The Longest Almost Wave Subsequence Problem

Shin-Cheng Lin, Kuo-Si Huang and Chang-Biau Yang

***Abstract*—In this paper, we first define a new *longest almost wave subsequence* (LaWS) problem by combining the *longest almost increasing subsequence* (LaIS) problem (Elmasry, 2010) and the *longest wave subsequence* (LWS) problem (Chen and Yang, 2020). The three problems are all generalized variants of the *longest increasing subsequence* (LIS) problem. The LaWS problem has two subprobems, LaWSt and LaWSr. Given a sequence A, a tolerance constant c, a trend sequence T and a constant r, the LaWSt problem aims to find the longest subsequence of A that matches the prefix of the trend sequence T. The LaWSr problem aims to find the longest subsequence of A within r segments, alternating between almost increasing and almost decreasing. The time and space complexities of our LaWSt algorithm are O$(n \log l)$ and O$(n + l)$, respectively, where n denotes the length of the given sequence, and l denotes the answer length. Our algorithm solves the LaWSr problem in O$(rn \log l)$ time and O$(n + rl)$ space.**

***Index Terms*—ongest increasing subsequence, longest almost increasing subsequence, longest wave subsequence, longest almost wave subsequence, trend sequence, segments**

I. Introduction

The *longest increasing subsequence* (LIS) problem [11] is a classic problem in computer science. Its goal is to find out the longest subsequence, obtained by the removal of zero or more elements (not necessarily consecutive) from an input sequence, which is increasing. For example, given a sequence $A = \langle 1, 12, 6, 8, 5, 7, 3\rangle$, then the LIS answer is $\langle 1, 6, 8\rangle$, $\langle 1, 6, 7\rangle$, or $\langle 1, 5, 7\rangle$, whose length is 3. Note that the LIS answer may not be unique. Almost all LIS algorithms [2, 3, 6, 9, 11], including related algorithms, first find the LIS length, and then get the LIS content with a tracing-back method or with a data structure for storing the tracing-back path. Thus, we focus the way for getting the LIS length, not go into details for obtaining the LIS content unless it is required.

For a given sequence of length n, Schensted [11] first defined the LIS problem in 1961, and then proposed an algorithm with O$(n \log n)$ time by using the Young tableau and the binary search. After that, when the input sequence is a permutation of $\{1, 2, ..., n\}$, Hunt and Szymanski [9] proposed an algorithm, based on the matching pairs with the van Emde Boas tree [14], to improve the time complexity to O$(n \log\log n)$. Then, Bespamyatnikh and Segal [3] enumerated all LIS answers with O$(n \log\log n)$ time. Crochemore and Porat [6] reduced the time complexity to O$(n \log\log l)$, where l denotes the LIS length. Alam and Rahman [2] proposed an O$(n \log n)$-time algorithm for the LIS problem based on the divide-and-conquer strategy.

This research work was partially supported by National Science and Technology Council of Taiwan under contract MOST 111-2221-E-110-034.

Shin-Cheng Lin is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan.

Kuo-Si Huang is with Department of Business Computing, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan. E-mail: huangks@webmail.nkmu.edu.tw.

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)

The LIS problem can only handle the monotonically incremental (or nondecremental) cases. However, some situations may not be monotonically increasing. For example, in the stock market, the prices may rise or fall alternately during a period, so it may not be monotonic. The temperature change is another example in real life. The temperature may be rising in recent years, but it may not be rising all the time. There may be some noise mixed in with the fluctuation. Therefore, the *longest almost increasing subsequence* (LaIS) [7] problem was first proposed in 2010 by Elmasry [7].

In the concept of *almost increasing*, it is allowed a small drop (a tolerance constant) to the maximal element that has appeared. For example, suppose that we are given a sequence $A = \langle 1, 12, 6, 8, 5, 7, 3\rangle$ and a constant $c = 3$, where c represents that the acceptable drop is less than 3 in the almost increasing subsequences. Then, the LaIS answer is $\langle 1, 6, 8, 7\rangle$ or $\langle 1, 6, 5, 7\rangle$, whose length is 4. In $\langle 1, 6, 8, 7\rangle$, the global trend is increasing, but there is a small drop between 8 and 7 with the drop $8-7 = 1$, which is less than the predefined tolerance constant $c = 3$. It is similar for $\langle 1, 6, 5, 7\rangle$. The LaIS problem can be solved in O$(n \log l)$ time [7], where l denotes the LaIS length.

Another generalized variant of the LIS problem is the *longest wave subsequence* (LWS) problem, proposed by Chen and Yang [5] in 2020. The LWS problem extends the concept of LIS, by dividing the answer into multiple segments, alternating between increasing and decreasing.

The LWS problem can be further divided into two subproblems: the *longest wave subsequence with trend* (LWSt) problem and the *longest wave subsequence within* r *segments* (LWSr) problem. The first one, for given a sequence and a trend sequence T, is to find a longest subsequence which matches the prefix of the trend T. The second one, for given a sequence and a constant r, is to find a longest subsequence which can be split into at most r segments, alternating between increasing and decreasing.

The related studies on the LIS-related problems are summarized in Table I.

This paper combines the LaIS problem and the LWS problem to define a new generalized problem, *longest almost wave subsequence* (LaWS). The answer of the LaWS problem has alternating trends in every two neighboring segments, and each segment allows some noise.

The LaWS problem can also be divided into two subproblems, *longest almost wave subsequence with trend* (LaWSt) problem and *longest almost wave subsequence within* r *segments* (LaWSr) problem. The time and space complexities of our LaWSt algorithm are $O(n \log l)$ and $O(n+l)$, respectively, where n denotes the length of the given sequence, and l denotes the answer length. In addition, our algorithm solves the LaWSr problem in $O(rn \log l)$ time and $O(n + rl)$ space.

The rest of the paper is organized as follows. Section II introduces the variants of the LIS problem and formally defines the LaWS problem. Sections III and IV present our algorithms for solving the LaWS problem. Finally, we summarize our conclusion in Section V.

II. Preliminaries

This section introduces two problems derived from the LIS problem, *longest almost increasing subsequence* (LaIS) and *longest wave subsequence* (LWS). Then, we define a generalized variant by combining the above two problems.

A. Longest Almost Increasing Subsequence

The *longest almost increasing subsequence* (LaIS) problem [7] allows a small drop (*tolerance constant*) c for each element in the answer.

Definition 1. (tolerance constant, almost increasing and almost decreasing) [7] *Given a sequence* $A = \langle a_1,\ a_2,\ a_3,\ \cdots,\ a_n\rangle$ *and a* tolerance constant *c, A is an* almost increasing sequence *if* $\forall i,\ 2 \le i \le n,\ \max\{a_k | 1 \le k \le i-1\} - c < a_i$, *and A is an* almost decreasing sequence *if* $\forall i,\ 2 \le i \le n,\ \min\{a_k | 1 \le k \le i-1\} + c > a_i$.

The LaIS problem aims to find a subsequence of a given sequence, which is the longest and almost increasing.

B. Longest Wave Subsequence

The *longest wave subsequence* (LWS) problem, a generalized variant of the LIS problem, was first proposed by Chen and Yang in 2020 [5]. The LWS answer can be split into many interleaved segments, where each segment is either strictly increasing or strictly decreasing.

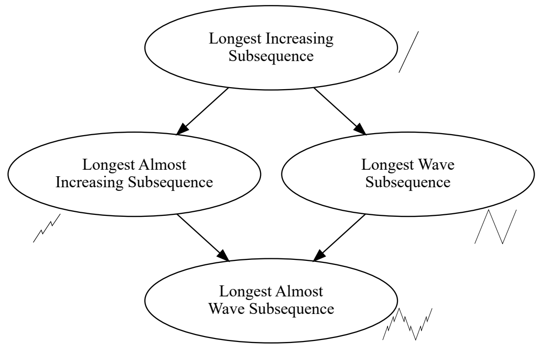
Definition 2. (trend) [5] *Given a numeric sequence* $B = \langle b_1,\ b_2,\ \cdots,\ b_n\rangle$, *where all of the values are distinct, its* trend sequence $T = \langle t_1,\ t_2,\ \cdots,\ t_n\rangle$ *is defined as follows.*

$$t_i = \begin{cases} 0 & \text{if } b_{i-1} > b_i \text{ for } 2 \le i \le n; \\ 1 & \text{if } b_{i-1} < b_i \text{ for } 2 \le i \le n; \\ \text{complement of } t_2 & \text{if } i = 1. \end{cases} \quad (1)$$

Definition 3. (turning point) [5] *Given a numeric sequence B with its trend sequence T, where $t_1 \ne t_2$, position i is a* turning point *if $t_i \ne t_{i+1}$. That is, the* turning point *is both the end of the previous segment and the start of the next segment.*

For example, sequence $\langle 2, 13, 5, 4\rangle$ can be divided into two segments, $\langle 2, 13\rangle$ and $\langle 13, 5, 4\rangle$. The former and latter trends are strictly increasing and strictly decreasing, respectively, where 13 is the turning point.

The LWS problem can be divided into two subproblems, the *longest wave subsequence problem with trend* (LWSt) and the *longest wave subsequence problem within r segments* (LWSr). In the LWSt problem, a numeric sequence A and a trend sequence T are given, and it aims to find the longest subsequence of A that matches the prefix of the trend T. Since the trend in T is fixed, the length of each segment in the LWSt answer is also fixed. In addition, the LWSr problem does not limit the length of each segment. Instead, it limits the maximum number of segments by a constant r.


Longest Increasing Subsequence
Longest Almost Increasing Subsequence
Longest Wave Subsequence
Longest Almost Wave Subsequence
Fig. 1: Extension flow of the problems with illustrations.

Here is an example for the LWSt problem. Given sequence $A = \langle 2, 13, 1, 5, 4\rangle$ and trend sequence $T = \langle t_1, t_2, t_3, t_4, t_5\rangle = \langle 0, 1, 0, 0, 1\rangle$, it is easy to see that t_2 and t_4 are turning points, where $\langle t_1, t_2\rangle$ represents an increasing segment, $\langle t_2, t_3, t_4\rangle$ represents a decreasing segment, and $\langle t_4, t_5\rangle$ represents an increasing segment. The LWSt answer of this example is $\langle 2, 13, 5, 4\rangle$, where $\langle 2, 13\rangle$ corresponds to $\langle t_1, t_2\rangle$, and $\langle 13, 5, 4\rangle$ corresponds to $\langle t_2, t_3, t_4\rangle$.

C. Longest Almost Wave Subsequence

In this paper, we first define the LaWS problem, which is a generalized variant of the LWS problem combined with the LaIS problem. Figure 1 shows the flow of the problem extension. The LaWS problem contains two subproblems, described in this section.

1) Longest Almost Wave Subsequence with Trend: The *longest almost wave subsequence with trend* (LaWSt) problem is a combination of the LWSt and the LaIS problems. The LaWSt answer can be divided into one or more segments according to the trend sequence. Each segment of the LaWSt answer is either almost increasing or almost decreasing.

Definition 4. (LaWSt problem) *Given a numeric sequence* $A = \langle a_1,\ a_2,\ a_3,\ \cdots,\ a_n\rangle$, *a trend sequence* $T = \langle t_1,\ t_2,\ t_3,\ \cdots,\ t_n\rangle$ *and a tolerance constant c, the* longest almost wave subsequence with trend (*LaWSt*) *is a longest subsequence of A that is consistent with the prefix of trend T. Here, $t_i \in \{0, 1\}$ for $1 \le i \le n$, $t_i = 1$ means almost increasing, and $t_i = 0$ means almost decreasing.*

For example, given sequence $A = \langle 2,\ 13,\ 1,\ 5,\ 4,\ 3,\ 6,\ 14,\ 15,\ 14,\ 16,\ 15,\ 17\rangle$, trend $T = \langle 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1\rangle$, and a tolerance constant $c = 3$, the LaWSt answer is $\langle 2,\ 1,\ 5,\ 14,\ 15,\ 14,\ 16,\ 15,\ 17\rangle$ with length 9. Note that $\langle 2,\ 1,\ 5,\ 14\rangle$, corresponding to $\langle t_1,\ t_2,\ t_3,\ t_4\rangle$, is an almost increasing segment; $\langle 14,\ 15,\ 14,\ 16,\ 15\rangle$, corresponding to $\langle t_4,\ t_5,\ \cdots,\ t_8\rangle$, is an almost decreasing segment; $\langle 15, 17\rangle$, corresponding to $\langle t_8, t_9\rangle$, is an almost increasing segment.

TABLE I: The development of LIS, LaIS, LCaIS and LWS problems. m and n: lengths of sequences A and B; l: length of the answer; r: number of segments; c: the tolerance constant in an almost increasing or almost decreasing segment.

The longest increasing subsequence (LIS) problem

Year	Author(s)	Time complexity	Note
1961	Schensted [11]	$O(n \log n)$	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]	$O(n \log \log n)$	Enumerate all answers
2010	Crochemore and Porat [6]	$O(n \log \log l)$	Split blocks
2013	Alam and Rahman [2]	$O(n \log n)$	Divide-and-conquer

The longest almost increasing subsequence (LaIS) problem

Year	Author(s)	Time complexity	Note
2010	Elmasry [7]	$O(n \log l)$	Dynamic programming

The longest common almost increasing subsequence (LCaIS) problem

Year	Author(s)	Time complexity	Note
2013	Moosa *et al.*[10]	$O(n(m + c^2))$	Dynamic programming, Match Pair
2021	Ta *et al.* [13]	$O(nml)$	Dynamic programming, Match Pair

The longest wave subsequence (LWS) problem

Year	Author(s)	Time complexity	Note
2020	Chen and Yang [5]	$O(n \log n)$	LWSt (with trend), greedy
2020	Chen and Yang [5]	$O(rn \log n)$	LWSr (within r segments), greedy
2021	Chang and Yang [4]	$O(mn)$	LCWSt (with trend)
2021	Chang and Yang [4]	$O(rmn)$	LCWSr (within r segments)

2) Longest Almost Wave Subsequence within r Segments: The *longest almost wave subsequence within r segments* (LaWSr) problem is a combination of the LWSr and the LaIS problems. In the LaWSr problem, the answer is split into at most r segments, with almost increasing and almost decreasing interleaved.

Definition 5. (LaWSr problem) *Given a numeric sequence* $A = \langle a_1,\ a_2,\ a_3,\ \ldots,\ a_n\rangle$*, a segment constraint* r *and a tolerance constant* c*, the* longest almost wave subsequence within r segments *(LaWSr) is the longest subsequence of* A *which can be split into at most r segments, and these segments are interleaved with almost increasing and almost decreasing.*

For example, suppose that we are given a sequence $A = \langle 9,$ 12, 11, 17, 10, 12, 9, 5, 11, 16$\rangle$, $r = 3$, and a tolerance constant $c = 3$. Then the LaWSr sequence is $\langle$9, 12, 11, 17, 10, 12, 9, 5, 11, 16$\rangle$, whose length is 10. Note that the first segment $\langle$9, 12, 11, 17$\rangle$ is almost increasing; the second segment $\langle$17, 10, 12, 9, 5$\rangle$ is almost decreasing; and the third segment $\langle$5, 11, 16$\rangle$ is almost increasing.

III. The Longest Almost Wave Subsequence Problem with Trend

Definition 6. (the tolerance constant c in LaWS) [7] *The* tolerance constant c *in LaWS is used to determine the value bound of each element. Given a sequence* $A = \langle a_1,\ a_2,\ a_3,\ \cdots,\ a_n\rangle$*, if* $\langle a_i, a_{i+1}, \cdots, a_j\rangle$ *is an* almost increasing segment*, then* $a_k > \max\{a_l | i \le l \le k-1\} - c$*, for* $i+1 \le k \le j$*. If* $\langle a_i, a_{i+1}, \cdots, a_j\rangle$ *is an* almost decreasing segment*, then* $a_k < \min\{a_l | i \le l \le k-1\} + c$*, for* $i+1 \le k \le j$*.*

The LaWSt answer can be split into segments according to the trend T, where each segment is either almost increasing or almost decreasing. For finding each segment, it is equivalent to solving an LaIS or an LaDS problem. Therefore, the tolerance constant c is used to determine the value bound. For example, in an almost increasing segment, if the current best sequence is $\langle 2, 1\rangle$ and $c = 3$, the bound of $\langle 2, 1\rangle$ is $\max\{2, 1\} - c = 2 - 3 = -1$. It means that any subsequent element greater than -1 can make the answer longer.

To solve the LaWSt problem, we maintain a working structure U with a nondecreasing order (for almost increasing) or a nonincreasing order (for almost decreasing). Each element $u_j \in U$ is the best element for further extension at length j.

Definition 7. (working structure U for LaWSt) *Each element* u_j *in the* working structure $U = \langle u_1,\ u_2,\ u_3,\ \cdots,\ u_l\rangle$ *is the best element (defined later) in the current segment at length j for the LaWSt answer,* $1 \le j \le l$*, where* l *denotes the LaWSt length.*

Note that U is dynamically maintained when the given sequence is linearly scanned.

Definition 8. (the best element) *Each* best element u_j *in* $U = \langle u_1,\ u_2,\ u_3,\ \cdots,\ u_l\rangle$ *is the maximal value of the almost increasing segment or the minimal value of the almost descending segment. If* $\langle a'_i, a'_{i+1}, \cdots, a'_j\rangle$ *is an almost increasing segment, then* $u_j = \max\{a'_k | i \le k \le j\}$. *And If* $\langle a'_i, a'_{i+1}, \cdots, a'_j\rangle$ *is an almost decreasing segment, then* $u_j = \min\{a'_k | i \le k \le j\}$.

For example, suppose an almost increasing segment is ⟨2, 1, 5⟩. Then the best sequences of lengths 1, 2 and 3 in an almost increasing segment are ⟨1⟩, ⟨2, 1⟩ and ⟨2, 1, 5⟩ respectively. Then the working structure U will be set as ⟨1, 2, 5⟩, since the maximum values are 1, 2, and 5 at lengths 1, 2, and 3, respectively.

When we deal with a new element in the given sequence and want to append it to U, we should confirm that the element conforms to the current trend. If the new element can make the answer longer, then we can append it to U. Otherwise, we have to replace an element already in U by this new element such that the best extensibility is preserved. Thus, we need the concepts of the predecessor and successor to maintain U.

Definition 9. (predecessor of e in V) *The* predecessor *of* e *with respect to all elements in* V*, denoted as* $p^+(V, e)$*, is the largest* $v_j \in V$ *that* $v_j < e$ *if* V *is a structure of an almost increasing segment. The* predecessor *of* e *with respect to all elements in* V*, denoted as* $p^-(V, e)$*, is the smallest* $v_j \in V$ *that* $v_j > e$ *if* V *is a structure of an almost decreasing segment. If no* predecessor *can be obtained, it is set to* $NULL$.

Definition 10. (successor of e in V) *The* successor *of* e *with respect to all elements in* V*, denoted as* $s^+(V, e)$*, is the smallest* $v_j \in V$ *that* $v_j > e$ *if* V *is a structure of an almost increasing segment. The* successor *of* e *with respect to all elements in* V*, denoted as* $s^-(V, e)$*, is the largest* $v_j \in V$ *that* $v_j < e$ *if* V *is a structure of an almost decreasing segment. If no* successor *can be obtained, it is set to* $NULL$.

Table II shows an example of the entire process of our LaWSt algorithm for linearly scanning A. The corresponding resulting subsequence for each length is shown in Table III.

In Table II, the first segment is almost increasing, so we maintain the working structure U as a nondecreasing list. We put the first element $a_1 = 2$ directly into U. And, we simply append $a_2 = 13$ to U, then the LaWSt length becomes 2.

The next one is $a_3 = 1$. We first insert 1 into U, so U temporarily becomes ⟨1, 2, 13⟩. Next, find the predecessor of $a_3 + c = 4$, which is 2. Then find the successor of 2, which is 13. This means that 13 should be removed, since the LaWSt sequence with length 2 has a smaller (better) upper bound $2 + c$, which has the better extensibility in the future. So after inserting $a_3 = 1$, U is finally modified as ⟨1, 2⟩. At this moment, the best LaWSt answer with length 1 is ⟨1⟩ and that with length 2 is ⟨2, 1⟩.

Here is an explanation why 13 is removed. While dealing with a newly coming element a_i, the length of U may be either unchanged or increased by one. If $a_i > \max_{e\in U}\{e\} - c$, we can simply add a_i to U to increase the length. Otherwise,

TABLE II: An example of the working structure U in the LaWSt algorithm, where A = ⟨2, 13, 1, 5, 4, 3, 6, 14, 15, 14, 16, 15, 17⟩, T = ⟨0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1⟩ and $c = 3$. The LaWSt answer is ⟨2, 1, 5, 14, 15, 14, 16, 15, 17⟩, with length 9.

T		0	1	1	1	0	0	0	0	1
A	l(length)	1	2	3	4	5	6	7	8	9
a_1	2	**2**								
a_2	13	2	**13**							
a_3	1	**1**	**2**							
a_4	5	1	2	**5**						
a_5	4	1	2	**4**	**4**					
a_6	3	1	2	4	4	**3**				
a_7	6	1	2	4	**6**	**4**				
a_8	14	1	2	4	**14**	4				
a_9	15	1	2	4	**15**	**14**				
a_{10}	14	1	2	4	15	14	**14**			
a_{11}	16	1	2	4	**16**	**15**	**14**	**14**		
a_{12}	15	1	2	4	16	15	**15**	**14**	**15**	
a_{13}	17	1	2	4	16	15	15	14	15	**17**

TABLE III: The resulting subsequence obtained in Table II for each length.

A		Resulting subsequences
a_1	2	⟨2⟩
a_2	13	⟨2⟩, ⟨2, 13⟩
a_3	1	⟨1⟩, ⟨2, 1⟩
a_4	5	⟨1⟩, ⟨2, 1⟩, ⟨2, 1, 5⟩
a_5	4	⟨1⟩, ⟨2, 1⟩, ⟨2, 1, 4⟩, ⟨2, 1, 5, 4⟩
a_6	3	-, -, -, ⟨2, 1, 5, 4⟩, ⟨2, 1, 5, 4, 3⟩
a_7	6	-, -, -, ⟨2, 1, 5, 6⟩, ⟨2, 1, 5, 4, 6⟩
a_8	14	-, -, -, ⟨2, 1, 5, 14⟩, ⟨2, 1, 5, 4, 6⟩
a_9	15	-, -, -, ⟨2, 1, 5, 15⟩, ⟨2, 1, 5, 14, 15⟩
a_{10}	14	-, -, -, ⟨2, 1, 5, 15⟩, ⟨2, 1, 5, 14, 15⟩, ⟨2, 1, 5, 14, 15, 14⟩
a_{11}	16	-, -, -, ⟨2, 1, 5, 16⟩, ⟨2, 1, 5, 15, 16⟩, ⟨2, 1, 5, 14, 15, 14⟩, ⟨2, 1, 5, 14, 15, 14, 16⟩
a_{12}	15	-, -, -, ⟨2, 1, 5, 16⟩, ⟨2, 1, 5, 15, 16⟩, ⟨2, 1, 5, 15, 16, 15⟩, ⟨2, 1, 5, 14, 15, 16, 15⟩, ⟨2, 1, 5, 14, 15, 14, 16, 15⟩
a_{13}	17	-, -, -, -, -, -, -, ⟨2, 1, 5, 14, 15, 14, 16, 15⟩, ⟨2, 1, 5, 14, 15, 14, 16, 15, 17⟩

the length remains unchanged. $a_3 = 1$ cannot make U longer. However, the segment containing 1 should have better extensibility in an increasing segment. Therefore, we replace 13 with the smaller 1. After this update, U is changed from ⟨2, 13⟩ to ⟨1, 2⟩, and the most extendable subsequence of length 2 goes from ⟨2, 13⟩ to ⟨2, 1⟩. The length is the same, but the maximum value at length 1 becomes 1 and at length 2 becomes 2, which have better extensibility in the future.

We simply add the next element $a_4 = 5$ to U, since it can make U longer. After inserting $a_5 = 4$ into U, U becomes ⟨1, 2, 4, 5⟩ temporarily, where 4 is placed before 5 because U is nondecreasing. Although 4 is not the maximum value in U, we still verify whether the result is out of bound. The predecessor of $a_5 + c = 7$ is 5, and there is no successor of 5, which means U = ⟨1, 2, 4, 5⟩ is legal. At this moment, the best LaWSt answer with length 3 is $\langle 2, 1, 4\rangle$ and with length 4 is $\langle 2, 1, 5, 4\rangle$. Moreover, $t_4 = 1 \ne 0 = t_5$ means that the current segment (almost increasing) ends and a new segment (almost decreasing) starts. In our algorithm, when an old segment finishes, the information for the old segment will not be changed any more. And for the new segment, we have to create a new U, initialized as the only one element (a_5)

which is the ending element of the old segment. So we put $a_5 = 4$ at u_4, and it is a turning point.

Now we look at the second segment. The first segment is almost increasing, ending with 4. Therefore, the second segment is almost decreasing, starting with this 4. We maintain the working structure U as a nonincreasing list for the second segment, initialized with only one element of this $a_5 = 4$. For $a_6 = 3$, which can be added simply to U, becoming $U = \langle 4, 3\rangle$. The next element is $a_7 = 6$. After inserting $a_7 = 6$ into U, the predecessor of $a_7 - c = 3$ is 4, and the successor of 4 is 3. So 3 is removed from U. At this moment, the best LaWSt answer with length 4 is $\langle 2, 1, 5, 6\rangle$ and with length 5 is $\langle 2, 1, 5, 4, 6\rangle$.

The dealing of $a_8 = 14$, $a_9 = 15$ and $a_{10} = 14$ can be done similarly. When it comes to $a_{11} = 16$, it is inserted into U. Nothing is removed since no element is out of bound. After $a_{12} = 15$ is inserted into U, U becomes $\langle 16, 15, 15, 14, 14\rangle$ temporarily. Here, we reach another turning point, since $t_8 = 0 \neq t_9 = 1$. Now the second segment finishes and a new segment begins. Thus, for the new segment, we initialize U to contain only $a_{12} = 15$, which is the ending element of the second segment, and the starting of the third segment.

For easy description of Table II, we regard U as a linear list. To reduce the time complexity, U can be implemented by a more efficient data structure, such as an AVL tree [1] or a red-black tree [8]. In these data structures, each operation of insertion, deletion, predecessor or successor can be done in $O(\log |U|)$ time. Obviously, $|U|$ is bounded by $O(l)$, where l denotes the LaWSt length. Since each of the above four operations for each element of A is done at most once. Therefore, the total time complexity of our LaWSt algorithm is $O(n \log l)$. Our LaWSt algorithm is formally presented in Algorithm 1.

Algorithm 1 Computing the LaWSt length.

Input: A numeric sequence $A = \langle a_1, a_2, a_3, \ldots, a_n\rangle$, a trend sequence $T = \langle t_1, t_2, t_3, \ldots, t_n\rangle$, $t_i \in \{0, 1\}$, and a tolerance constant c.

Output: The LaWSt length l.

```
1:  $l \leftarrow 1$, $U \leftarrow \{a_1\}$
2:  for $i = 2$ to $n$ do
3:      Insert $a_i$ into $U$
4:      if $t_{l+1} = 1$ then                ▷ almost increasing segment
5:          $p \leftarrow s^+(U, p^+(U, a_i + c))$
6:          if $p$ is found then             ▷ $p$ is not NULL
7:              Delete $p$
8:          else
9:              $l \leftarrow l + 1$
10:     else if $t_{l+1} = 0$ then           ▷ almost decreasing segment
11:         $p \leftarrow s^-(U, p^-(U, a_i - c))$
12:         if $p$ is found then             ▷ $p$ is not NULL
13:             Delete $p$
14:         else
15:             $l \leftarrow l + 1$
16:     if $t_l \neq t_{l+1}$ then            ▷ turning point
17:         $U \leftarrow \{a_i\}$ ▷ new segment, initialized as only one element
```

It is worth mentioning that Algorithm 1 cannot handle the case of $c = 0$ (strictly increasing or decreasing). To handle the case of $c = 0$, we can easily modify Line 3 as follows.

If the value a_i is already in U, then discard a_i and do nothing.
Otherwise, set $c = \epsilon$ and insert a_i into U.
In the above, ϵ is a small positive constant less than the difference between any pair of elements in A.

Theorem 1. *Updating only the current segment in Algorithm 1 can maintain the best extensibility of the current segment.*

Proof. The proof is omitted here [12]. □

Theorem 2. *The LaWSt problem can be solved in $O(n \log l)$ time and $O(n + l)$ space, where n denotes the length of the given sequence and l denotes the LaWSt length.*

Proof. The proof is omitted here [12]. □

IV. The Longest Almost Wave Subsequence Problem within r Segments

The core concept of our LaWSr algorithm is that the element with the least number of segments or the larger length is better. We maintain a working structure Q to solve the LaWSr problem. Q is divided into r parts, where each part Q_k, $1 \leq k \leq r$, corresponds to segment k. Each segment is considered independently when the update is done within the segment. After updating a segment, we further check if inheritance from the previous segment would get a better answer. Each element $e \in Q_k$ is a 2-tuple of the form $e = \langle e.val, e.len\rangle$, where $e.val$ is the maximum (minimum) value of the LaWSr answer at length $e.len$ in an almost increasing (decreasing) segment k. Note that $e.len$ is not the answer length ending at $e.val$. Only elements with better extensibility are recorded in Q, described with the domination concept in Definition 11.

Definition 11. (domination) *For two elements $\alpha, \beta \in Q_k$ in an almost increasing segment, α is said to be* dominated *by β* (β dominates α) *if $\alpha.val \geq \beta.val$ and $\alpha.len \leq \beta.len$; in an almost decreasing segment, α is said to be* dominated *by β if $\alpha.val \leq \beta.val$ and $\alpha.len \leq \beta.len$.*

The first segment in the LaWSr answer may be either almost increasing or almost decreasing. Thus, our algorithm tries both cases for the first segment. And then we select the better result obtained from both cases as the final answer. Without loss of generality, the first segment in the following example is assumed to be almost increasing.

Each $e \in Q_k$ is a bound element, meaning the bound value $e.val$ in segment k at length $e.len$. Updating within a segment can obtain the bound element. However, in the LaWSr problem, a bound element may inherit from the previous segment. So, we have to check which element is better, with the domination concept in Definition 11.

Table IV shows an example of the entire process of our LaWSr algorithm. Table V shows the resulting sequence in Table IV for each length after $A_{1..i}$ has been processed, $1 \leq$

TABLE IV: An example of our LaWSr algorithm with $A = \langle 9,$ $12, 11, 17, 10, 12, 9, 5, 11, 16\rangle$, $r = 3$ and $c = 3$. The LaWSr length is 10, and the LaWSr sequence is ⟨9, 12, 11, 17, 10, 12, 9, 5, 11, 16⟩, where the first segment ⟨9, 12, 11, 17⟩ is almost increasing, the second segment ⟨17, 10, 12, 9, 5⟩ is almost decreasing, and the third segment ⟨5, 11, 16⟩ is almost increasing.

length		1	2	3	4	5	6	7	8	9	10
$a_1 = 9$	Q_1	**9**									
	Q_2	**9**									
	Q_3	**9**									
$a_2 = 12$	Q_1	9	**12**								
	Q_2	~~9~~	**12**								
	Q_3	9	**12**								
$a_3 = 11$	Q_1	9	**11**	12							
	Q_2		12	**11**							
	Q_3	9	~~12~~ -	**11**							
$a_4 = 17$	Q_1	9	11	12	**17**						
	Q_2		~~12~~	~~11~~	**17**						
	Q_3	9	-	11	**17**						
$a_5 = 10$	Q_1	9	**10**	11	~~17~~ 12						
	Q_2				17	**10**					
	Q_3	9	-	~~11~~ -	~~17~~ -	**10**					
$a_6 = 12$	Q_1	9	10	11	12	**12**					
	Q_2				17	**12**	10				
	Q_3	9	-	-	-	10	**12**				
$a_7 = 9$	Q_1	9	**9**	10	~~12~~ 11	12					
	Q_2				17	12	10	**9**			
	Q_3	~~9~~	-	-	-	~~10~~	~~12~~	**9**			
$a_8 = 5$	Q_1	~~9~~ **5**	9	10	11	12					
	Q_2				17	12	10	9	**5**		
	Q_3							~~9~~	**5**		
$a_9 = 11$	Q_1	5	9	10	11	**11**	12				
	Q_2				17	12	**11**	10	~~5~~ 9		
	Q_3								5	**11**	
$a_{10} = 16$	Q_1	5	9	10	11	12	12	**16**			
	Q_2				17	~~12~~ -	~~11~~ -	~~10~~ **16**	9		
	Q_3								5	11	**16**

$i \leq n$. For easily describing the entire process, we view each Q_k as a linear array, where $Q_{k,j}$ stores the element value and j denotes its length.

The first element $a_1 = 9$ is inserted into Q_1, Q_2 and Q_3 as initialization. The next element $a_2 = 12$ is simply appended to Q_1 to increase the length. Then after inserting 12 into Q_2 for the second segment (almost decreasing), Q_2 temporarily becomes $\langle 12, 9\rangle$. Next, we find the predecessor of $a_2 - c = 9$, which is 12 in $\langle 12, 9\rangle$. Then the successor of 12 in $\langle 12, 9\rangle$ is 9. Thus, 9 is removed from Q_2. Now Q_2 becomes $\langle 12\rangle$ with length 1 after updating within the segment. In brief, 12 replaces 9 to get more extensibility for the future.

Now consider the inheritance of Q_2 from Q_1. Q_2 is $\langle 12\rangle$ with length 1, but Q_1 is $\langle 9, 12\rangle$ and 12 is with length 2. Inheriting from Q_1, we can obtain better Q_2 in both lengths and bounds. As a result, Q_2 inherits the end of Q_1 as the beginning of Q_2 at length 2. Note that $a_2 = 12$ is the currently inserted value, not necessarily the end of Q_1, because the end of Q_k is not necessarily the value with the length in the answer sequence. Finally, Q_2 becomes $\langle 12\rangle$ with length 2.

TABLE V: The resulting subsequence in Table IV for each length, where $A = \langle 9, 12, 11, 17, 10, 12, 9, 5, 11, 16\rangle$, $r = 3$ and $c = 3$. Here, the first segment is almost increasing.

A		Resulting subsequences
a_1	9	$\langle 9\rangle$
a_2	12	$\langle 9\rangle$, $\langle 9, 12\rangle$
a_3	11	$\langle 9\rangle$, $\langle 9, 11\rangle$, $\langle 9, 12, 11\rangle$
a_4	17	$\langle 9\rangle$, $\langle 9, 11\rangle$, $\langle 9, 12, 11\rangle$, $\langle 9, 12, 11, 17\rangle$
a_5	10	$\langle 9\rangle$, $\langle 9, 10\rangle$, $\langle 9, 11, 10\rangle$, $\langle 9, 12, 11, 10\rangle$, $\langle 9, 12, 11, \mathbf{17}, 10\rangle$
a_6	12	$\langle 9\rangle$, $\langle 9, 10\rangle$, $\langle 9, 11, 10\rangle$, $\langle 9, 12, 11, 10\rangle$, $\langle 9, 12, 11, 10, 12\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12\rangle$
a_7	9	$\langle 9\rangle$, $\langle 9, 9\rangle$, $\langle 9, 10, 9\rangle$, $\langle 9, 11, 10, 9\rangle$, $\langle 9, 12, 11, 10, 12\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12\rangle$, $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9\rangle$
a_8	5	$\langle 5\rangle$, $\langle 9, 9\rangle$, $\langle 9, 10, 9\rangle$, $\langle 9, 11, 10, 9\rangle$, $\langle 9, 12, 11, 10, 12\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12\rangle$, $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9, 5\rangle$
a_9	11	$\langle 5\rangle$, $\langle 9, 9\rangle$, $\langle 9, 10, 9\rangle$, $\langle 9, 11, 10, 9\rangle$, $\langle 9, 11, 10, 9, 11\rangle$ $\langle 9, 12, 11, 10, 12, 11\rangle$, $\langle 9, 12, 11, \mathbf{17}, 10, 12, 11\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9, \mathbf{5}, 11\rangle$, $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9, 11\rangle$
a_{10}	16	$\langle 5\rangle$, $\langle 9, 9\rangle$, $\langle 9, 10, 9\rangle$, $\langle 9, 11, 10, 9\rangle$, $\langle 9, 11, 10, 9, 11\rangle$ $\langle 9, 12, 11, 10, 12, 11\rangle$, $\langle 9, 12, 11, 10, 12, 11, 16\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9, 11\rangle$, $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9, 5, 11\rangle$ $\langle 9, 12, 11, \mathbf{17}, 10, 12, 9, \mathbf{5}, 11, 16\rangle$

Then the update of Q_3 for the third almost increasing segment is done similarly. After inserting $a_2 = 12$, Q_3 becomes $\langle 9, 12\rangle$ and it does not violate the increasing trend. Placing the 12 in the correct position completes the in-segment update. Then consider inheritance. Q_2 is $\langle 12\rangle$ with length 2. Either length or bound is the same in both Q_2 and Q_3. The inheritance does not get better. Thus, the update of Q_3 is completed.

The next element is $a_3 = 11$. Starting from the first segment, insert 11 into Q_1, and check whether the insertion fulfills the almost increasing trend. The predecessor of $a_3 + c = 14$ in $Q_1 = \langle 9, 11, 12\rangle$ is 12, but the successor of 12 cannot be found in Q_1. The insertion of a_3 into Q_1 is done. Then insert $a_3 = 11$ into Q_2. 11 can be placed directly at the end of Q_2. Moreover, the inheritance from Q_1 cannot obtain a better result.

Now go on to update Q_3. After inserting $a_3 = 11$ into Q_3, Q_3 becomes $\langle 9, 11, 12\rangle$ temporarily and it conforms to the almost increasing trend. Then we check inheritance from Q_2. 11 in Q_2 is of length 3, which dominates 11 in Q_3 with length 2 and 12 in Q_3 with length 3. Then, 11 with length 3 is inserted into Q_3, and the dominated elements are deleted. In a simple summary, when dealing a new element in Q_k, we should perform in-segment update and inheritance.

The next element $a_4 = 17$ is inserted Q_1 simply. After inserting 17 into Q_2, Q_2 is temporarily changed to $\langle 17, 11\rangle$ (12 is removed because it is out of bound). Then the inheritance 17 with length 4 from Q_1 would make Q_2 better. Thus, 17 with length 2 in Q_2 and 11 with length 3 in Q_2 are removed since they are dominated. $a_4 = 17$ is inserted into Q_3 simply.

After processing all elements in A, the algorithm finishes. However, the LaWSr problem does not restrict that the first segment is almost increasing. Thus, we can use the same algorithm to build another working structure Q with the same input, but this time the first segment is almost decreasing.

Our LaWSr algorithm is formally presented in Algorithm 2. Here, last(Q_k) denotes a function to get the 2-tuple of maximum (minimum) value in nondecreasing (nonincreasing)

Q_k. It can be easily done by recording the maximum or minimum value in Q_k. Without loss of generality, assume that the first segment is almost increasing. It can easily be modified as that the first segment is almost decreasing. Algorithm 2 can be divided into two parts: in-segment update and inheritance. In in-segment update, a_i is inserted into Q_k. It can be done similar to one segment in the LaWSt algorithm. The second part is inheritance from Q_{k-1}. Inheritance will remove all elements in Q_k dominated by $\langle a_i$, the answer length ending at a_i in $Q_{k-1}\rangle$, which can be obtained by $\mathrm{p}^+(Q_1, a_i + c + \epsilon).len$ in an almost increasing segment and $\mathrm{p}^-(Q_1, a_i - c - \epsilon).len$ in an almost decreasing segment.

Algorithm 2 Computing the LaWSr length with the first segment being almost increasing.

```
Input: A numeric sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$, a number $r$ of segments, and a tolerance constant $c$.
Output: The LaWSr length $l$.
1:  for $i = 1$ to $n$ do
2:      for $k = 1$ to $r$ do
3:          if $k \mod 2 = 1$ then                          ▷ almost increasing segment
4:              Insert $\langle a_i, \mathrm{p}^+(Q_k, a_i).len + 1\rangle$ into $Q_k$   ▷ in-segment update
5:              $p \leftarrow \mathrm{s}^+(Q_k, \mathrm{p}^+(Q_k, a_i + c))$
6:              if $p$ is found then
7:                  Remove $p$ from $Q_k$
8:              $length \leftarrow$ get the answer length ending at $a_i$ in $Q_{k-1}$
9:              $p \leftarrow \mathrm{s}^+(Q_k, a_i - \epsilon)$   ▷ $\epsilon$ is a small positive constant
10:             if $p.len < length$ then                    ▷ inheritance from $Q_{k-1}$
11:                 while $p$ is found and $p.len \leq length$ do
12:                     Dominant remove $p$ from $Q_k$, $p \leftarrow \mathrm{s}^+(Q_k, a_i - \epsilon)$
13:                 Insert $\langle a_i, length\rangle$ into $Q_k$
14:         else if $k \mod 2 = 0$ then                     ▷ almost decreasing segment
15:             Insert $\langle a_i, \mathrm{p}^-(Q_k, a_i).len + 1\rangle$ into $Q_k$   ▷ in-segment update
16:             $p \leftarrow s^-(Q_k, p^-(Q_k, a_i - c))$
17:             if $p$ is found then
18:                 Remove $p$ from $Q_k$
19:             $length \leftarrow$ get the answer length ending at $a_i$ in $Q_{k-1}$
20:             $p \leftarrow \mathrm{s}^-(Q_k, a_i + \epsilon)$   ▷ $\epsilon$ is a small positive constant
21:             if $p.len < length$ then                    ▷ inheritance from $Q_{k-1}$
22:                 while $p$ is found and $p.len \leq length$ do
23:                     Dominant remove $p$ from $Q_k$, $p \leftarrow \mathrm{s}^-(Q_k, a_i + \epsilon)$
24:                 Insert $\langle a_i, length\rangle$ into $Q_k$
25:     $l \leftarrow \mathrm{last}(Q_r).len$                   ▷ answer length
```

In Algorithm 2, we use a 2-tuple $\langle val, len\rangle$ to store each element, where $e.val$ denotes the element value, and $e.val$ is the maximum (minimum) value of the LaWSr answer at length $e.len$ in an almost increasing (decreasing) segment k. In order to reduce the time complexity, we can use an *enhanced AVL-tree* to store the 2-tuples $\langle val, \delta\rangle$, instead of $\langle val, len\rangle$, where val is one of a_i, $1 \leq i \leq n$, and δ is the length difference compared to its parent node. It supports the searching by the value or length, insertion, removal, and successor. Each operation can be accomplished in O($\log l$) time, where l represents the number of nodes stored in the tree. The detailed implementation can be found in [12].

Theorem 3. *The LaWSr problem can be solved in $O(rn \log l)$ time, and $O(n + rl)$ space, where n denotes the length of the given sequence and l denotes the LaWSr length.*

Proof. The proof is omitted here [12]. □

V. Conclusion

In this paper, we first define a new problem, the longest almost wave subsequence (LaWS) problem, which is a more generalized variant of the longest increasing subsequence (LIS) problem. It is also a combination of the longest almost increasing subsequence (LaIS) and the longest wave subsequence (LWS) problems. The LaIS problem defines the almost increasing segment that is not strictly increasing. And the LWS problem makes the answer have one or more increasing or decreasing segments. Similarly, the answer of the LaWS has one or more almost increasing or decreasing segments. Besides, within each almost increasing (decreasing) segment, a small drop (raise) bounded by a tolerance constant is allowed.

There are two versions of the LaWS problem, including the longest almost wave subsequence with trend (LaWSt) and longest almost wave subsequence within r segments (LaWSr) problems. We also propose efficient algorithms for solving the two versions of the LaWS problem. The LaWSt problem can be solved in O($n \log l$) time and O($n+l$) space, and the LaWSr problem can be solved in O($rn \log l$) time and O($n+rl$) space, where n denotes the length of the given sequence and l denotes the length of the answer.

The LaWS problem is helpful for analyzing the sequence with a trend but some fluctuation, such as stock price or temperature. In the analysis, there may be more than one series to be processed. Thus there may be a new problem combined with the LCS problem to form an LCaWS problem. And we may have to find the common trends among three or more sequences.

In order to analyze sequences more precisely, it may be necessary to have different constraint for each segment. This can be done by given difference tolerance constant for each segment, which may be a new research direction.

References

[1] G. M. Adelson-Velsky and E. Landis, "An algorithm for organization of information," *Doklady Akademii Nauk SSSR*, Vol. 146, No. 2, pp. 263–266, 1962.

[2] 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.

[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] Y.-C. Chang and C.-B. Yang, "The longest common wave subsequence problem," *Proceedings of the 38th Workshop on Combinatorial Mathematics and Computation Theory*, Taipei, Taiwan, pp. 9–17, 2021.

[5] G.-Z. Chen and C.-B. Yang, "The longest wave subsequence problem: Generalizations of the longest increasing subsequence problem," *Proceedings of the 37th*

Workshop on Combinatorial Mathematics and Computation Theory, Kaohsiung, Taiwan, pp. 28–33, 2020.

[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] A. Elmasry, “The longest almost-increasing subsequence,” *Information Processing Letters*, Vol. 110, No. 16, pp. 655–658, 2010.

[8] L. J. Guibas and R. Sedgewick, “A dichromatic framework for balanced trees,” *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (SFCS 1978)*, Barcelona, Spain, pp. 8–21, 1978.

[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] 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.

[11] C. Schensted, “Longest increasing and decreasing subsequences,” *Canadian Journal of Mathematics*, Vol. 13, pp. 179–191, 1961.

[12] Shin-Cheng, “The longest almost wave subsequence problem,” *Master’s Thesis, Department of Computer Science and Engineering, National Sun Yat-sen University*, Kaohsiung, Taiwan, 2022.

[13] 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.

[14] 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.