The Most and Longest Increasing Interval Subsequence Problems

Guan-Ting Chen and Chang-Biau Yang

***Abstract*—This paper focuses on the interval sequences and design efficient algorithms for solving the *longest increasing subsequence* (LIS) on the interval sequences. We propose three ways for comparing two intervals: single value, full interval, and subinterval. We combine the interval comparison with the LIS problem to define two variants of the problem. The most *increasing interval subsequence* (MIIS) problem aims to find the maximum number of intervals involved in the answer. The *longest increasing interval subsequence* (LIIS) problem focuses on the total length of the intervals involved in the answer. We give a new research direction for interval sequences and design the algorithms to solve them in $O(n^2)$ time, where n denotes the number of elements in the input interval sequence.**

***Index Terms*—interval, interval sequence, longest increasing subsequence, longest increasing interval subsequence, most increasing interval subsequence**

I. Introduction

Given a numeric sequence, the *longest increasing subsequence problem* (LIS) aims to find the strictly increasing subsequence with the maximal length. For example, suppose that $A = \langle 6, 5, 7, 3, 9, 4\rangle$. Then, the LIS length is 3, and $\langle 5, 7, 9\rangle$ is one of the LIS answers. The LIS problem has been extensively studied over the past decades and it is widely applicable in various fields. In 1961, Schensted [14] first introduced the concept of the LIS problem and proposed an algorithm to solve it in $O(n \log n)$ time.

The previous researches on the LIS problem are listed in Table I. Besides the traditional LIS problem, some variants of the LIS problem have also been proposed [2, 4, 7, 8, 10–13, 15, 16, 19, 20]. The previous researches focused on numeric sequences, but the real-world data often consist of intervals, rather than individual values, such as daily temperatures and stock prices (low and high). Due to the interval sequences (examples including temperatures and stock prices) in real life, this paper focuses on the study of interval sequences, where each element in the sequence is an interval, represented by a starting (low) value and an ending (high) value.

The comparison of two intervals is more complex than the comparison of two single values. Before studying the related topics of an increasing or decreasing interval sequence, we need to give a proper definition of the comparison of two intervals. To compare two intervals, we propose three ways as follows.

Guan-Ting Chen is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan.

Chang-Biau Yang is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan. E-mail: cbyang@cse.nsysu.edu.tw. (Corresponding author)

(1) Single value: If there exists one value in one interval that is greater than any one value in another interval, then we say that the former interval is greater than the latter.

(2) Full interval: If the full range of one interval is greater than the full range of another interval, then we say that the former interval is greater than the latter.

(3) Subinterval: If there exists a subinterval of one interval that is greater than a subinterval of another interval, then we say that the former interval is greater than the latter.

The meaning of the length of an increasing interval subsequence is different from the traditional LIS. In this paper, we present two ways to define the length. Like the LIS, we can regard an interval as an element and aim to find as many intervals as possible. To distinguish between two different concepts, we call this the *most increasing interval subsequence* (MIIS) problem. On the other hand, the *longest increasing interval subsequence* (LIIS) considers the length of each interval and tries to find the increasing interval subsequence with the maximal sum of the lengths of all intervals contained in the subsequence.

By incorporating the comparison of two intervals and the definition of the increasing interval subsequence's length, we outline the problems as follows.

- *most increasing interval subsequence* (MIIS)
  1) single value
  2) full interval
  3) subinterval
- *longest increasing interval subsequence* (LIIS)
  1) full interval
  2) subinterval

The rest of the paper is structured as follows. Section II and Section III provide the definitions and algorithms of the MIIS problem and the LIIS problem, respectively. The conclusion is presented in Section IV.

II. The Most Increasing Interval Subsequence

A. The Problem Definition

We first give the definition of an interval as follows.

Definition 1. (interval) *An interval $x = [x_s, x_e]$ is an integer range between a starting point x_s and an ending point x_e, $x_s \le x_e$. The interval includes x_s and x_e, and the interval length $|x| = x_e - x_s + 1$.*

Before handling the increasing subsequence, we need the definition of the comparison of two intervals.

TABLE I: The previous researches on the LIS problem and its related problems, where n denotes the length of the input sequence.

Year	Author(s)	Time complexity	Note
1961	Schensted [14]	$O(n \log n)$	Young tableau, binary search
1977	Hunt and Szymanski [9]	$O(n \log \log n)$	Match Pair, van Emde Boas tree
2000	Bespamyatnikh and Segal [3]	$O(n \log \log n)$	Enumerating all answers
2010	Crochemore and Porat [6]	$O(n \log \log l)$	Split blocks
2013	Alam and Rahman [1]	$O(n \log n)$	Divide-and-conquer

Definition 2. (comparison by a single value) *Given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, if there exist an integer $a \in x$ and an integer $b \in y$ such that $a < b$, then we say that $x < y$.*

In Definition 2, if two intervals overlap, the comparison result is not unique. For example, given two intervals $x = [3, 5]$ and $y = [4, 6]$, we get $a = 3 \in x$ and $b = 4 \in y$ that $a < b$, so $x < y$. Meanwhile, if we get $a = 5 \in x$ and $b = 4 \in y$, the result will become $y < x$ because $b < a$. The definition may be complicate, but it makes the related problems more flexible in different situations.

Definition 3. (comparison by full interval) *Given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, if $x_e < y_s$, then we say that $x < y$.*

In Definition 3, we can get the comparison result only if the two compared intervals do not overlap. If the two intervals overlap, the result is defined to be undetermined. The interval increasing subsequence under Definition 3 can be regarded as a strictly increasing interval sequence. Then, the algorithm designed for solving the related problem based on Definition 3 is also similar to the traditional LIS problem.

However, the comparison by full interval is overly rigid. Therefore, we propose a third definition that combines the comparison by a single value and full interval.

Definition 4. (subinterval) *A subinterval $x' = [x'_s, x'_e]$ of an interval $x = [x_s, x_e]$, denoted by $x' \subseteq x$, is a continuous range that $x_s \leq x'_s \leq x'_e \leq x_e$.*

For instance, $[3, 4]$, $[3, 5]$ and $[6, 9]$ are some subintervals of an interval $x = [3, 9]$.

Definition 5. (comparison by a subinterval) *Given two subintervals $x' = [x'_s, x'_e] \subseteq x$, $y' = [y'_s, y'_e] \subseteq y$ and a predefined constant c, we say that $x < y$ if $x'_e < y'_s$ and $|x'| = |y'| = c$.*

In Definition 5, we can adjust c to bring the results closer to the requirements. Similar to Definition 2, if the overlapping length between two intervals is larger than or equal to $2 \times c$, the comparison result will be not unique.

Definition 6. (MIIS problem) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$, the* most increasing interval subsequence (*MIIS*) *problem is to find the increasing interval subsequence with the maximum number of intervals.*

Combined with the three types of interval comparison, we have three versions of MIIS problems. We will design algorithms for solving these problems in the following subsections.

TABLE II: An example of MIIS with comparison by a single value, where $X = \langle [42, 50], [20, 27], [77, 81], [28, 36], [25, 46], [19, 45] \rangle$. The answer is $\langle [20, 27], [28, 36], [25, 46], [19, 45] \rangle$ with 4 intervals.

X \ q(quantity)		1	2	3	4
x_1	[42, 50]	**42**			
x_2	[20, 27]	**20**			
x_3	[77, 81]	20	**77**		
x_4	[28, 36]	20	**28**		
x_5	[25, 46]	20	**25**	**29**	
x_6	[19, 45]	**19**	**21**	**26**	**30**

B. The Algorithm for Comparison by a Single Value

Suppose we are given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$. In this problem, the interval comparison is based on Definition 2. An interval x_i can be a member of an increasing interval subsequence if a single value a can be found as a representative of x_i and satisfies the strictly increasing property. Therefore, the main concept is that a should be as small as possible in the strictly increasing case to provide the best extension in the future. In addition, there may be more than one value in x_i that fulfills strict increment, so all possible answers found previously should be extended or replaced.

We illustrate our algorithm with an example shown in Table II. Each round works as follows. In round 1, $x_1 = [42, 50]$, the ideal minimum value is 42 in the interval to ensure that the future is extendable enough to put more intervals. In round 2, $x_2 = [20, 27]$. Since $x_{e_2} = 27 < x_{s_1} = 42$, we cannot extend the subsequence. However, the minimal value $x_{s_2} = 20$ can replace 42 to get more potential for future extension. In round 3, $x_3 = [77, 81]$ is larger than 20, so we append the minimum value 77 to the answer. In round 4, $x_4 = [28, 36]$ is not large enough to extend the answer, but we can replace 77 with 28.

In round 5, $x_5 = [25, 46]$, we cannot do the same work as the traditional LIS algorithm. We have to consider all possible extensions. $x_{s_5} = 25 < 28$, so we replace 28 with 25 for more future potential. Besides, the last value in the previous answer is 28, which can be extended by appending 29, the minimally extendable value in x_5. Lastly, in round 6, all values

in the previous answer are contained by $x_6 = [19, 45]$, so all values have to be replaced. We replace the first value (20) with 19. We replace 21 as the second one because 21 is the minimally extendable value of 20. Similarly, we replace 26 as the third one. Finally, we extend the answer with 30 based on 29. Therefore we get the increasing interval subsequence $\langle[20, 27],\ [28, 36],\ [25, 46],[19, 45]\rangle$ with 4 intervals.

In Algorithm 1, we present the pseudo code of our algorithm for solving the MIIS problem with comparison by a single value. T is a tree structure to store the sorted answer integer list, and t is one element in T. The functions of T are described as follows. $T.append$ appends the element to the last position. $T.begin$ and $T.end$ mean the first and the last elements, respectively. $T.successor(e)$ returns the smallest element, which is larger than e; and if a successor does not exist, it returns $T.end$. $t.previous$ returns the previous element of the current t.

Algorithm 1 The algorithm for MIIS with comparison by a single value.

Input: An interval sequence $X = \langle x_1,\ x_2,\ x_3,\ \ldots,\ x_n\rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$.
Output: Length of the MIIS with comparison by a single value.

```
T.append(x_{s_1})
for i ← 2 to n do
    for t ← T.successor(x_{e_i}) to T.begin do
        if t < x_{s_i} then
            break
        if t.previous exists then
            t ← max(x_{s_i}, t.previous + 1)
        else
            t ← x_{s_i}
    if T.end < x_{e_i} then
        T.append(max(x_{s_i}, T.end + 1))
return T.size
```

In our algorithm, we ensure that all existing answers are the most potential. This results in the time complexity of O(n^2) and O(q) space, where q is the number of intervals in the answer.

C. The Algorithm for Comparison by Full Interval

If the comparison is made by two full intervals, we can choose the ending value of each interval as its representative. A smaller ending value will make the answer more extendable if we have the same number of intervals. Therefore, we aim to minimize the ending value of the interval subsequence while ensuring that we have the correct number of intervals. Table III shows an example to explain our algorithm

In round 1, $x_1 = [42, 50]$, we insert $x_{e_1} = 50$ into the answer. In the MIIS problem with Definition 3, since the intervals cannot overlap, we use the ending value of each interval as the representative when we insert it into the answer. With this way, we can directly use the starting value of a new coming interval to check if there is any overlap with the previous intervals. In round 2, $x_2 = [20, 27]$. This ending value $x_{e_2} = 27$ is smaller than 50. Therefore, we replace 50 by $x_{e_2} = 27$. In round 3, $x_3 = [77, 81]$. Since $x_{s_3} = 77 > x_{e_2} = 27$, we extend the answer length by appending $x_{e_3} = 81$. In the next round, $x_{s_4} = 28$ is larger than 27 but smaller than 81, so we replace 81 by 36 for a higher potential.

TABLE III: An example of MIIS with comparison by full interval, where $X = \langle[42, 50],\ [20, 27],\ [77, 81],\ [28, 36],\ [25, 33],\ [40, 45]\rangle$. The answer is $\langle[20, 27],\ [28, 36],\ [40, 45]\rangle$ with 3 intervals.

X \ q(quantity)		1	2	3
x_1	[42, 50]	**50**		
x_2	[20, 27]	**27**		
x_3	[77, 81]	27	**81**	
x_4	[28, 36]	27	**36**	
x_5	[25, 33]	27	36	
x_6	[40, 45]	27	36	**45**

TABLE IV: An example of MIIS with comparison by a subinterval, where $c = 3$ and $X = \langle[42, 50],\ [20, 27],\ [77, 81],\ [28, 36],\ [25, 40]\rangle$. The answer is $\langle[20, 27],\ [28, 36],\ [25, 40]\rangle$ with 3 intervals.

X \ q(quantity)		1	2	3
x_1	[42, 50]	**44**		
x_2	[20, 27]	**22**		
x_3	[77, 81]	22	**79**	
x_4	[28, 36]	22	**30**	
x_5	[25, 40]	22	**27**	**33**

The most special case in this problem is in round 5, $x_5 = [25, 33]$. x_5 overlaps the two intervals in the answer. If we choose x_5 as a member of the answer, then the number of intervals will be reduced to 1. Therefore, we abandon x_5. In the last round, like x_3, we can extend the answer length to 3 by appending 45 to the answer. Finally, we get the answer $\langle[20, 27],\ [28, 36],\ [40, 45]\rangle$ with 3 intervals.

Here, we omit the pseudo code of our algorithm. Since each new coming interval can be inserted at zero or one position, and the position can be decided by the successor operation, our algorithm requires O($n \log n$) time and O(q) space.

D. The Algorithm for Comparison by a Subinterval

This problem type is based on Definition 5. Two conditions need to be satisfied. Since the subinterval is a part of an interval, we have to ensure that all the elements in the answer are the most potential. At the same time, the subintervals cannot overlap with each other.

We illustrate our algorithm by the example shown in Table IV, where the subinterval length is set as $c = 3$. In round 1, we insert the first interval $x_1 = [42, 50]$ into the answer. To keep the more extendable, we hope the subinterval to end at the minimal value, so the subinterval should start at the minimal

value. Therefore, the first subinterval is $[42, 44]$. We use the maximal value 44 of the subinterval to represent it. In round 2, since $x_2 = [20, 27]$ is not larger than $[42, 44]$, we try to replace the latter. We get a more potential subinterval $x'_2 = [20, 22]$ to replace $x'_1 = [42, 44]$. In the next round, since $x_3 = [77, 81]$ and x_{s_3} is larger than $[20, 22]$, we can extend the answer by appending the subinterval $x'_3 = [77, 79]$. In round 4, since $x_4 = [28, 36]$ cannot be extended but it can replace $x'_3 = [77, 79]$ with the minimal subinterval $x'_4 = [28, 30]$. In the last round, $x_5 = [25, 40]$, two operations are performed. First, we use the minimal non-overlapping subinterval $x'_5 = [31, 33]$ to extend. Second, we have to check whether the subinterval is contained by $x_5 = [25, 40]$ in the answer and whether it may become better. We can replace $x'_4 = [28, 30]$ with $x'_5 = [25, 27]$. Finally, we get the answer $\langle[20, 27], [28, 36], [25, 40]\rangle$ with 3 intervals.

Our algorithm for solving the MIIS with comparison by a subinterval requires $O(n^2)$ time and $O(q)$ space. The pseudo code of our algorithm is omitted here.

III. The Longest Increasing Interval Subsequence

A. The Problem Definition

Here, we consider the length of each interval involved in the answer.

Definition 7. (length of an interval sequence) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n\rangle$ where $x_i = [x_{s_i}, x_{e_i}]$, $1 \le i \le n$, the length of the interval sequence is $|X| = |x_1| + |x_2| + |x_3| + \cdots + |x_n|$.*

Definition 8. (LIIS problem) *Given an interval sequence $X = \langle x_1, x_2, x_3, \ldots, x_n\rangle$, the* longest increasing interval subsequence (*LIIS*) *problem is to find the increasing interval subsequence whose total length is maximal.*

Now, we focus on the total length of the intervals, not the number of intervals. In other words, the answer should contains as large range as possible. Due to this concept, comparison by a single value is meaningless, and the definition of comparison by a subinterval is modified as follows.

Definition 9. (comparison by a subinterval in LIIS) *Given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, if $x_e < y_e$, then we say that $x < y$.*

We desire any part of an interval to extend the length in the LIIS problem, no matter how long it is. In other words, we do not consider the overlapping part. Suppose y has at least one value greater than x_e, then we can say that $x < y$. Before introducing the detailed algorithm, the domination concept is very critical for solving the problem, defined as follows.

Definition 10. (Domination) *Given two increasing interval sequences $X = \langle x_1, x_2, x_3, \ldots x_n\rangle$ and $Y = \langle y_1, y_2, y_3, \ldots, y_m\rangle$ with their ending value x_{ne} and y_{me}, respectively, if $x_{ne} < y_{me}$ and $|X| > |Y|$, then we say that X dominates Y.*

According to Definition 10, when x ends at a smaller integer with a larger length compared with y, the next inserted interval should be x, rather than y, if the insertion does not produce any overlap. As a result, y may becomes useless because it is dominated by x. However, Definition 10 cannot handle all situations in the LIIS problem with comparison by a subinterval. Based on Definition 9, if two intervals are overlapping and increasing, we can combine them by their non-overlapping parts into a single interval. If a short interval is fully contained in a long interval, then the short one should be removed.

TABLE V: An example of the algorithm for LIIS with comparison by full interval, where $X = \langle[42, 50], [20, 27], [30, 35], [3, 20], [23, 31], [27, 36]\rangle$. The answer is $\langle[3, 20], [27, 36]\rangle$ with length 28.

X		$\lvert x_i\rvert$	(ending value, length)		
x_1	[42, 50]	9	**(50, 9)**		
x_2	[20, 27]	8	**(27, 8)**	(50, 9)	
x_3	[30, 35]	6	(27, 8)	**(35, 14)**	~~**(50, 9)**~~
x_4	[3, 20]	18	**(20, 18)**	~~**(27, 8)**~~	~~**(35, 14)**~~
x_5	[23, 31]	8	(20, 18)	**(31, 26)**	
x_6	[27, 36]	10	(20, 18)	(31, 26)	**(36, 28)**

B. The Algorithm for Comparison by Full Interval

Two steps are performed for inserting a newly coming interval x_i into the answer. The first step is to find possible intervals for extending and to ensure that they are not overlapping. In the second step, after inserting x_i, we check if any domination exists. If not, the insertion is finished.

The example shown in Table V illustrates our algorithm for LIIS with comparison by full interval. In round 1, $x_1 = [42, 50]$. We initialize the answer with the ending value $x_{e_1} = 50$ and length $|x_1| = 9$, represented by $(50, 9)$. In round 2, $x_2 = [20, 27]$ cannot be extended from x_{e_1}, and there is no domination between x_2 and the existing answer. So we insert x_2 into the structure as a potential answer. In round 3, we can append $x_2 = [30, 35]$ to $(27, 8)$ to extend the length as 14 ending at 35, , represented by $(35, 14)$. Now $(50, 9)$ is dominated by $(35, 14)$ since the latter has more potential to become a better answer. Thus, $(50, 9)$ is deleted.

In round 4, $x_4 = [3, 20]$ has length 18 ending at 20, represented by $(20, 18)$. Then, both $(27, 8)$ and $(35, 14)$ are dominated by $(20, 18)$. So, the dominated elements are deleted. In round 5, we insert x_5 to increase the answer length. In the final round, $x_6 = [27, 36]$ can be extended from $(20, 18)$ to become $(36, 28)$. $(31, 26)$ and $(36, 28)$ cannot dominate each other, so they both are kept in the answer structure. Finally, the answer is got from $(36, 28)$, which consists of $\langle[3, 20], [27, 36]\rangle$ with length 28.

Our algorithm for LIIS with comparison by full interval is presented in Algorithm 2

There are two steps in every round. In the first step, we find the closest ending value to do extension, accomplished by the predecessor operation. It can be finished in a tree structure in $O(\log n)$. The other step is to check if there is domination. Each interval can only be removed at most once, so the process of the overall domination needs $O(n)$ time. Therefore, the

Algorithm 2 The algorithm for LIIS with comparison by full interval.

Input: An intervals sequence $X = \langle x_1, x_2, x_3, \ldots, x_n \rangle$, where $x_i = [x_{s_i}, x_{e_i}]$, $1 \leq i \leq n$.
Output: Length of the LIIS by comparison with full interval.

```
for i ← 1 to n do
    if t_pred ← T.predecessor(x_{s_i}) exist then
        t_cur ← (x_{e_i}, |t_pred| + |x_i|)
    else
        t_cur ← (x_{e_i}, |x_i|)
    T.insert(t_cur)
    if t_cur.previous dominates t_cur then
        T.remove(t_cur)
    while t_cur dominates t_cur.next do
        T.remove(t_cur.next)
return |T.end|
```

overall time complexity is O($n \log n$) and space complexity is O(q).

C. The Algorithm for Comparison by a Subinterval

In this algorithm, when inserting a newly coming interval, we need to check the closest non-overlapping and overlapping intervals. If we choose the overlapping intervals to extend, we should combine them to reduce the time and space complexities. Here, to record each potential solution, we use one 3-tuple (ending value, length of the combined interval, total length). The second tuple means that the subtotal length of the several consecutive intervals ends at the ending value.

Table VI shows an example of our algorithm. First, we insert $x_1 = [31, 40]$ into the answer structure and represent by $(40, 10, 10)$. In round 2, $x_2 = [37, 45]$, we use 37 to find the predecessor and the successor. However, no predecessor exists, so we can only choose the overlapping interval to extend. We merge the two intervals and get a new combined interval ending at 45 with length 15. In round 3, we insert a new interval $x_2 = [25, 36]$, represented by $(36, 12, 12)$, into the structure. There is no domination between these two intervals. In the next round, we have a shorter interval $x_4 = [26, 30]$, which is fully covered by x_3. Therefore, x_4 is removed immediately because of domination. The last round is the most important. $x_5 = [43, 51]$ has two choices, $(36, 12, 12)$ and $(45, 15, 15)$, to extend the length. It can be seen that the extension from $(36, 12, 12)$ gets the better result, $(51, 10, 22)$. Finally, we obtain the answer length 22 ending at 51, which is $\langle [25, 36], [42, 51] \rangle$.

The time complexity is O($n \log n$), and the space complexity is O(q). The pseudo code of this algorithm is omitted here.

IV. Conclusion

In this paper, we focus on the interval and LIS problems. Accordingly, we define the MIIS and LIIS problems and design algorithms to solve them. However, our definitions of interval comparison may not be so rigid. The comparison results with a single value and a subinterval are not unique when overlapping. We may add some restriction on the comparison. For example, given two intervals $x = [x_s, x_e]$ and $y = [y_s, y_e]$, we can make a restriction that $x_s < y_s$ and $x_e < y_e$ should be satisfied when $x < y$.

If the van Emde Boas (vEB) tree [17] is used to main the solution structure in the LIIS algorithm, the time complexity would be O($n \log \log m$), since each operation of insertion, deletion, predecessor and successor requires O($\log \log m$) time, where m denotes the maximal value in the inputs.

The interval sequence problem may be further generalized, such as the common part of two interval sequences (or with some constraints). The *longest common increasing subsequence* [5, 10, 11, 13, 18–20] may be a good direction for further study. The temperatures of a year are circular, so we may apply the concept of intervals on the *longest increasing circular subsequence* [7].

TABLE VI: An example of the algorithm for LIIS with comparison by a subinterval, where $X = \langle [31, 40], [37, 45], [25, 36], [26, 30], [42, 51] rangle$.

X		$\lvert x_i \rvert$	(ending value, |interval|,length)		
x_1	[31, 40]	10	**(40, 10, 10)**		
x_2	[37, 45]	9	**(45, 15, 15)**		
x_3	[25, 36]	12	**(36, 12, 12)**	(45, 15, 15)	
x_4	[26, 30]	5	~~(30, 5, 5)~~	(36, 12, 12)	(45, 15, 15)
x_5	[42, 51]	10	(36, 12, 12)	(45, 15, 15)	**(51, 10, 22)**

References

[1] M. R. Alam and M. S. Rahman, "A divide and conquer approach and a work-optimal parallel algorithm for the LIS problem," *Information Processing Letters*, Vol. 113, No. 13, pp. 470–476, 2013.

[2] M. H. Albert, M. D. Atkinson, D. Nussbaum, J.-R. Sack, and N. Santoro, "On the longest increasing subsequence of a circular list," *Information Processing Letters*, Vol. 101, No. 2, pp. 55–59, 2007.

[3] S. Bespamyatnikh and M. Segal, "Enumerating longest increasing subsequences and patience sorting," *Information Processing Letters*, Vol. 76, No. 1-2, pp. 7–11, 2000.

[4] W.-T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC 2005)*, Sanya, Hainan, China, 2005. Also in *Lecture Notes in Computer Science*, Vol. 3827, pp. 665–674, 2005.

[5] W.-T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," Vol. 13, pp. 277–288, 2007.

[6] M. Crochemore and E. Porat, "Fast computation of a longest increasing subsequence and application," *Information and Computation*, Vol. 208, No. 9, pp. 1054–1059, 2010.

[7] S. Deorowicz, "An algorithm for solving the longest increasing circular subsequence problem," *Information Processing Letters*, Vol. 109, No. 12, pp. 630–634, 2009.

[8] A. Elmasry, "The longest almost-increasing subsequence," *Information Processing Letters*, Vol. 110, No. 16, pp. 655–658, 2010.
[9] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, Vol. 20, pp. 350–353, 1977.
[10] M. Kutz, G. S. Brodal, K. Kaligosi, and I. Katriel, "Faster algorithms for computing longest common increasing subsequences," *Journal of Discrete Algorithms*, Vol. 9, No. 4, pp. 314–325, 2011.
[11] S.-F. Lo, K.-T. Tseng, C.-B. Yang, and K.-S. Huang, "A diagonal-based algorithm for the longest common increasing subsequence problem," *Theoretical Computer Science*, Vol. 815, pp. 69–78, 2020.
[12] J. M. Moosa, M. S. Rahman, and F. T. Zohora, "Computing a longest common subsequence that is almost increasing on sequences having no repeated elements," *Journal of Discrete Algorithms*, Vol. 20, pp. 12–20, 2013.
[13] Y. Sakai, "A linear space algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 99, No. 5, pp. 203–207, 2006.
[14] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, Vol. 13, pp. 179–191, 1961.
[15] T. T. Ta, Y.-K. Shieh, and C. L. Lu, "Computing a longest common almost-increasing subsequence of two sequences," *Theoretical Computer Science*, Vol. 854, pp. 44–51, 2021.
[16] C.-T. Tseng, C.-B. Yang, and H.-Y. Ann, "Minimum height and sequence constrained longest increasing subsequence," *Journal of Internet Technology*, Vol. 10, No. 2, pp. 173–178, 2009.
[17] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, Vol. 10, No. 1, pp. 99–127, 1976.
[18] C.-B. Yang and R. C.-T. Lee., "A fast algorithm for computing a longest common increasing subsequence," *Journal of the Chinese Institute of Engineers*, Vol. 10, No. 6, pp. 691–699, 1987.
[19] I.-H. Yang, C.-P. Huang, and K.-M. Chao, "A fast algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 93, No. 5, pp. 249–253, 2005.
[20] D. Zhu, L. Wang, T. Wang, and X. Wang, "A simple linear space algorithm for computing a longest common increasing subsequence," *IAENG International Journal of Computer Science*, Vol. 45, No. 3, pp. 472–477, 2018.

外送平台之最佳化問題

劉紀佑, 李佳衛

摘要

因為生活型態的轉變，促使外送產業的興起，而茁壯了具有高便利性的外送平台。外送平台所提供的外送服務已經成為現代城市生活中不可或缺的一部分。隨著科技的不斷進步和外送需求人口的增加，外送員這個行業的發展越來越迅速。特別是在全球新冠肺炎疫情的影響下，大眾對於外送飲食與包裹的需求越來越高，更加突顯了外送員的重要性。越來越多的顧客選擇減少出門的必要，促使外送員的需求不斷增加，而有越來越多人投入外送員這個行業。在大量的訂單需求與外送員人數密集的情況下，如何提高外送員配送訂單的效率，成為外送平台必須解決的問題。因此如何指派外送員，使得平台可以在最短時間內，完成指定之訂單數量，成為了我們所關注之議題。本論文旨在研究外送平台之最佳化問題，分別從 (1) 外送平台、(2) 顧客、(3) 外送員，三個不同角度來討論最佳化問題：(1) 外送平台：在最短的時間內完成所有訂單；(2) 顧客：等待訂單配送的時間最短；(3) 外送員：完成訂單之時間最短。本論文提出四種啟發式演算法來完成上述三種最佳化問題，在完成上述條件的前提下，更進一步要求所有外送員分配到的訂單差異數量能夠降低，用以改善外送員訂單數量分配不均，從而為城市生活帶來更多的便利。

關鍵字**:** O2O 外送平台, 最佳化問題, 任務指派問題, 排程演算法

I. 前言

當前社會因經濟和科技的快速發展，現代人的生活步調變得日趨匆忙。尤其在繁華的都市中，高效率和高便利性已成為人們日常生活中不可或缺的元素。臺灣趨勢研究在 2022 年的報告中指出 [1]：餐飲業者的銷售額在 2017 年至 2020 年有逐年上升的趨勢。同時在 2019 年之年末爆發影響全球甚鉅的新冠肺炎，在疫情與防疫政策的影響下，許多人降低了出門的意願，以減少接觸的機會，使得統合各家餐廳餐點與日常用品外送服務之外送平台快速興起，如 Uber Eats（優食）、foodpanda（富胖達）和 Lalamove（啦啦快送）等，皆為臺灣知名的外送平台。這些平台不僅為顧客提供了方便的服務，也為外送員提供了更多的工作機會。

自由時報在 2018 的報導 [2] 中指出，訂單的數量相較於 2013 年成長 100 倍，合作餐廳也成長 10 倍之多。天下雜誌於 2019 的調查 [3] 中指出，全台有超過四分之一的消費者使用過外送服務。至 2022 年時，MIC 產業情報研究所的研究 [4]，則是指出已經有超過 70% 的消費者使用過外送服務，且未來有意願使用的消費者高達八成。

根據主管機關的初步統計，在 2021 年底時，臺灣的外送員人數已達 10 萬多人 [5]。然而，在如此大量的外送員和外送訂單的情況下，如何妥善地分配訂單形成一個重要問題。根據未來流通研究所的調查數據，於 2020 年所提出的報告中指出 [6]：臺灣外送平台在整體餐飲產業中的占有比例，於 2019 年時，已由年初的 0.24% 攀升至年底的 1.02%，成長了四倍。在 2020 年疫情爆發後，外送平台消費金額於 2020 上半年同期成長近 300%、單季消費金額突破 30 億新台幣、單季消費筆數超過 1,500 萬筆，外送平台在整體餐飲產業中的佔比更是於 2020 年 4 月份創下 2.79% 的歷史新高。

Yuxin-Lu 等人在 2017 年提兩種基於遺傳演算法的元啟發式演算法 [7]，分別為階層式 (Hierarchical) 方法和整合式 (Integrated) 方法，用於同時解決訂單分配和路徑規劃問題，通過實驗數據的測試，發現階層式方法在目標值方面明顯優於整合式方法。

Zhilan Lou, Wanchen Jie 及 Shuzhu Zhang 在 2020 年的論文中提出一個非線性 (Non-linear) 的多目標優化模型 (Multi-objective Optimization Model) [8]，用於同時考慮訂單分配和配送路徑規劃的問題。

Jiulin Li, Senyan Yang, Wenbo Pan, Ziwen Xu 及 Bo Wei 在 2022 年的研究中提出了一種基於轉運站 (Transfer Stations) 的訂單分配策略 [9]，用於解決長距離美食訂單的配送路徑優化問題，並建立了一個混合整數規劃模型 (Mixed Integer Programming Model) 和一個自適應性大鄰域搜索演算法 (Adaptive Large Neighbourhood Search Algorithm) 求得最佳解。

林琬真在 2022 年的研究中 [10]，應用保護動機理論，對消費者進行外送平台使用之意願進行研究，得出關於疫情的知覺威脅脆弱度、自我效能、反應效能皆會正向影響民眾選用外送平台服務的意圖，代表因為疫情與相關防疫政令的關係，使民眾對於外送平台的使用意願增加。

王均安在 2022 年的研究中 [11]，則是將蟻群演算法應用於解決外送訂單之最佳化路徑問題。透過蟻群演算法改良訂單的派發對象，成功有效的使外送的距離、外送員的收入、成本皆有顯著的改善。

另外，鄭介東在 2018 年的研究中 [12] 也提出了整合 O2O 餐點外送之生產與配送問題，整合餐廳與平台外送員的路線規劃，決策接受哪些外送訂單；林國裕、卓信宏 [13] 在 2022 年的研究中，也討論過基於基因演算法的飲食外送員公平派單之多目標路徑最佳化，透過基因演算法的機制，得出一個更好的外送訂單派單順序。

外送服務的蓬勃發展，提供了新穎且方便快捷的生活體驗，但也引發了許多問題。其中一個重要的問題是如何有效地分配訂單給不同的外送員，以提高整體效率和客戶滿意度，同時兼顧每一位外送員的權益。現有的外送平台大多具備顧客可以透過支付額外費用，以獲得優先配送訂單的功能。但這種設計往往會導致外送員的配送路線將會變得十分複雜，並可能產生需要繞路或是重複來回路徑的情形發生，進而影響整體效率和顧客體驗。為了解決這個問

劉紀佑, 資訊工程學系, 國立臺東大學, 臺東縣臺東市大學路二段 369 號. E-mail: yoyo30618@gmail.com
李佳衛, 資訊工程學系, 國立臺東大學, 臺東縣臺東市大學路二段 369 號. E-mail: cwlee@nttu.edu.tw