Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 96 (2005) 197–201
Information Processing Letters
www.elsevier.com/locate/ipl

Optimal algorithms for locating the longest and shortest segments satisfying a sum or an average constraint

Kuan-Yu Chen [a], Kun-Mao Chao [a,b,*]

[a] *Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106*
[b] *Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan 106*

Received 10 May 2005; received in revised form 3 August 2005; accepted 10 August 2005

Available online 8 September 2005

Communicated by F.Y.L. Chin

Abstract

We study several fundamental problems arising from biological sequence analysis. Given a sequence of real numbers, we present two linear-time algorithms, one for locating the "longest" sum-constrained segment, and the other for locating the "shortest" sum-constrained segment. These two algorithms are based on the same framework and run in an online manner, hence they are capable of handling data stream inputs. Our algorithms also yield online linear-time solutions for finding the longest and shortest average-constrained segments by a simple reduction.

Keywords: Sequence analysis; Longest segment; Shortest segment; Algorithms

1. Introduction

Computer analysis of newly determined DNA sequences, i.e., strings of the four letters A, C, G, and T, usually includes a task that identifies GC-rich segments. A segment of a sequence, sometimes called an interval or a region, is a contiguous subsequence of that sequence [2]. Segments of a DNA sequence that are rich in nucleotides G and C are usually biologically significant. As a consequence, two lines of investigation into such segments arise. One line tries to locate a segment whose length is constrained and GC-ratio is maximized [3–6, 8–10], and the other tries to locate a segment whose GC-ratio is constrained and length is maximized [1,7,11].

By assigning G, C a number one and A, T a number zero, the latter problem, which this paper mainly tackles, can be described as: finding the longest segment of a number sequence satisfying the average of the numbers in that segment is greater than or equal to a lower bound L. For example, given a DNA sequence GACGTCCCAGCAACAAA and a ratio $L = 0.8$, the longest segment whose GC ratio is at least 0.8 is the segment from position 3 to position 8, i.e., CGTCCC. This segment contains 4 C's, 1 G's, and 1 T's, thus it has a CG ratio of 0.83 with the maximum length of 6. For this problem, Allison [1] gave an algorithm which is guaranteed to run in linear-time if the input sequence is a DNA sequence and L is a rational number. But if the input sequence is a sequence of real numbers, his algorithm may run in quadratic-time in the worst case. Wang and Xu [11] provided a linear-time algorithm that can

* Corresponding author.
E-mail address: kmchao@csie.ntu.edu.tw (K.-M. Chao).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.08.006

cope with real number sequences, yet their algorithm contains a preprocessing stage. In this paper, we give an alternative linear-time algorithm that runs in an online manner, meaning that there is no preprocessing and the algorithm "interactively" outputs the best segment right after obtaining the next new number. This feature is important for tasks coping with data stream inputs, or for applications requiring real-time responses.

As was mentioned in [1,11], finding the longest segment whose average is at least L is equivalent to finding the longest segment whose sum is non-negative. Therefore, the first part of the paper deals with a more general problem having an arbitrary sum lower bound. As for the second part, we consider another interesting problem opposite to the longest segment problem. Specifically, we wish to locate the "shortest" segment satisfying a sum or an average constraint. We show that the shortest segment problem can be solved based on the same framework as the longest segment problem, but differs in that the maintenance of the data structure is more complicated.

2. Preliminaries

Given a nonempty sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$ of real numbers, and a lower bound L, let $A(i, j)$ denote the segment $\langle a_i, \ldots, a_j\rangle$ of A, and let $S(i, j)$ denote the sum of $A(i, j)$, defined as $S(i, j) = \sum_{i\leqslant k\leqslant j} a_k$ for $i \leqslant j$. The average of $A(x, y)$ is $S(x, y)/(y - x + 1)$. There are four problems of concern in this paper:

Problem 1. Locating the longest segment $A(x, y)$ of A satisfying $S(x, y) \geqslant L$.

Problem 2. Locating the longest segment $A(x, y)$ of A satisfying $\frac{S(x,y)}{(y-x+1)} \geqslant L$.

Problem 3. Locating the shortest segment $A(x, y)$ of A satisfying $S(x, y) \geqslant L$.

Problem 4. Locating the shortest segment $A(x, y)$ of A satisfying $\frac{S(x,y)}{(y-x+1)} \geqslant L$.

Lemma 1. *The average-constrained segment problems, i.e., Problems* 2 *and* 4*, can be reduced to a special case of their corresponding sum-constrained segment problems, i.e., Problems* 1 *and* 3 *with* $L = 0$.[1]

[1] In fact, Problem 4 can be solved by simply finding $a_i \geqslant L$ for $i \in [1, n]$.

Proof. Since

$$\frac{S(x, y)}{(y - x + 1)} = \sum_{x\leqslant k\leqslant y} \frac{a_k}{(y - x + 1)} \geqslant L$$

$$\Leftrightarrow \quad \sum_{x\leqslant k\leqslant y} (a_k - L) \geqslant 0,$$

it suffices to find the longest and shortest segment with a non-negative sum in a new sequence $B = \langle a_1 - L, a_2 - L, \ldots, a_n - L\rangle$. □

Since sequence B can be computed "on the fly", the next two sections focus on giving online linear-time algorithms for Problems 1 and 3. The main result is stated as follows.

Theorem 2. *There exist online linear-time algorithms for Problems* 1–4.

For convenience in later proofs, we adopt the following notations. Let $[i, j]$, $[i, j)$, and $(i, j]$ denote the sets $\{i, i + 1, \ldots, j\}$, $\{i, i + 1, \ldots, j - 1\}$, and $\{i + 1, i + 2, \ldots, j\}$ for $i \leqslant j$, respectively. Let $C[i]$ denote the cumulative sum of A, defined as $C[0] = 0$ and $C[i] = \sum_{1\leqslant k\leqslant i} a_k$ for $1 \leqslant i \leqslant n$. It is easy to see that $S(i, j) = C[j] - C[i - 1]$ for $1 \leqslant i \leqslant j \leqslant n$.

3. Locating the longest segment satisfying a sum lower bound

In order to achieve the online property, we adopt a dynamic programming approach. Suppose, at the $(i - 1)$th iteration, the longest segment of $A(1, i - 1)$ with sum at least L has been computed as $A(x, y)$. The goal is to locate, at the ith iteration, the longest segment of $A(1, i)$ with sum at least L. It suffices to find the smallest index $k \in [0, i - y + x - 1)$ satisfying $S(k + 1, i) = C[i] - C[k] \geqslant L$. If such k exists, then $A(k + 1, i)$ is the longest segment of $A(1, i)$. Otherwise, $A(x, y)$ remains the answer. Instead of searching for k by brute force, the algorithm proceeds in the "safest" way to avoid unnecessary search steps.

Definition 1. For each index $i \in [1, n]$, we define inductively an ordered list of the "safest" indices $I_{i,1}, I_{i,2}, \ldots, I_{i,\mu(i)}$, where $\mu(i)$ denotes the size of the list: $I_{i,0} = i - y + x - 1$ and $I_{i,j+1} = \min\{k \mid k \in [0, I_{i,j})$ and $C[k] \leqslant C[l]\ \forall l \in [0, I_{i,j})\}$ for $j \geqslant 1$ and $I_{i,j} > 0$. Notice that, the last index of the list is index 0, i.e., $I_{i,\mu(i)} = 0$.

We say $I_{i,j}$ is the safest index, in the sense that if $A(I_{i,j} + 1, i)$ does not satisfy the sum lower bound,

Algorithm LONGEST_SEGMENT

Input: A nonempty array of n real numbers $A[1\ldots n]$ and a lower bound L.
Output: The start and end index of the longest segment of A with sum at least L.
$C[0\ldots n]$ and $M[0\ldots n]$ are arrays of size $n+1$, as defined in the context.

```
1  M[0] ← C[0] ← 0; x ← y ← 0;
2  for i ← 1 to n do
3     C[i] ← C[i − 1] + A[i];
4     if C[i − 1] < C[M[i − 1]] then M[i] ← i − 1 else M[i] = M[i − 1];
5     k ← i − y + x − 1;
6     while k > 0 do
7        if C[i] − C[M[k]] ⩾ L then k ← M[k] else break;
8        x ← k + 1; y ← i;
9     end while
10    OUTPUT A(x, y);
11 end for
```

Fig. 1. Algorithm for finding the longest segment with a sum lower bound. Notice that, at each iteration, $A(x, y)$ represents the answer of the prefix sequence read so far, meaning that the algorithm runs in an online manner.

neither does $A(l+1, i)$ for all $l \in [0, I_{i,j-1})$. This is because $C[I_{i,j}]$ has the minimum cumulative sum of $C[l]$ for all $l \in [0, I_{i,j-1})$. The list of indices defined above has the following property.

Lemma 3. *The cumulative sums of the safest indices increase strictly, i.e.,* $C[I_{i,1}] < C[I_{i,2}] < \cdots < C[I_{i,\mu(i)}]$, *where* $I_{i,1} > I_{i,2} > \cdots > I_{i,\mu(i)}$.

Proof. For two consecutive "safest" indices $I_{i,j}$ and $I_{i,j+1}$, $j \geqslant 1$, since by definition $C[I_{i,j}] < C[l]$ for all $l \in [0, I_{i,j-1})$ and $I_{i,j+1} \in [0, I_{i,j}) \subset [0, I_{i,j-1})$, we have $C[I_{i,j}] < C[I_{i,j+1}]$ with $I_{i,j} > I_{i,j+1}$. □

The main role of the "safest" index is to inform, without the need of examining the whole interval, the algorithm when to stop. The algorithm retrieves the first "safest" index $I_{i,1}$ and checks if $C[i] - C[I_{i,1}] \geqslant L$. If not, we have the immediate conclusion that no such k exists since $C[I_{i,1}]$ is the minimum value of $C[l]$ for $l \in [0, i - y + x - 1)$. Otherwise, the algorithm retrieves the next "safest" index $I_{i,2}$ and checks if $C[i] - C[I_{i,2}] \geqslant L$. Continuing in this manner, the algorithm retrieves $I_{i,1}, I_{i,2}, \ldots, I_{i,\mu(i)}$ one by one until it finds that $C[i] - C[I_{i,j}] \geqslant L$ and $C[i] - C[I_{i,j+1}] < L$ for some $j \in [1, \mu(i)]$, or that $C[i] - C[I_{i,l}] \geqslant L$ for all $l \in [1, \mu(i)]$. In the latter case, $A(I_{i,\mu(i)} + 1, i) = A(1, i)$ is clearly the longest segment of $A(1, i)$ with sum at least L. In the former case, $C[i] - C[I_{i,j+1}] < L$ implies that $S(l+1, i) = C[i] - C[l] < L$ for all $l \in [0, I_{i,j})$. Hence, $A(I_{i,j} + 1, i)$ is the longest segment of $A(1, i)$ with sum at least L. Now, the challenge lies in whether or not we can maintain a data structure on the fly that helps us retrieve, at the ith iteration, $I_{i,1}, I_{i,2}, \ldots, I_{i,\mu(i)}$ one by one. An auxiliary array M is defined as: $M[i] = \min\{k \mid k \in [0, i)$ and $C[k] \leqslant C[l]$ $\forall l \in [0, i)\}$. It is not hard to verify that

$$I_{i,1} = M[i - y + x - 1],$$
$$I_{i,2} = M\big[M[i - y + x - 1]\big], \quad \ldots,$$
$$I_{i,\mu(i)} = \underbrace{M[\ldots M[M[i - y + x - 1]]\ldots]}_{\mu(i) \text{ times}}.$$

Array M can be computed on the fly by the following recurrence:

$$M[i] = \begin{cases} 0 & \text{if } i = 0; \\ i - 1 & \text{if } i \geqslant 1 \text{ and} \\ & \quad C[i-1] < C[M[i-1]]; \\ M[i-1] & \text{if } i \geqslant 1 \text{ and} \\ & \quad C[i-1] \geqslant C[M[i-1]]. \end{cases}$$

The algorithm for finding the longest segment satisfying a sum lower bound is given in Fig. 1. An intuitive way to show that the LONGEST_SEGMENT algorithm runs in $O(n)$ time is to observe that the cost of the steps for finding a longer segment at each iteration is equivalent to the length of the segment grown throughout the execution. Since the length of the longest segment at the end is less than or equal to n, the cost is bounded by $O(n)$. A more formal proof is given in the following.

Theorem 4. *The* LONGEST_SEGMENT *algorithm runs in* $O(n)$ *time.*

Proof. The total number of operations of LONGEST_SEGMENT is clearly bounded by $O(n)$ except for the while-loop body of lines 6–9. By verifying that

(1) $y - x = 0$ holds initially,
(2) $y \geqslant x$ holds throughout the execution,
(3) the value of $y - x$ never decreases, and
(4) $y - x + 1 \leqslant n$ at the end of the execution,

we conclude that the overall time required by the loop is $O(n)$. □

4. Locating the shortest segment satisfying a sum lower bound

Suppose, at the $(i-1)$th iteration, the shortest segment of $A(1, i-1)$ with sum at least L has been computed as $A(x, y)$. Following a paradigm similar to the last section, we define another list of the "safest" indices, only this time the "safest" indices move towards the other direction since our main goal now is to find a shorter segment.

Definition 2. For each index $i \in [1, n]$, we define inductively an ordered list of the "safest" indices $I^*_{i,1}, I^*_{i,2}, \ldots, I^*_{i,\mu(i)}$, where $\mu(i)$ denotes the size of the list: $I^*_{i,0} = i - y + x - 1$ and $I^*_{i,j+1} = \max\{k \mid k \in (I^*_{i,j}, i-1]$ and $C[k] \leqslant C[l]\ \forall l \in (I^*_{i,j}, i-1]\}$ for $j \geqslant 1$ and $I^*_{i,j} < i-1$. Notice that, the last index of the list is index $i-1$, i.e., $I^*_{i,\mu(i)} = i-1$.

Lemma 5. *The cumulative sums of the safest indices increase strictly, i.e.,* $C[I^*_{i,1}] < C[I^*_{i,2}] < \cdots < C[I^*_{i,\mu(i)}]$, *where* $I^*_{i,1} < I^*_{i,2} < \cdots < I^*_{i,\mu(i)}$.

Proof. Similar to Lemma 3. □

The algorithm proceeds along the list of safest indices $I^*_{i,1}, I^*_{i,2}, \ldots, I^*_{i,\mu(i)}$ until it finds that $C[i] - C[I^*_{i,j}] \geqslant L$ and $C[i] - C[I^*_{i,j+1}] < L$ for some $j \in [1, \mu(i)]$, or that $C[i] - C[I^*_{i,l}] \geqslant L$ for all $l \in [1, \mu(i)]$. However, the same challenge occurs, i.e., whether or not we can maintain a data structure on the fly that helps us retrieve, at the ith iteration, $I^*_{i,1}, I^*_{i,2}, \ldots, I^*_{i,\mu(i)}$ one by one. We use a *linked list* to implement the data structure. The maintenance of the list is described below. Suppose, at the $(i-1)$th iteration, we have the complete list $I^*_{i-1,1}, I^*_{i-1,2}, \ldots, I^*_{i-1,\mu(i-1)}$. Then, the list for the ith iteration, i.e., $I^*_{i,1}, I^*_{i,2}, \ldots, I^*_{i,\mu(i)}$, can be obtained by the following procedure:

Step 1. The list is searched from left to right for the largest $p \in [1, \mu(i-1)]$ satisfying $I^*_{i-1,p} \leqslant I^*_{i,0} = i - y + x - 1$. If such p exists, then delete $I^*_{i-1,1}, I^*_{i-1,2}, \ldots, I^*_{i-1,p}$ from the list; Otherwise, let $p = 0$.

Step 2. The remaining list is then searched from right to left for the smallest $q \in (p, \mu(i-1)]$ satisfying $C[i-1] \leqslant C[I^*_{i-1,q}]$. If such q exists, then delete $I^*_{i-1,q}, I^*_{i-1,q+1}, \ldots, I^*_{i-1,\mu(i-1)}$ from the list; Otherwise, let $q = \mu(i-1) + 1$.

Step 3. Insert $i-1$ to the end of the list.

Lemma 6. *After the above procedure, the resulting list* $I^*_{i-1,p+1}, I^*_{i-1,p+2}, \ldots, I^*_{i-1,q-1}, i-1$ *is equivalent to the list* $I^*_{i,1}, I^*_{i,2}, \ldots, I^*_{i,\mu(i)}$ *needed for the* i*th iteration.*

Proof. After Steps 1 and 2, the cumulative sums of the indices in the remaining list are less than $C[i-1]$ since $C[I^*_{i-1,p+1}] < C[I^*_{i-1,p+2}] < \cdots < C[I^*_{i-1,q-1}] < C[i-1]$ by Lemma 5. Therefore, the expression of $I^*_{i-1,j}$, for all $j \in [p+1, q-1]$, can be rewritten as: $I^*_{i-1,j} = \max\{k \mid k \in (I^*_{i-1,j-1}, i-1]$ and $C[k] \leqslant C[l]\ \forall l \in (I^*_{i,j-1}, i-1]\}$. Since $I^*_{i-1,p} \leqslant I^*_{i,0}$, we have $(I^*_{i,0}, i-1] \subseteq (I^*_{i-1,p}, i-1]$ and can further rewrite the expression of $I^*_{i-1,p+1}$ as: $I^*_{i-1,p+1} = \max\{k \mid k \in (I^*_{i,0}, i-1]$ and $C[k] \leqslant C[l]\ \forall l \in (I^*_{i,0}, i-1]\}$. It follows that $I^*_{i-1,p+1} = I^*_{i,1}$, which yields $I^*_{i-1,p+2} = I^*_{i,2}$, leading to $I^*_{i-1,p+3} = I^*_{i,3}$, etc. □

The algorithm for finding the shortest segment satisfying a sum lower bound is given in Fig. 2. Notice that, some update steps of Step 1 is incorporated into the while-loop body (see line 7) due to the fact that the value of $i - y + x - 1$ for the next iteration increases with the increasing k.

Theorem 7. *The* SHORTEST_SEGMENT *algorithm runs in* $O(n)$ *time.*

Proof. It is clear that the update steps of the list dominate the time complexity of the algorithm. By observing that each update step can be viewed as an action of each index getting into and out of the list, we have the total cost for updating the list is bounded by $O(n)$. Since each index gets into and out of the list at most once, the total time required is $O(n)$. □

5. Concluding remarks

In this paper, we present two linear-time algorithms for finding the longest and shortest segments $A(x, y)$ of A satisfying $S(x, y) \geqslant L$. The sum constraint replaced by an upper-bound U, specifying $S(x, y) \leqslant U$, leads to an equivalent problem since $\sum_{x \leqslant k \leqslant y} a_k \leqslant U \Leftrightarrow \sum_{x \leqslant k \leqslant y} -a_k \geqslant -U$. However, both sides of the sum constraints cannot hold at the same time. It remains

Algorithm SHORTEST_SEGMENT

Input: A nonempty array of n real numbers $A[1 \ldots n]$ and a lower bound L.
Output: The start and end index of the shortest segment of A with sum at least L.
The list I^* can be implemented as a linked list, initially assigned empty.
$C[0 \ldots n]$ is an array of size $n+1$, as defined in the context.

```
1  C[0] ← 0; x ← y ← k ← 0; I* ← φ;
2  for i ← 1 to n do
3     C[i] ← C[i − 1] + A[i];
4     UPDATE(I*, i, x, y);
5     if y > 0 then k ← i − y + x − 1;
6     while k < i − 1 do
7        if C[i] − C[HEAD(I*)] ⩾ L then k ← DELETE_HEAD(I*) else break;
8        x ← k + 1; y ← i;
9     end while
10    OUTPUT A(x, y);
11 end for
```

subroutine UPDATE(I^*, i, x, y)

```
1  while y > 0 and !EMPTY(I*) and HEAD(I*) ⩽ i − y + x − 1 do
2     DELETE_HEAD(I*);
3  end while
4  while !EMPTY(I*) and C[TAIL(I*)] ⩾ C[i − 1] do DELETE_TAIL(I*);
5  INSERT_TAIL(I*, i − 1);
```

Fig. 2. Algorithm for finding the shortest segment with sum lower bound. EMPTY(I^*) returns *true* if list I^* is empty, otherwise it returns *false*. HEAD(I^*) returns the value of the first element in list I^*. TAIL(I^*) returns the value of the last element in list I^*. DELETE_HEAD(I^*) deletes the first element in list I^* and returns its value. DELETE_TAIL(I^*) deletes the last element in list I^* and returns its value. INSERT_TAIL(I^*, x) inserts value x into the end of list I^*. Notice that, at each iteration, $A(x, y)$ represents the answer of the prefix sequence read so far, meaning that the algorithm runs in an online manner.

open if there exist linear-time algorithms for locating longest and shortest segments $A(x, y)$ of A satisfying $L \leqslant S(x, y) \leqslant U$.

Acknowledgements

We thank William C. Vocke and the referees for their helpful comments. Kuan-Yu Chen and Kun-Mao Chao were supported in part by an NSC grant 93-2213-E-002-029 from the National Science Council, Taiwan.

References

[1] L. Allison, Longest biased interval and longest non-negative sum interval, Bioinformatics 19 (2003) 1294–1295.
[2] J. Bentley, Programming Pearls—algorithm design techniques, Comm. ACM 27 (1984) 865–871.
[3] K.-Y. Chen, K.-M. Chao, On the range maximum-sum segment query problem, in: ISAAC, in: Lecture Notes in Comput. Sci., vol. 3341, Springer, Berlin, 2004, pp. 294–305.
[4] K. Chung, H.-I. Lu, An optimal algorithm for the maximum-density segment problem, SIAM J. Comput. 34 (2) (2004) 373–387.
[5] T.-H. Fan, S. Lee, H.-I. Lu, T.-S. Tsou, T.-C. Wang, A. Yao, An optimal algorithm for maximum-sum segment and its application in bioinformatics, in: CIAA, in: Lecture Notes in Comput. Sci., vol. 2759, Springer, Berlin, 2003, pp. 251–257.
[6] M.H. Goldwasser, M.-Y. Kao, H.-I. Lu, Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications, J. Comput. System Sci. 70 (2) (2005).
[7] X. Huang, An algorithm for identifying regions of a DNA sequence that satisfy a content requirement, Comput. Appl. Biosci. 10 (1994) 219–225.
[8] Y.-L. Lin, X. Huang, T. Jiang, K.-M. Chao, MAVG: Locating non-overlapping maximum average segments in a given sequence, Bioinformatics 19 (2003) 151–152.
[9] Y.-L. Lin, T. Jiang, K.-M. Chao, Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis, J. Comput. System Sci. 65 (2002) 570–586.
[10] W.L. Ruzzo, M. Tompa, A linear-time algorithm for finding all maximal scoring subsequences, in: Proc. 7th Internat. Conf. Intelligent Systems for Molecular Biology, Heidelberg, Germany, 1999, pp. 234–241.
[11] L. Wang, Y. Xu, SEGID: Identifying interesting segments in (multiple) sequence alignments, Bioinformatics 19 (2003) 297–298.