, y_3\}$
4. $B_4 = \{y_2, y_3, y_1\}$
5. $B_5 = \{y_3, y_1, y_2\}$
6. $B_6 = \{y_3, y_2, y_1\}$

From the six permutations of B, only two of them can form a precise concatenations, which are $A \cdot B_4 = \{(x_1, y_2), (x_2, y_3), (x_3, y_1)\}$ and $A \cdot B_5 = \{(x_1, y_3), (x_2, y_1), (x_3, y_2)\}$. Therefore, $!3 = 2$. Although this methodology guarantees to return a correct answer but its performance in an aspect of running time is not quite efficient since its complexity is $n!$ and $n! \notin \Theta(n^k), k \in \mathbb{Z}$, i.e., it is not polynomial-bound.

2.1.3 Relationship with Bell's Number

Recall that a Bell's number is an infinite sequence of a natural number [15]. Let $\mathbb{B} = (b_1, b_2, ...)$ be a sequence of Bell's number. We define b_i, $i \in \mathbb{N}$, by:

b_i = the total number of partitions of a finite set S where $|S| = i$

With this definition, $\mathbb{B}$ looks like:

$$\mathbb{B} = (1, 2, 5, 15, 52, 203, ...)$$

There exists a formula [18] to compute $!n$ by using $\mathbb{B}$, particularly, the formula is:

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

2.2 Dynamic Programming

A dynamic programming (DP) is a well known algorithm that implements a concept of recursion plus a usage of a memory to memorize a sub-solution previously computed by a recursive call for future use. The design of DP can be divided into two types, which are either top-down or bottom-up. Even though both always returns a correct solution, choosing between top-down or bottom up most is mostly based on the structure and the way that we formulate the problem. Several problems in both mathematics and computer science have an efficient algorithm that implementing DP as a backbone, e.g., Computing a Fibonacci number at arbitrary index i or solving a maximum-weight interval scheduling problem or solving a deadlock problem in an operating system [4, 7].

3 Understanding the Basis of Derangement

There is no better way to design a DP algorithm than unrolling the instance of the problem to the most inner layer and revealing the hidden pattern of the tentative solution. With this guidance, we start with the most basic case, particularly, the solution of $!n$ when $n = 2$. The instance of the case $n = 2$ contains $X = \{x_1, x_2\}$ and $y = \{y_1, y_2\}$. Clearly, there is only one valid candidate of $\mathscr{C}'$, which is

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

This can be interpreted as Person x_1 gets Umbrella y_2 and Person x_2 gets Umbrella y_1. Hence, $!2 = 1$. To eliminate any ambiguity, let us define $C_{(i,j)}$ to be a total ways of assigning umbrellas such that $j \in \mathbb{N}$ out of $i \in \mathbb{N}$ people do not get their umbrellas. Equivalently, a solution of $!2$ is nothing but $C_{(2,2)}$. Hence, for any case of $n \in \mathbb{N}$, $!n = C_{(n,n)}$. We also define $\mathbb{S}_n$ for future use where $\mathbb{S}_n = n!$. The purpose of the symbol is just to simplify the proof in the next section. Therefore, with this basic case that $n = 2$, we can write:

$$!2 = C_{(2,2)} = 1 = \mathbb{S}_2 - 1$$

Although we have seen that there is only one candidate of $\mathscr{C}$ when $n = 2$ as described previously, it is worth here to mention all possible $X \cdot Y$ concatenations if there exists any clue that can lead us to the base case for DP algorithm. Since $n = 2$, there are totally $2! = 2$ permutations for Y, which are $\{y_1, y_2\}$ and $\{y_2, y_1\}$. Hence, all possible cross products of $X \times Y$ are:

$$\mathscr{C}'_1 = \{(x_1, y_1), (x_2, y_2)\}$$
$$\mathscr{C}'_2 = \{(x_1, y_2), (x_1, y_1)\}$$

We can see that there is only one concatenation, which is $\mathscr{C}'_2 = \{(x_1, y_2), (x_2, y_1)\}$. So the fact that $!2 = \mathbb{S}_2 - 1 = C_{(2,2)} = 1$ still has no change. The interesting question is, what else we can tell about $\mathscr{C}'_1 = \{(x_1, y_1), (x_2, y_2)\}$. It is quite straightforward that $\mathscr{C}'_1$ represents the case that at least one person gets his own umbrella. Since $n = 2$, the case that at least one person gets his umbrella is the same as the case that every person gets his umbrella, which can be represented by $\mathscr{C}_{(2,0)}$. We then can construct the following equation to represent this scenario:

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

We are going to observe one more example when $n = 3$ to witness if there exists any difference comparing to the previous

case. For $n = 3$, the instance of the problem consists of $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$. All valid permutations are:

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

From the list, out of $3! = 6$ permutations/assignments, only $\mathscr{C}'_5$ and $\mathscr{C}'_6$ are valid derangement. Hence, $!3 = 2$. However, all six permutations can be partitioned into a set A_i by the following manner:

$$\mathscr{C}'_k \in A_i \iff i \text{ people do not get their own umbrellas in } \mathscr{C}'$$

Following this criteria that we use to do A_i partition, it is clear that $|A_i| = C_{(3,i)}$. Particularly:

$$\begin{aligned}
A_3 &= \{\mathscr{C}'_5, \mathscr{C}'_6\} \\
A_2 &= \{\mathscr{C}'_2, \mathscr{C}'_3, \mathscr{C}'_4\} \\
A_1 &= \emptyset \\
A_0 &= \{\mathscr{C}'_1\}
\end{aligned}$$

With th definition of A_i, $|A_3|$ is equivalent to $!3$. Since it fits the concept of partition, we can construct the equation:

$$\begin{aligned}
\mathbb{S}_3 &= \sum_{i=0}^{3} |A_i| \\
\mathbb{S}_3 &= |A_3| + |A_2| + |A_1| + |A_0| && \because \text{partition} \\
|A_3| &= \mathbb{S}_3 - (|A_2| + |A_1| + |A_0|) \\
2 &= 6 - (3 + 0 + 1) && \because \mathbb{S}_3 = 3! \\
2 &= 2 && \blacksquare
\end{aligned}$$

At this point, there are a hidden pattern that reveals itself after we study both cases ($n = 2$ and $n = 3$). The official proof of the concept starts from here.

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

Proof We recall the complete proof of this lemma from PatanasakPinyo and Sulaiman [14]. Note that some variables have been changed according to the context of the paper. Suppose there exists a permutation $\mathscr{C}'_i$ such that it belongs to A_k and $A_m, m \neq k$, at the same time. Since $\mathscr{C}'_i \in A_k$, there are k umbrellas that are not assigned to their owners, i.e., there are k numbers of (x_a, y_b) pairs that $a \neq b$ in $\mathscr{C}'_i$. Since $\mathscr{C}'_i \in A_m$, there are m umbrellas that are not assigned to 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, $k = m$ must hold. Thus there exists a contradiction. Hence, proved ■

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

Proof Again, we recall the formal proof of the theorem from PatanasakPinyo and Sulaiman [14]. This is a more formal version of the equation that we constructed previously when we discussed about the case of $n = 3$. For $n \in \mathbb{N}$:

$$\begin{aligned}
\mathbb{S}_n &= C_{(n,0)} + C_{(n,1)} + C_{(n,2)} + \cdots + C_{(n,n)} && \because \textbf{Lemma 1} \\
\mathbb{S}_n &= C_{(n,0)} + C_{(n,1)} + C_{(n,2)} + \cdots + C_{(n,n-1)} + C_{(n,n)} \\
\mathbb{S}_n &= \sum_{i=0}^{n-1} C_{(n,i)} + C_{(n,n)} \\
\mathbb{S}_n &= \sum_{i=0}^{n-1} C_{(n,i)} + !n \\
!n &= \mathbb{S}_n - \sum_{i=0}^{n-1} C_{(n,i)} && \blacksquare
\end{aligned}$$

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

Proof The proof of this lemma is very obvious since it is impossible, for any case of $n \in \mathbb{N}$, where there is only one person that is assigned a wrong umbrella. The total of people who are assigned the wrong umbrellas must be at least two. Hence, proved ■

We will use Theorem 2 and Lemma 3 to apply on the cases of $n = 2, 3, 4$, respectively.

1. Case $n = 2$

$$!2 = \mathbb{S}_2 - (C_{(2,0)} + C_{(2,1)}) \quad (1)$$
$$!2 = \mathbb{S}_2 - C_{(2,0)} \quad (2)$$
$$!2 = \mathbb{S}_2 - 1 \quad (3)$$

2. Case $n = 3$

$$!3 = \mathbb{S}_3 - (C_{(3,0)} + C_{(3,1)} + C_{(3,2)}) \quad (4)$$
$$!3 = \mathbb{S}_3 - (C_{(3,0)} + C_{(3,2)}) \quad (5)$$
$$!3 = \mathbb{S}_3 - (1 + C_{(3,2)}) \quad (6)$$

3. Case $n = 4$

$$!4 = \mathbb{S}_4 - (C_{(4,0)} + C_{(4,1)} + C_{(4,2)} + C_{(4,3)}) \quad (7)$$
$$!4 = \mathbb{S}_4 - (1 + C_{(4,2)} + C_{(4,3)}) \quad (8)$$

The case of $n = 2$ might cause zero problem since it can be directly computed (Line 3). However, the cases of $n = 3$ and $n = 4$ might do since there exist terms that need further computations, which are $C_{(3,2)}$ (Case $n = 3$, Line 6) and $C_{(4,2)} + C_{(4,3)}$ (Case $n = 4$, Line 8). These terms are leading us to a recursive property of the derangement problem. We begin unrolling $C_{(3,2)}$ from Line 6 first. By the definition, $C_{(3,2)}$ is equivalent to the scenario that there are two out of three people that are not assigned their umbrellas. The total ways of $C_{(3,2)}$ can be computed via two steps. We first compute $\binom{3}{1}$ to pick one person who correctly gets his umbrella. The second is to compute how many ways that the remaining two people are

assigned wrong umbrellas, which is exactly computing !2. Now we can see that the recursive call has popped up, i.e.,

$$!3 = \mathbb{S}_3 - (1 + C_{(3,2)}) \tag{9}$$

$$!3 = \mathbb{S}_3 - (1 + \binom{3}{1}!2) \tag{10}$$

We continue with the case of $n = 4$ in which we have to compute $C_{(4,2)} + C_{(4,3)}$. First, to achieve $C_{(4,2)}$, we compute $\binom{4}{2}$ to get total ways of picking 2 people who are correctly assigned the umbrellas. Next, we compute total ways of the remaining two-people derangement by recalling !2. Similarly for $C_{(4,3)}$, we compute $\binom{4}{1}$ for one person who correctly gets his umbrella and !3 for the remaining three-people derangement. Hence:

$$!4 = \mathbb{S}_4 - (1 + C_{(4,2)} + C_{(4,3)}) \tag{11}$$

$$!4 = \mathbb{S}_4 - (1 + \binom{4}{2}!2 + \binom{4}{1}!3) \tag{12}$$

To compute !4 (Line 12), the first term that we can easily compute is $\mathbb{S}_4$, which is nothing but 4!. The second term is $\binom{4}{2}!2$. This one is also computably easy since !2 is a basis case of derangement. The last term that we have to get it done is $\binom{4}{1}!3$. We can compute !3 by Line 10, which we have to do !2. This is the place that the DP gets involved since we have already computed !2 in the previous term. We just recall the value of !2 that we previously computed and stored it somewhere in the memory rather than re-computing the same expression. Searching for the value stored in an array of size $n \in \mathbb{N}$ has $O(n)$ running time, which is considered efficient.

We finally decide to demonstrate the complete process of this DP algorithm for derangement problem. We select the case of $n = 5$. The instance for this case includes $X = \{x_1, x_2, x_3, x_4, x_5\}$ and $Y = \{y_1, y_2, y_3, y_4, y_5\}$. Let $\mathbb{O}$ represent an array of size n where $\mathbb{O}[i] = !i, 1 \leq i \leq n$. Tables 3 - 9 illustrate all the steps happen during the run-time of the algorithm.

Table 3: Initial Step of Computing !5.

Process	$!5 = \mathbb{S}_5 - (C_{(5,0)} + C_{(5,1)} + C_{(5,2)} + C_{(5,3)} + C_{(5,4)})$
$\mathbb{O}$	$[0]$

At the initial step (Table 3), We recall Theorem 2 to construct an equation for !5. $\mathbb{O}$ is initiated with only one member, $\mathbb{O}[1] = 0$, which represents !1.

Table 4: Step 2 of Computing !5 (Compute $\mathbb{S}_5$).

Process	$!5 = 5! - (C_{(5,0)} + C_{(5,1)} + C_{(5,2)} + C_{(5,3)} + C_{(5,4)})$
$\mathbb{O}$	$[0]$

At Step 2 (Table 4), we solve $\mathbb{S}_5$ by $\mathbb{S}_5 = 5!$. There is no change for $\mathbb{O}$ at this step.

At Step 3 (Table 5), we replace $C_{(5,0)}$ by 1 since there is only one way that everyone correctly gets his umbrella.

Table 5: Step 3 of Computing !5 (Compute $C_{(5,0)}$).

Process	$!5 = 5! - (1 + C_{(5,1)} + C_{(5,2)} + C_{(5,3)} + C_{(5,4)})$
$\mathbb{O}$	$[0]$

Table 6: Step 4 of Computing !5 (Compute $C_{(5,1)}$).

Process	$!5 = 5! - (1 + 0 + C_{(5,2)} + C_{(5,3)} + C_{(5,4)})$
$\mathbb{O}$	$[0]$

Table 7: Step 5 of Computing !5 (Compute $C_{(5,2)}$).

Process	$!5 = 5! - (1 + 0 + \binom{5}{3}!2 + C_{(5,3)} + C_{(5,4)})$ $!5 = 5! - (1 + 0 + \binom{5}{3}(1) + C_{(5,3)} + C_{(5,4)})$
$\mathbb{O}$	$[0, 1]$

At Step 4 (Table 6), we recall Lemma 1 for $C_{(5,1)}$. Hence, $\mathbb{O}$ still has no update.

At Step 5 (Table 7), we would like to compute $C_{(5,2)}$. So, we replace it with $\binom{5}{3}!2$. Since $!2 = 1$ is the base case of derangement, we replace !2 with 1 in the equation and add 1 to $\mathbb{O}[2]$.

Table 8: Step 6 of Computing !5 (Compute $C_{(5,3)}$).

Process	$!5 = 5! - (1 + 0 + \binom{5}{3}(1) + \binom{5}{2}!3 + C_{(5,4)})$ $!5 = 5! - (1 + 0 + \binom{5}{3}(1) + \binom{5}{2}(2) + C_{(5,4)})$
Recursive Process	$!3 = \mathbb{S}_3 - (C_{(3,0)} + C_{(3,1)} + C_{(3,2)})$ $!3 = 3! - (1 + 0 + \binom{3}{1}!2)$ $!3 = 2$
$\mathbb{O}$	$[0, 1, 2]$

At Step 6 (Table 8), we would like to compute $C_{(5,3)}$. So, we replace it with $\binom{5}{2}!3$. Unfortunately, we have not had the value of !3 stored in $\mathbb{O}$. We have to recursively call Theorem 2 to handle !3. After we constructed an equation for !3 (Row 2 of Table 8), we, rather than re-computing, search on $\mathbb{O}$ the value of !2 that we previously computed, which is required to compute !3. We end up with $!3 = 2$. So the term $\binom{5}{2}!3$ is done. We also add $!3 = 2$ to $\mathbb{O}[3]$.

Table 9: Step 7 of Computing !5 (Compute $C_{(5,4)}$).

Process	$!5 = 5! - (1 + 0 + \binom{5}{3}(1) + \binom{5}{2}!3 + \binom{5}{1}!4)$ $!5 = 5! - (1 + 0 + \binom{5}{3}(1) + \binom{5}{2}(2) + \binom{5}{1}(9))$ ■
Recursive Process	$!4 = \mathbb{S}_4 - (C_{(4,0)} + C_{(4,1)} + C_{(4,2)} + C_{(4,3)})$ $!4 = 4! - (1 + 0 + \binom{4}{2}!2 + \binom{4}{1}!3)$ $!4 = 9$
$\mathbb{O}$	$[0, 1, 2, 9]$

At Step 7 (Table 9), we would like to compute $C_{(5,4)}$. So, we replace it with $\binom{5}{1}!4$. Unfortunately, we have not had the

value of !4 stored in $\mathbb{O}$. We have to recursively call Theorem 2 to handle !4. After we constructed an equation for !4 (Row 2 of Table 9), we, rather than re-computing, search on $\mathbb{O}$ both the values of !2 and !3 that we previously computed, which are required to compute !4. We end up with $!4 = 9$. So the term $\binom{5}{2}!3$ is done. We also add $!3 = 2$ to $\mathbb{O}[3]$. This is also the final step since we unlock every term required to compute !5. The algorithm halts and return the solution of !5. The complete pseudocode and the analysis part to verify the performance of the purposed DP algorithm will be discussed in the next section.

4 Dynamic Programming as Efficient Algorithm for Derangement

4.1 Design Phase

Algorithm 1 Dynamic Programming Algorithm for Derangement Problem.

```
1: function TOTAL-DERANGEMENT(n)
2:     return REC-DERANGE(n, n)
3: end function
```

Algorithm 2 Algorithm for Subproblem of Derangement.

```
1: function REC-DERANGE(n, i)
2:     𝕆 ← [0]
3:     if i = 0 then
4:         return 1
5:     end if
6:     if i = 1 then                                   ▷ Lemma 3
7:         return 0
8:     end if
9:     if 1 < i ≤ n − 1 then
10:        d_i ← 0
11:        if 𝕆[i] ≠ null then
12:            d_i ← 𝕆[i]
13:        else
14:            d_i ← REC-DERANGE(i, i)
15:            𝕆[i] ← d_i
16:        end if
17:        return C(n, n−i) d_i
18:    end if
19:    if i = n then
20:        sum ← 0
21:        for j = 0 : n − 1 do
22:            sum ← sum + REC-DERANGE(n, j)
23:        end for
24:        return n! − sum                  ▷ !n = $_n − Σ_{i=0}^{n−1} C_(n,i)
25:    end if
26: end function
```

We officially present two algorithms [14]. Algorithm 1 is the main procedure that retrieves an input $n \in \mathbb{N}$ and output $!n$. Algorithm 2 is a sub-procedure, which implementing DP approach previously described in the last section, called by Algorithm 1. The procedure that Algorithm 2 executes is totally based on Lemma 1, Theorem 2, and Lemma 3 as we mark in-line of the pseudocode. Furthermore for Algorithm 2, we added a modification to the previous version of PatansakPinyo and Sulaiman [14] on Lines 2, 10, 11, 12, 13, 14, 15, and 16 to fulfill the usage of Array $\mathbb{O}$, which makes it fits the concept of the DP algorithm.

4.2 Analysis Phase

The former analysis done by PatanasakPinyo and Sulaiman [14] is still valid even though we add more instructions to the pseudocode in Algorithm 2. Particularly, Algorithm 2 consists of two parts that affect the running time. The first part os located on Lines 14 and 22. This part involves with a recursive call. The upper bound of the recursive call is at most $n-1$ calls. Another part that causes a significant running time is an iteration on Line 21. Again, the upper bound of the iteration is $O(n)$. Combining these two parts together, we come up with the running time of $O(n^2)$, which is a polynomial running time. For Algorithm 1, the pseudocode has no change at all, so there is no effect on the running time. Hence, the running time of the proposed DP algorithm is dominated by Algorithm 2. We conclude that the running time is $O(n)$, which is technically considered efficient as it has a polynomial bound [5].

5 Generalization of Derangement Problems

5.1 Hat-check Problem: A Popular Special Case of Derangement Problems and the Way to Generalize It

The classic derangement is a special case that there exists the efficient algorithm (we also proposed one in this paper). However, there is no guarantee that the algorithm still works precisely when we generalize the problem. In this paper, we use the umbrella problem to depict the derangement problem. In mathematics and probability theory, this same approach is known by the name of the hat-check problem. An instance of the hat-check problem is very similar to the umbrella problem where there are two sets: X and Y. $X = \{x_1, x_2, ..., x_n\}$ is a set of $n \in \mathbb{N}$ customers. $Y = \{y_1, y_2, ..., y_n\}$ is a set of n hats where Hat y_i belongs to Customer x_i for all $1 \leq i \leq n$. We say that both the hat-check problem or the umbrella problem are a special case of derangement problem where each customer has possessed only one item (either a hat or an umbrella). Hence, we generalize the derangement problem that allows each customer to possess $k \in \mathbb{N}$ items. Readers can think of the case there are $n \in \mathbb{N}$ customers where each customer has $k \in \mathbb{N}$ items. The instance of the generalized version of the hat-check problem would include the following $(k+1)$ sets:

1. $X = \{x_1, x_2, ..., x_n\}$ where $x_i, 1 \leq i \leq n$, represents Customer i.
2. $Y_1 = \{y_{[1,1]}, y_{[1,2]}, ..., y_{[1,n]}\}$ where $y_{[1,i]}$ represents the first item of Customer i. We can assume that the first item is a

hat. Hence, Y_1 is a set of n hats.

3. $Y_2 = \{y_{[2,1]}, y_{[2,2]}, ..., y_{[2,n]}\}$ where $y_{[2,i]}$ represents the second item of Customer i. We can assume that the second item is an umbrella. Hence, Y_2 is a set of n umbrellas.

....
...
..

k. $Y_{k-1} = \{y_{[k-1,1]}, y_{[k-1,2]}, ..., y_{[k-1,n]}\}$ where $y_{[k-1,i]}$ represents the $(k-1)^{\text{th}}$ item of Customer i. We can assume that the $(k-1)^{\text{th}}$ item is a pair of sunglasses. Hence, Y_{k-1} is a set of n pairs of sunglasses.

$(k+1)$. $Y_k = \{y_{[k,1]}, y_{[k,2]}, ..., y_{[k,n]}\}$ where $y_{[k,i]}$ represents the k^{th} item of Customer i. We can assume that the k^{th} item is a mobile phone. Hence, Y_k is a set of n mobile phones

With this instance of generalized version, it is obvious that all the existing methodologies for classical derangement problem might not function correctly as we expect. Moreover, the previous definition of derangement does not fit with this version of the problem since the value of k is not necessary to be $k = 1$. We then categorize a derangement into two types: **complete derangement** and **partial derangement**.

Definition 4 Let an instance of the hat-check problem be $n \in \mathbb{N}$ people and $k \in \mathbb{N}$ items each. An arrangement $\mathscr{C}'$ is called **complete derangement** if for every ordered tuple $(x_i, y_{[1,j_1]}, ..., y_{[k,j_k]}) \in \mathscr{C}', \bigwedge_{\lambda=1}^{k} i \neq j_\lambda$.

Definition 5 Let an instance of the hat-check problem be $n \in \mathbb{N}$ people and $k \in \mathbb{N}$ items each. An arrangement $\mathscr{C}'$ is called **partial derangement** if for every ordered tuple $(x_i, y_{[1,j_1]}, ..., y_{[k,j_k]}) \in \mathscr{C}', \bigvee_{\lambda=1}^{k} i \neq j_\lambda$.

Definition 6 Let an instance of the hat-check problem be $n \in \mathbb{N}$ people and $k \in \mathbb{N}$ items each. An arrangement $\mathscr{C}'$ is called **non-derangement** if there exists an ordered tuple $(x_i, y_{[1,j_1]}, ..., y_{[k,j_k]}) \in \mathscr{C}', \bigwedge_{\lambda=1}^{k} i = j_\lambda$.

Definition 7 Let an instance of the hat-check problem be $n \in \mathbb{N}$ people and $k \in \mathbb{N}$ items each. An arrangement $\mathscr{C}'$ is called **antiderangement** if for every ordered tuple $(x_i, y_{[1,j_1]}, ..., y_{[k,j_k]}) \in \mathscr{C}', \bigwedge_{\lambda=1}^{k} i = j_\lambda$.

We use the following scenario to explain the difference between those two types of derangement. Let the instance be $n = 2$ customers and $k = 2$ items for each customer. To reduce ambiguity, we use Y rather than Y_1 and Z rather than Y_2. Particularly:

$$X = \{x_1, x_2\}$$
$$Y = \{y_1, y_2\}$$
$$Z = \{z_1, z_2\}$$

Initially, every customer comes with his own items, so the input collection would be $\mathscr{C}_{\text{in}} = \{(x_1, y_1, z_1), (x_2, y_2, z_2)\}$. The total arrangements, which is in a form of a collection of ordered triples (x_i, y_i, z_i), that can happen are:

$$\mathscr{C}'_1 = \{(x_1, y_1, z_1), (x_2, y_2, z_2)\}$$
$$\mathscr{C}'_2 = \{(x_1, y_1, z_2), (x_2, y_2, z_1)\}$$
$$\mathscr{C}'_3 = \{(x_1, y_2, z_1), (x_2, y_1, z_2)\}$$
$$\mathscr{C}'_4 = \{(x_1, y_2, z_2), (x_2, y_1, z_1)\}$$

From the list of outcomes above, $\mathscr{C}'_1$ is both non-derangement and antiderangement since every customer completely gets his own items. $\mathscr{C}'_2$ and $\mathscr{C}'_3$ are partial derangement since every customer gets at least one item that belongs to the other customers. Lastly, $\mathscr{C}'_4$ is the only complete derangement where every customer does not get his own items. The total number of arrangements is not computably difficult. Given n people and k items, the total number of arrangements, denoted by $\mathbb{S}(n,k)$, can be computed by:

$$\mathbb{S}(n,k) = (n!)^k$$

So for the example case ($n = k = 2$), we then get $\mathbb{S}(2,2) = (2!)^2 = 4$.

The next example that we would like to demonstrate is the case that $n = 2$ and $k = 3$. Again, we use Y rather than Y_1 and Z rather than Y_2 and W rather than Y_3. Hence, the total number of arrangements is $\mathbb{S}(2,3) = (2!)^3 = 8$. The list of those eight arrangements is:

$$\mathscr{C}'_1 = \{(x_1, y_1, z_1, w_1), (x_2, y_2, z_2, w_2)\}$$
$$\mathscr{C}'_2 = \{(x_1, y_1, z_1, w_2), (x_2, y_2, z_2, w_1)\}$$
$$\mathscr{C}'_3 = \{(x_1, y_1, z_2, w_1), (x_2, y_2, z_1, w_2)\}$$
$$\mathscr{C}'_4 = \{(x_1, y_1, z_2, w_2), (x_2, y_2, z_1, w_1)\}$$
$$\mathscr{C}'_5 = \{(x_1, y_2, z_1, w_1), (x_2, y_1, z_2, w_2)\}$$
$$\mathscr{C}'_6 = \{(x_1, y_2, z_1, w_2), (x_2, y_1, z_2, w_1)\}$$
$$\mathscr{C}'_7 = \{(x_1, y_2, z_2, w_1), (x_2, y_1, z_1, w_2)\}$$
$$\mathscr{C}'_8 = \{(x_1, y_2, z_2, w_2), (x_2, y_1, z_1, w_1)\}$$

Clearly from the list, $\mathscr{C}'_1$ is antiderangement. $\mathscr{C}'_8$ is complete derangement. $\mathscr{C}'_2$ to $\mathscr{C}'_7$ are partial derangement. Observe that for $n = 2$, the arrangement that is non-derangement but not antiderangement does not exist.

To show readers every type that we have a definition for, we demonstrate the last example with the case that $n = 3$ and $k = 2$. Again, we use Y rather than Y_1 and Z rather than Y_2. Hence, the total number of arrangements is $\mathbb{S}(3,2) = (3!)^2 = 36$. The list

of those eight arrangements is:

$$\mathscr{C}'_1 = \{(x_1,y_1,z_1),(x_2,y_2,z_2),(x_3,y_3,z_3)\} \quad \clubsuit$$
$$\mathscr{C}'_2 = \{(x_1,y_1,z_1),(x_2,y_2,z_3),(x_3,y_3,z_2)\} \quad \blacklozenge$$
$$\mathscr{C}'_3 = \{(x_1,y_1,z_1),(x_2,y_3,z_2),(x_3,y_2,z_3)\} \quad \blacklozenge$$
$$\mathscr{C}'_4 = \{(x_1,y_1,z_1),(x_2,y_3,z_3),(x_3,y_2,z_2)\} \quad \blacklozenge$$

$$\mathscr{C}'_5 = \{(x_1,y_1,z_2),(x_2,y_2,z_1),(x_3,y_3,z_3)\} \quad \blacklozenge$$
$$\mathscr{C}'_6 = \{(x_1,y_1,z_2),(x_2,y_2,z_3),(x_3,y_3,z_1)\} \quad \heartsuit$$
$$\mathscr{C}'_7 = \{(x_1,y_1,z_2),(x_2,y_3,z_1),(x_3,y_2,z_3)\} \quad \heartsuit$$
$$\mathscr{C}'_8 = \{(x_1,y_1,z_2),(x_2,y_3,z_3),(x_3,y_2,z_1)\} \quad \heartsuit$$

$$\mathscr{C}'_9 = \{(x_1,y_1,z_3),(x_2,y_2,z_1),(x_3,y_3,z_2)\} \quad \heartsuit$$
$$\mathscr{C}'_{10} = \{(x_1,y_1,z_3),(x_2,y_2,z_2),(x_3,y_3,z_1)\} \quad \blacklozenge$$
$$\mathscr{C}'_{11} = \{(x_1,y_1,z_3),(x_2,y_3,z_1),(x_3,y_2,z_2)\} \quad \heartsuit$$
$$\mathscr{C}'_{12} = \{(x_1,y_1,z_3),(x_2,y_3,z_2),(x_3,y_2,z_1)\} \quad \heartsuit$$

$$\mathscr{C}'_{13} = \{(x_1,y_2,z_1),(x_2,y_1,z_2),(x_3,y_3,z_3)\} \quad \blacklozenge$$
$$\mathscr{C}'_{14} = \{(x_1,y_2,z_1),(x_2,y_1,z_3),(x_3,y_3,z_2)\} \quad \heartsuit$$
$$\mathscr{C}'_{15} = \{(x_1,y_2,z_1),(x_2,y_3,z_2),(x_3,y_1,z_3)\} \quad \heartsuit$$
$$\mathscr{C}'_{16} = \{(x_1,y_2,z_1),(x_2,y_3,z_3),(x_3,y_1,z_2)\} \quad \heartsuit$$

$$\mathscr{C}'_{17} = \{(x_1,y_2,z_2),(x_2,y_1,z_1),(x_3,y_3,z_3)\} \quad \blacklozenge$$
$$\mathscr{C}'_{18} = \{(x_1,y_2,z_2),(x_2,y_1,z_3),(x_3,y_3,z_1)\} \quad \heartsuit$$
$$\mathscr{C}'_{19} = \{(x_1,y_2,z_2),(x_2,y_3,z_1),(x_3,y_1,z_3)\} \quad \heartsuit$$
$$\mathscr{C}'_{20} = \{(x_1,y_2,z_2),(x_2,y_3,z_3),(x_3,y_1,z_1)\} \quad \spadesuit$$

$$\mathscr{C}'_{21} = \{(x_1,y_2,z_3),(x_2,y_1,z_1),(x_3,y_3,z_2)\} \quad \heartsuit$$
$$\mathscr{C}'_{22} = \{(x_1,y_2,z_3),(x_2,y_1,z_2),(x_3,y_3,z_1)\} \quad \heartsuit$$
$$\mathscr{C}'_{23} = \{(x_1,y_2,z_3),(x_2,y_3,z_1),(x_3,y_1,z_2)\} \quad \spadesuit$$
$$\mathscr{C}'_{24} = \{(x_1,y_2,z_3),(x_2,y_3,z_2),(x_3,y_1,z_1)\} \quad \heartsuit$$

$$\mathscr{C}'_{25} = \{(x_1,y_3,z_1),(x_2,y_1,z_2),(x_3,y_2,z_3)\} \quad \heartsuit$$
$$\mathscr{C}'_{26} = \{(x_1,y_3,z_1),(x_2,y_1,z_3),(x_3,y_2,z_2)\} \quad \heartsuit$$
$$\mathscr{C}'_{27} = \{(x_1,y_3,z_1),(x_2,y_2,z_2),(x_3,y_1,z_3)\} \quad \blacklozenge$$
$$\mathscr{C}'_{28} = \{(x_1,y_3,z_1),(x_2,y_2,z_3),(x_3,y_1,z_2)\} \quad \heartsuit$$

$$\mathscr{C}'_{29} = \{(x_1,y_3,z_2),(x_2,y_1,z_1),(x_3,y_2,z_3)\} \quad \heartsuit$$
$$\mathscr{C}'_{30} = \{(x_1,y_3,z_2),(x_2,y_1,z_3),(x_3,y_2,z_1)\} \quad \spadesuit$$
$$\mathscr{C}'_{31} = \{(x_1,y_3,z_2),(x_2,y_2,z_1),(x_3,y_1,z_3)\} \quad \heartsuit$$
$$\mathscr{C}'_{32} = \{(x_1,y_3,z_2),(x_2,y_2,z_3),(x_3,y_1,z_1)\} \quad \heartsuit$$

$$\mathscr{C}'_{33} = \{(x_1,y_3,z_3),(x_2,y_1,z_1),(x_3,y_2,z_2)\} \quad \spadesuit$$
$$\mathscr{C}'_{34} = \{(x_1,y_3,z_3),(x_2,y_1,z_2),(x_3,y_2,z_1)\} \quad \heartsuit$$
$$\mathscr{C}'_{35} = \{(x_1,y_3,z_3),(x_2,y_2,z_1),(x_3,y_1,z_2)\} \quad \heartsuit$$
$$\mathscr{C}'_{36} = \{(x_1,y_3,z_3),(x_2,y_2,z_2),(x_3,y_1,z_1)\} \quad \blacklozenge$$

From the provided list, an arrangement marked with $\clubsuit$ is antiderangement. An arrangement marked with $\blacklozenge$ is non-derangement. An arrangement marked with $\heartsuit$ is partial derangement. Lastly, an arrangement marked with $\spadesuit$ is complete derangement. For the case of $n = 3$ and $k = 2$, only one arrangement ($\mathscr{C}'_1$) is antiderangement. There are nine arrangements that are non-derangement but not antiderangement ($\mathscr{C}'_2, \mathscr{C}'_3, \mathscr{C}'_4, \mathscr{C}'_5, \mathscr{C}'_{10}, \mathscr{C}'_{13}, \mathscr{C}'_{17}, \mathscr{C}'_{27}$ and $\mathscr{C}'_{36}$). There are four arrangements that are complete derangement ($\mathscr{C}'_{20}, \mathscr{C}'_{23}, \mathscr{C}'_{30}$ and $\mathscr{C}'_{33}$). The remaining twenty-two arrangements are partial derangement but not complete derangement.

Currently, as of our best knowledge, there exists no DP algorithm that is capable of tackling the generalized version of the hat-check problem where an instance of the problem contains $n \in \mathbb{N}$ customers and $k \in \mathbb{N}$ items.

6 Conclusions and Possible Future Direction

We report an algorithm that computes the total number of arrangements that are derangement with an input $n \in \mathbb{N}$ by using a concept of DP, which known to be efficient for several applications. An instance of the derangement problem comes with inputs $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$. The problem inquires how many $\mathscr{C}' = \{(x_i, y_j) \in X \times Y\}$ such that $i \neq j$ and every $x_i \in X$ shows up exactly once as well as every $y_j \in Y$. We invoke a methodology to compute from PatanasakPinyo and Sulaiman [14], which has an objective to extract a recursive call to solve a subproblem with smaller instance. The output of the proposed algorithm is correct and the algorithm itself has a polynomial running time, $O(n^2)$, that proves its efficiency.

We initiate an idea to generalize the hat-check problem by allowing the instance of the problem to be $n \in \mathbb{N}$ customers and $k \in \mathbb{N}$ items rather than $k = 1$, which is a default setup of the classic instance of the problem. A door is opened for theoretical computer scientists to come up with an efficient algorithm that overcomes this generalized version.

Furthermore, the derangement problem is potentially a key to several applications that need to rearrange a finite set such that the input arrangement should not to be returned. The example is designing a sequence of addresses for the address verification task [1, 6, 8, 9]. Another good example is designing a matching problems for the paper folding test, which is a test of individual spatial visualization (VZ) [10, 11, 12, 13, 16].

Acknowledgments

This study is partially supported by the Faculty of Information and Communication Technology, Mahidol University, Thailand.

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] Pallavi Joshi, Chang-Seo Park, Koushik Sen, and Mayur Naik. A randomized dynamic program analysis technique for detecting real deadlocks. *ACM Sigplan Notices*, 44(6):110–120, 2009.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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).

[11] 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.

[12] 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.

[13] 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.

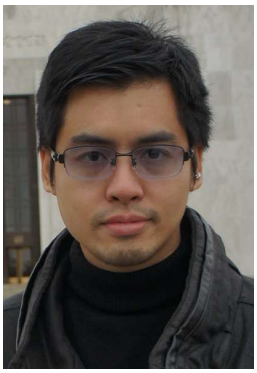
[14] Thitivatr Patanasakpinyo and Adel Sulaiman. Alternative approach to achieve a solution of derangement problems by dynamic programming. In Ajay Bandi, Mohammad Hossain, and Ying Jin, editors, *Proceedings of 38th International Conference on Computers and Their Applications*, volume 91 of *EPiC Series in Computing*, pages 98–107. EasyChair, 2023.

[15] 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.

[16] 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.

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

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



Thitivatr PatanasakPinyo is a Teaching Professor at the Faculty of Information and Communication Technology, Mahidol University. He received his PhD in Computer Science from Iowa State University of Science and Technology, Iowa, USA under a supervision of Prof. Dr. Les Miller and Prof. Dr. Pak Tavanapong. His dissertation is "Flattening methods for adaptive location-based software to user abilities". His research area consists of HCI theory, algorithm, and theoretical computer science.

Adel Sulaiman is an Assistant Professor at the College of Computer Science and Information Systems, Najran University. He received his PhD in Computer Science from Iowa State University of Science and Technology, Iowa, USA under a supervision of Prof. Dr. Les Miller and Assoc. Prof. Dr. Stephen Gilbert. His research area consists of HCI and usability.