

# Algorithms for Problems on Maximum Density Segment

Md. Shafiul Alam and Asish Mukhopadhyay<sup>(✉)</sup>

School of Computer Science, University of Windsor,  
Windsor, ON N9B3P4, Canada  
{alam9, asishm}@uwindsor.ca

**Abstract.** Let  $A$  be a sequence of  $n$  ordered pairs of real numbers  $(a_i, l_i)$  ( $i = 1, \dots, n$ ) with  $l_i > 0$ , and  $L$  and  $U$  be two positive real numbers with  $0 < L \leq U$ . A segment, denoted by  $A[i, j]$ ,  $1 \leq i \leq j \leq n$ , of  $A$  is a consecutive subsequence of  $A$  between the indices  $i$  and  $j$  ( $i$  and  $j$  included). The length  $l[i, j]$ , sum  $s[i, j]$  and density  $d[i, j]$  of a segment  $A[i, j]$  are  $l[i, j] = \sum_{t=i}^j l_t$ ,  $s[i, j] = \sum_{t=i}^j a_t$  and  $d[i, j] = \frac{s[i, j]}{l[i, j]}$  respectively. A segment  $A[i, j]$  is feasible if  $L \leq l[i, j] \leq U$ . The length-constrained maximum density segment problem is to find a feasible segment of maximum density. We present a simple geometric algorithm for this problem for the uniform length case ( $l_i = 1$  for all  $i$ ), with time and space complexities in  $O(n)$  and  $O(U - L + 1)$  respectively. The  $k$  length-constrained maximum density segments problem is to find the  $k$  most dense length-constrained segments. For the uniform length case, we propose an algorithm for this problem with time complexity in  $O(\min\{nk, n \lg(U - L + 1) + k \lg^2(U - L + 2), n(U - L + 1)\})$ .

**Keywords:** Biomolecular sequence analysis · Maximum density segment · Computational geometry · Slope selection · Data structure

## 1 Introduction

Let  $A$  be a sequence  $(a_i, l_i)$  ( $i = 1, \dots, n$ ) of  $n$  ordered pairs of real numbers  $a_i$ , called values, and  $l_i > 0$ , called lengths, and  $L, U$  two positive real numbers with  $0 < L \leq U$ . A segment of  $A$ , denoted by  $A[i, j]$ ,  $1 \leq i \leq j \leq n$ , is a consecutive subsequence of  $A$  between the indices  $i$  and  $j$ , both inclusive. The length  $l[i, j]$ , sum  $s[i, j]$  and density  $d[i, j]$  of a segment  $A[i, j]$  are  $l[i, j] = \sum_{t=i}^j l_t$ ,  $s[i, j] = \sum_{t=i}^j a_t$  and  $d[i, j] = \frac{s[i, j]}{l[i, j]}$  respectively. A feasible segment of  $A$  is a segment  $A[i, j]$  such that  $L \leq l[i, j] \leq U$ . In this paper we study the following problems.

**Problem 1.** *The length-constrained maximum density segment problem is to find a feasible segment  $A[i, j]$  of maximum density  $d[i, j]$ .*

---

A. Mukhopadhyay—Research supported by an NSERC discovery grant awarded to this author.

When the lengths are uniform (i.e.,  $l_i = 1$ ) and  $U$  and  $L$  are arbitrary, Goldwasser *et al.* [7] gave an  $O(n)$  time algorithm. For the case of non-uniform lengths and arbitrary  $U$  and  $L$ , Goldwasser *et al.* [8] extended the right skew decomposition method of Lin *et al.* [11] to develop an  $O(n)$ -time and space algorithm. An online, combinatorial algorithm with time-complexity in  $O(n)$  and space complexity in  $O(m)$ , where  $m$  is the maximum of the number of elements in a segment of length  $U - L$ , was proposed in [4]. It also pointed out a flaw in the linearity claim of a geometry-based algorithm by Kim [9]. Lee *et al.* [10] fixed this flaw by exploiting the property of decomposability of a tangent query, and proposed a revised algorithm with time and space complexities in  $O(n)$ . In this paper, we present a simple modification of Kim's algorithm [9] that redresses the flaw using a batched mode approach, while retaining the simplicity, elegance and linearity of his geometric approach. For the uniform length case and arbitrary  $L$  and  $U$ , the time and space complexities of our algorithm are in  $O(n)$  and  $O(U - L + 1)$  respectively.

As a natural extension, we consider the  $k$  length-constrained maximum density segments problem, defined next:

**Problem 2.** *Given a positive integer  $k$  such that  $1 \leq k \leq$  total number of feasible segments, the  $k$  length-constrained maximum density segments problem is to find  $k$  feasible segments  $A[i, j]$  such that their densities  $d[i, j]$  are the  $k$  largest.*

Our proposed algorithm solves this problem for the uniform length case and arbitrary  $L$  and  $U$  in  $O(\min\{nk, n \lg(U - L + 1) + k \lg^2(U - L + 2), n(U - L + 1)\})$  time. Its space complexity is in  $O(U - L + k)$ ,  $O((U - L + 1) \lg(U - L + 2) + k)$  or  $O(k)$ , depending on the value of  $k$ .

The proposed algorithms can be extended to the non-uniform case as also to higher dimensions by reducing them to 1-dimensional problems as described in [2, 16]. These discussions are omitted for lack of space.

The ratio  $(C + G)/(A + C + G + T)$  is a measure of the GC content of a DNA-sequence, where  $A, C, G, T$  are the nucleotide bases. According to [12, 15] the compositional heterogeneity of a genomic sequence is strongly correlated to its GC content regardless of genome size. It has also been found that gene length [5], gene density [17], patterns of codon usage [14] and other properties are related to GC content. However, it is not established that the single most dense region is the only meaningful region. Other segments with high GC content might also be meaningful. Our proposed algorithms can be used to find length constrained CG-rich regions with the maximum density and  $k$  maximum density in a DNA sequence efficiently.

In Sect. 2 we describe our algorithm for the maximum density segment problem. Our algorithm for the  $k$  maximum density segments problem is presented in Sect. 3. Concluding remarks are given in Sect. 4.

## 2 SPLITHULL Algorithm for Maximum Density Segment

The prefix sums of the sequence  $A$ , defined by  $s_0 = 0$  and  $s_i = \sum_{t=1}^i a_t$  for  $1 \leq i \leq n$ , are computable in linear time. Define  $n + 1$  points in the plane thus:

$p_i = (i, s_i)$ ,  $0 \leq i \leq n$ . The density of a segment  $A[i, j]$  is then equal to slope of the line segment through the points  $p_{i-1}$  and  $p_j$ . This reduces our problem to finding a segment  $\overline{p_i p_j}$  of largest slope.

The main idea underlying the new algorithm is to consider the right end points  $p_j$  (for  $U \leq j \leq n$ ) of all feasible segments  $\overline{p_i p_j}$  in batches of a fixed size. For each  $p_j$ , instead of computing a single lower convex hull of the feasible set of left points  $p_i$ , we compute two lower convex hulls - a left one and a right one. These start at 2 adjacent points  $p_{m-1}$  and  $p_m$ ,  $j - U + 1 \leq m \leq j - L + 1$ , going left and right respectively. The right lower hulls are computed incrementally in a left-to-right (*LR*) pass for a batched set  $p_j$ , and the left hulls in a right-to-left (*RL*) pass for the same batched set. This pre-empts a dynamic convex hull update problem that arises in Kim's algorithm [9]. Note that the points  $p_j$  with  $L \leq j \leq U - 1$  can be handled in a single LR pass. The correctness of this scheme follows from the following observation:

**Observation 1.** *For a point  $p_j$ ,  $U \leq j \leq n$ , let  $G^j$  be the set of the candidate left end points  $p_i$  of all feasible segments. If  $G_1^j$  and  $G_2^j$  are any 2 subsets of  $G^j$  such that  $G^j = G_1^j \cup G_2^j$ , then*

$$\max_{p_i \in G^j} \text{slope}(\overline{p_i p_j}) = \max \left\{ \max_{p_i \in G_1^j} \text{slope}(\overline{p_i p_j}), \max_{p_i \in G_2^j} \text{slope}(\overline{p_i p_j}) \right\}.$$

We consider the right end points  $p_j$ ,  $j \geq U$ , in batches of size  $U - L + 1$ . The details of the *LR* and *RL* passes for a batch of right end points  $p_j$ ,  $j \in [k, k + U - L]$ ,  $k \geq U$ , are described below.

## 2.1 LR Pass

In this pass, we process the right end points  $p_j$ ,  $j \in [k, k + U - L]$ , left to right. For each new point  $p_j$ ,  $j \in [k, k + U - L]$ , the current Lower Convex Hull (LCH)  $H_r$  is dynamically updated by the insertion of a new point on the right of  $H_r$ . Following Kim [9], we maintain 2 parameters to aid the incremental computation: a tangent line  $l$  to the current hull  $H_r$  with the maximum slope found so far, and the point of contact  $\alpha$  of  $l$  with  $H_r$ . The line  $l$  is always represented by a pair of points and its slope is the current maximum density for this batch of points  $p_j$ .

Initially,  $H_r = \{p_{k-L}\}$ ,  $l = \overline{p_{k-L} p_k}$  and  $\alpha = p_{k-L}$ . Assume that  $H_r$ ,  $l$  and  $\alpha$  have been computed for the right end point  $p_j$ . For the next point  $p_{j+1}$ , these are updated as follows.  $H_r$  is reset to  $H_r = \text{LCH}(H_r, p_{j+1-L})$ . The updated  $H_r$  is traversed counterclockwise from  $\alpha$  (or from the newly inserted hull point  $p_{j+1-L}$ , if  $\alpha$  has been deleted from  $H_r$ ) to find the new tangent line  $l$  of maximum slope so far, and the new point of contact  $\alpha$  on  $H_r$  with the updated  $l$ . We have to consider 4 cases:

**Case 1:** Both  $p_{j+1-L}$  and  $p_{j+1}$  are above  $l$ .

$H_r$  is first updated and then traversed counterclockwise from the current  $\alpha$  to the point of contact of the tangent from  $p_{j+1}$  to  $H_r$ . This tangent line and its point of contact are set to be the new  $l$  and  $\alpha$  respectively.

**Case 2:**  $p_{j+1-L}$  is above, and  $p_{j+1}$  is on or below  $l$ .

$H_r$  is updated. However,  $\alpha$  and  $l$  remain unchanged.

**Case 3:**  $p_{j+1-L}$  is on or below  $l$ .

$H_r$  is updated. Let  $l'$  be a line through  $p_{j+1-L}$  and parallel to  $l$ . Let  $p_{j+1}$  be above  $l'$ ; reset  $l = \overline{p_{j+1-L}p_{j+1}}$  and  $\alpha = p_{j+1-L}$ .

**Case 4:**  $p_{j+1-L}$  is on or below  $l$ , and  $p_{j+1}$  is on or below  $l'$ .

$H_r$  is updated. Set  $l$  to  $l'$  and  $\alpha = p_{j+1-L}$ .

Each point in the left window  $\{p_{k-L}, p_{k+1-L}, \dots, p_{k+U-2L}\}$  is added to an  $H_r$  once, and deleted at most once from a subsequent  $H_r$ . Noting that  $\alpha$  never moves left, for a new point  $p_j$ , if  $\alpha$  remains stationary (as in Case 2 above), the cost of computation is constant and is charged to the point  $p_{j-L}$  that is added to the hull. Consider the case in which  $\alpha$  moves counterclockwise (and thus right) along an updated hull  $H_r$ . Each point on  $H_r$  is accessed at most once during the recomputation of  $\alpha$ , since it never moves left. The cost of recomputing  $\alpha$  is charged to the hull points that are passed over as we move counterclockwise on  $H_r$  from the current  $\alpha$ , and the cost of deleting the points on  $H_r$  on the left of  $\alpha$  are charged to them. Thus, each point  $p_i$  in the left window is charged at most 3 times: 2 times for insertion into and deletion from  $H_r$  and once for being passed over by  $\alpha$ . So, the cost for this pass is linear in the number of  $p_j$ 's considered.

## 2.2 RL Pass

In this pass, we process the right end points  $p_j$ ,  $j \in [k, k+U-L-1]$ , right to left. For each new point  $p_j$ ,  $j \in [k, k+U-L-1]$ , the current Lower Convex Hull (LCH)  $H_l$  is dynamically updated by the insertion of a new point  $p_{j-U}$  on the left of  $H_r$ .

As in the LR pass, we maintain a tangent line  $l$  and the point of contact  $\alpha$  of  $l$  with the current hull  $H_l$  to aid the incremental computation.

Initially,  $H_l = \{p_{k-L-1}\}$ ,  $l = \overline{p_{k-L-1}p_{k+U-L-1}}$  and  $\alpha = p_{k-L-1}$ . Assume that  $H_l$ ,  $l$  and  $\alpha$  have been updated for the right end point  $p_j$ . For the next right end point  $p_{j-1}$ , these are updated as follows.  $H_l$  is updated by inserting the point  $p_{j-1-U}$  on the left so that  $H_l = LCH(p_{j-1-U}, H_l)$ . The updated  $H_l$  is traversed counterclockwise from  $\alpha$  (or from the newly inserted hull point  $p_{j-1-U}$  - if  $\alpha$  is deleted from  $H_l$ ) to find the new tangent line  $l$  having the maximum slope found so far, and the new point of contact  $\alpha$  on  $H_l$  with the updated  $l$ . Again, there are 4 cases to consider:

**Case 1:**  $p_{j-1-U}$  is on or above  $l$ , and  $p_{j-1}$  is above  $l$ .

$H_l$  is updated. We traverse  $H_l$  counterclockwise from  $\alpha$  to find a tangent to it from  $p_{j-1}$ . We reset  $l$  to this tangent line and  $\alpha$  to the point of contact between updated  $l$  and  $H_l$ .

**Case 2:**  $p_{j-1-U}$  is on or above  $l$ , and  $p_{j-1}$  is on or below  $l$ .

$H_l$  is updated. However,  $\alpha$  and  $l$  remain unchanged.

**Case 3:**  $p_{j-1-U}$  is below  $l$ .

$H_l$  is updated. Let  $l'$  be a line through  $p_{j-1-U}$  and parallel to  $l$ . Let  $p_{j-1}$  be above  $l'$ .

There will be only one point, viz.,  $p_{j-1-U}$ , on the updated  $H_l$  that is on the left side of  $\alpha$ . We traverse the updated  $H_l$  from  $p_{j-1-U}$  counterclockwise from  $\alpha$  to the point of contact of the tangent from  $p_{j-1}$  to the new  $H_l$ , while  $\alpha$  and  $l$  are updated to the new tangent and the point of contact respectively. In this case, on the left of  $\alpha$  at most one point, viz., the newly added point  $p_{j-1-U}$ , is checked to find  $\alpha$ . Consequently,  $\alpha$  can move left by at most one point.

**Case 4:**  $p_{j-1-U}$  is below  $l$ , and  $p_{j-1}$  is on or below  $l'$ .

$H_l$  is updated as in Case 3. We reset  $l$  to  $l'$  and  $\alpha$  to  $p_{j-1-U}$ .

Time complexity analysis for this pass is exactly the same as that for the LR pass, except that for a new point  $p_j$ ,  $\alpha$  may move clockwise on  $H_l$  exactly by one position. If it does move clockwise, then it moves to  $p_{j-U}$ . This cost is charged to the new point  $p_{j-U}$  in the left window. Thus, each point  $p_i$  in the left window is charged at most 4 times: 2 times for insertion into and deletion from  $H_l$ , once when  $\alpha$  moves clockwise to it and once when  $\alpha$  passes over it.

We note that once  $\alpha$  moves clockwise and passes over a point  $p_i$  on  $H_l$ , it never moves back to that point again, or to any point lying on its left in the current  $H_l$ . Consequently, those points cannot be in contention for  $\alpha$  anymore.

### 2.3 Analysis

Each batch of  $U - L + 1$  points in the left index window is considered at most twice by SPLITHULL algorithm: once for an LR pass of a batch of  $U - L + 1$  right end points and once for an RL pass of a batch of  $U - L$  right end points. As discussed above, the cost charged to each of these left end points is constant for each pass. Each of the right end points is accessed at most twice and that cost is charged to the respective point. Consequently, the time complexity is in  $O(n)$ . Thus we have the following theorem:

**Theorem 1.** *The SPLITHULL algorithm, described above, solves the length-constrained maximum density segment problem for the uniform length case with arbitrary  $L$  and  $U$  in  $O(n)$  time and  $O(U - L + 1)$  working space.*

## 3 $k$ Maximum Density Segments

Three different algorithms are proposed, depending on the size of  $k$  relative to the parameters  $n, U$  and  $L$ .

### 3.1 Small $k$

By small  $k$  we mean  $k = f(n_0)$  for some  $n_0$  and some  $f(n) = O(\lg(U - L + 1))$ . We propose an algorithm that is better in terms of asymptotic time complexity for such small  $k$ . As before, the points are processed in batches of  $U - L + 1$  right end points. Let  $X$  be the left end points of feasible segments whose right

end-points belong to a current batch and  $D$  be a candidate set of  $k$  maximum density segments. For each batch,  $D$  is updated as follows. First, a maximum density segment with left end point  $x \in X$  is found by using the LR and RL passes of the SPLITHULL algorithm. If the density of this segment is less than the minimum density  $d_0$  for all the segments in  $D$ , then we skip to the next batch. Otherwise, all the feasible segments with left end point  $x$  are inserted into  $D$ . From  $D$ ,  $k$  maximum density segments are selected using a linear time selection algorithm, and  $D$  is updated with them. Then  $x$  is deleted from  $X$ , and the above steps are repeated with the updated  $X$ . We iterate at most  $k$  times. The number of iterations is maximized if the density of a maximum density segment in each iteration is greater than the minimum density of all the segments in the current  $D$ .

Clearly, each iteration costs  $O(U - L + 1)$  time. There are at most  $k$  iterations in a pass. Total time for a pass is in  $O(k(U - L + 1))$ . The total cost per left end point is in  $O(k)$ . Thus, we have the following theorem:

**Theorem 2.** *The above algorithm solves the  $k$  length-constrained maximum density segments problem for the uniform length case and arbitrary  $L$  and  $U$  in  $O(kn)$  time and  $O(U - L + 1)$  working space.*

### 3.2 Medium $k$

By medium  $k$  we mean  $k = f(n_1)$  for some  $n_1$  and some  $f(n) = \omega(\lg(U - L + 1))$ , and  $k = g(n_2)$  for some  $n_2$  and some  $g(n) = o(n(U - L + 1))$ . We propose an algorithm which is more efficient for such values of  $k$ . For each batch of  $U - L + 1$  right end points, we make both LR and RL-passes to consider all the feasible segments whose right points are in this batch. Let  $[b, b + U - L]$  be the index window of the current batch of right end points. In the LR-pass, the left end points of all the feasible segments are in the index window  $[b - U + 1, b - L + 1]$ , while in the RL-pass they are in the window  $[b - L + 2, b + U - 2L + 1]$ . Thus the LR and RL-passes consider all feasible segments with right end points in the index window  $[b, b + U - L]$ . As both the passes are very similar, we describe only the LR-pass for the current batch.

**Grouping the Feasible Segments:** We outline a mechanism for grouping feasible segments that aid their efficient processing. A group of feasible segments is represented by a pair  $I_l \times I_r$  where  $I_l$  and  $I_r$  are the index windows of  $|I_l|$  consecutive left end points and  $|I_r|$  consecutive right end points respectively of  $|I_l \times I_r|$  feasible segments. Henceforth, we shall call  $I_l$  the left index window and  $I_r$  the right index window. We do not construct the groups explicitly; instead, identify them by pairs of index windows. The processing of these groups are described next.

First, with the single right end point  $p_b$ , we make a group of all feasible segments with the single left end point  $p_{b-U+1}$  and represent it by the index pair  $[b - U + 1, b - U + 1] \times [b, b]$ . Next, we make the following 2 groups of feasible

segments:  $[b - U + 2, b - U + 3] \times [b, b + 1]$  and  $[b - U + 3, b - U + 3] \times [b + 2, b + 2]$ . This completes the scan of 2 more left end points  $p_i, i \in [b - U + 2, b - U + 3]$  and groups all the feasible segments with 3 consecutive left end points starting from  $p_{b-U+1}$  and 3 consecutive right end points starting from  $p_b$ .

Next, we make the following 4 groups of feasible segments:  $[b - U + 4, b - U + 7] \times [b, b + 3]$ ,  $[b - U + 5, b - U + 5] \times [b + 4, b + 4]$ ,  $[b - U + 6, b - U + 7] \times [b + 4, b + 5]$  and  $[b - U + 7, b - U + 7] \times [b + 6, b + 6]$ . This completes scanning 4 more left end points  $p_i, i \in [b - U + 4, b - U + 7]$ . After this scan all the feasible segments with consecutive 7 left end points starting from  $p_{b-U+1}$  and consecutive 7 right end points starting from  $p_b$  have been completely grouped.

This is a recursive pattern, and at the end of the  $i$ -th step we have grouped all the combinations of segments generated by  $2^i - 1$  consecutive right end points and the same number of consecutive left end points such that they are feasible. We note that for each of the groups of feasible segments generated by the above algorithm, the left and right index windows have the same length, and that the length of the index windows are in powers of 2. For simplicity, let us assume that  $U - L + 1 = 2^s - 1$  for some positive integer  $s$ . After  $s$ , steps all the feasible segments with consecutive  $2^s - 1$  left end points starting from  $p_{b-U+1}$  and ending at  $p_{b-L+1}$ , and the same number of consecutive right end points starting from  $p_b$  and ending at  $p_{b+U-L}$  have been completely grouped. Thus, all the feasible segments corresponding to the LR-pass have been completely grouped. Note that the  $G_i$ s are pairwise disjoint.

**Lemma 1.** *The above algorithm constructs groups of feasible segments  $G_i, i = 1, \dots, U - L + 1$  such that  $\cup_{i=1}^{U-L+1} G_i$  is the set of all feasible segments in the LR-pass and all the  $G_i$ s are mutually disjoint.*

The two properties of  $G_i$ s mentioned in Lemma 1 ensures that the segments in each group can be processed independently of the other groups and that we need to process the  $G_i$ s only.

In the above grouping procedure we do not consider the segments, but their indices. It will take constant time to construct a group. For  $2^s - 1$  right end points,  $2^s - 1$  groups of feasible segments will be created.

**Lemma 2.** *In both the LR and RL-passes, groups  $G_i$  can be created in  $O(U - L + 1)$  time.*

**Organizing the Points:** Now we describe the processing of a group of feasible segments. Let  $G = I_l \times I_r$  be a group of feasible segments where  $|I_l| = |I_r| = m = 2^t$  for some positive integer  $t$ . Then  $|G| = |I_l| \times |I_r| = 2^{2t}$ . Let  $Q$  and  $R$  be the sets of points having index windows  $I_l$  and  $I_r$  respectively. Then  $|Q| = |R| = m = 2^t$ .

First, we organize the points in  $Q$ . We use Overmars and Leeuwen [13] algorithm, with a simple modification, to construct the *lch* (lower convex hull) of  $Q$  by composition. By construction of the geometric problem all the points are already sorted by  $x$ -coordinate and vertically separated. In fact, all the  $n$  input points are separated by unit distance in  $x$ -coordinate, and consequently all the points of  $Q$  are separated by unit distance in  $x$ -coordinate.

The algorithm iteratively constructs the convex hull as follows. In the first iteration, construct  $2^{t-1}$  *lchs* of 2 consecutive points each. In the 2nd iteration, construct  $2^{t-2}$  *lchs* of  $2^2$  consecutive points each by composing pairs of adjacent constituent *lchs* of 2 consecutive points each. In the 3rd iteration, construct  $2^{t-3}$  *lchs* of  $2^3$  consecutive points each by composing pairs of adjacent constituent *lchs* of  $2^2$  consecutive points each. Continue this for  $t$  iterations. The information of all of these constituent *lchs* as well as the *lch* of  $Q$  is stored in a balanced binary search tree, say  $C$ . This tree will be called LCH Tree. Its leaf nodes represent the points of  $Q$ . Direct parents of the leaves represent the next higher level of *lchs*. Direct parents of these parents represent the next higher level of *lchs* and so on. The root represent the *lch* of  $Q$ . We denote the *lch* of  $Q$  by  $H^1$  and a *lch* at  $i$ -th level and  $j$ -th position from left by  $H_j^i$ .

In Overmars and Leeuwen [13] algorithm each node  $u$  of  $C$  is associated with a concatenable queue [1] to store the information about the *lch* of all the leaves in the subtree of  $u$ . Thus, we have the following Lemma due to Overmars and Leeuwen [13] (Proposition 4.1):

**Lemma 3.** *The tree  $C$  for a set of  $m$  points can be constructed in  $O(m)$  time.*

*Proof.* See proof of Proposition 4.1 of Overmars and Leeuwen [13].

Let us find the time to construct all the LCH Trees ( $C$ s). Let  $U - L + 1 = 2^s - 1$ . There will be  $\frac{U-L+2}{2^{i+1}}$  groups of size  $2^i - 1$ . Total time for the construction of all the  $C$ s for the LR-pass is

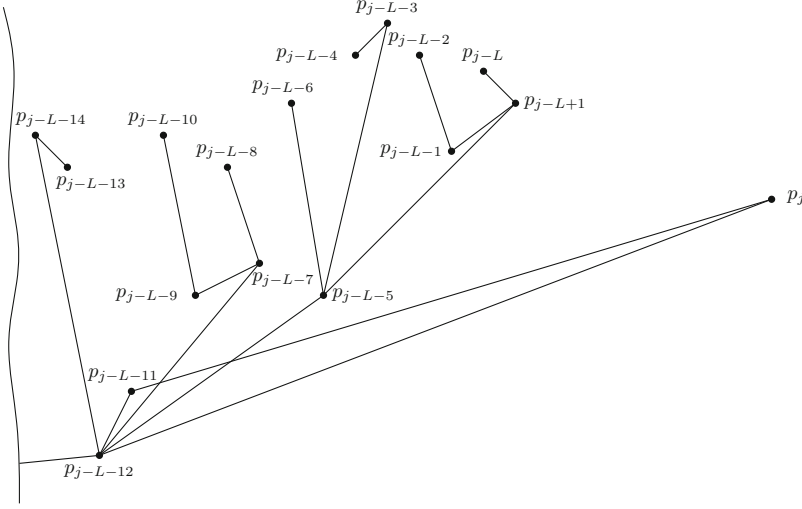
$$\sum_{i=0}^{s-1} \frac{U - L + 2}{2^{i+1}} O(2^i) = O((U - L + 1) \lg(U - L + 1))$$

**Lemma 4.** *All the LCH trees ( $C$ s) for LR and RL passes can be constructed in  $O(n \lg(U - L + 1))$  time.*

**Searching:** Let us assume that the LCH Tree  $C$  of all levels of *lchs* of  $Q$  have already been constructed. Now we describe the search for maximum density segments. For a right end point  $p_j \in R$ , the maximum density segment is found by drawing tangent to the top most level *lch*  $H^1$  (Fig. 1). For simplicity, we assume that there is only one point of contact always. But this assumption is not essential for the method being described in the following. For if there are multiple points of contact, say  $s$  number of points of contact  $p_i, p_{i'}, p_{i''}, \dots$  etc., then all of them will correspond to the same density. If necessary, all of them will be selected first at no extra cost. Only then, the search needs to find another segment of lower density by following all of  $p_i, p_{i'}, p_{i''}, \dots$ . If this total cost is averaged over the  $p_i$ s, then it will be the same as that of following each of some  $s$  points with different tangents separately.

Let the single point of contact be  $p_1^j$  ( $p_{j-L-12}$  in Fig. 1). By construction, the contour of lower hull  $H^1$  consists of a portion of the contour of each of  $H_1^2$  and  $H_2^2$ . They are joined by an edge, called bridge [13], between the 2 nearest





**Fig. 1.** Finding the next point with respect to right end point  $p_j$

end points of those portions. So,  $p_1^j$  must lie either on  $H_1^2$  or on  $H_2^2$ . Let it lie on  $H_2^2$ . We want to find the next maximum density segment with the same right end point  $p_j$ . Let the left end point of this segment be  $p_2^j$ . We need to find it. Clearly, it lies either on the contour of or interior to  $H^1$ .

By construction of  $H$ , any pair of  $lch$  at the same level are mutually disjoint,  $H_{j_1}^i \cap H_{j_2}^i = \phi$  for all  $j_1$  and  $j_2$  with  $j_1 \neq j_2$ . Since  $p_1^j$  lies on the contour of  $H_2^2$ ,  $p_2^j$  can either be the point of contact of tangent from  $p_j$  to  $H_1^2$ , or on the contour or interior of  $H_2^2$ . To find the point of contact with  $H_1^2$ , the contour of  $H_1^2$  can be searched in  $O(\lg m)$  time using binary search on the array associated with the corresponding node  $c_1^2$  in  $C$ . The second case is the same as the initial problem except for the  $lch$  changed from  $H^1$  to  $H_2^2$ . Thus, the problem is solved recursively. There are  $\lg m$  recursions. In each recursion the tangent point to the contour of one  $lch$  is found by using binary search on the array of points of the contour. Total time for searching  $p_2^j$  is  $O(\lg^2 m)$ .

For each point  $p_j \in R$ , we find the length constrained maximum density segment with  $p_j$  as the right end point. This is done in  $O(\lg m)$  time by drawing tangent from  $p_j$  to the top level  $lch$   $H^1$  (stored at the root of  $C$ ). The tangent point is found by using a binary search of the array associated with the root of  $C$ . For each  $p_j$  a node  $v_{j_1}$  is constructed for the maximum density segment w.r.t.  $p_j$ . Since the tangents to  $H^1$  from multiple points in  $R$  may have the same point of contact, the same point in  $H^1$  may be left end points for multiple nodes  $v_{j_1}, v_{k_1}, \dots, etc.$ , having distinct right end points  $p_j, p_k, \dots, etc.$  respectively.

A maximum heap  $T$  is constructed using  $v_{j_1}$ ,  $j \in I_r$ , as its node and the density of a segment as the order of the heap. The heap is initially constructed as a balanced binary search tree with the exception that each node has a null

middle child. The middle children will point to an implicit heap that will be initialized and expanded as needed.

From the heap the  $k$  maximum density elements are selected by using Fredrickson's [6] heap selection algorithm. A middle child will be explicitly constructed only when Fredrickson's [6] algorithm reaches there. Each of the vertices in the subtree rooted at a middle child will have a maximum of  $\lg m$  number of children. After the initial construction of  $T$ , we will never create a left child or a right child of any of its initial nodes.

Let  $t$  be any vertex of  $T$ . Let  $p_i$  and  $p_j$  be the left and right end points corresponding to  $t$ . Let  $p_i$  be the tangent point on the  $lch$   $H_r^q$ . Then  $t$  will contain a pointer to  $c_r^q$ , where  $c_r^q$  represents the  $lch$   $H_r^q$ . For each vertex  $u_1^j$  of  $T$ ,  $j \in I_r$ , all the vertices of its middle subtrees as well as itself represent the segments for which right end points are the same point  $p_j$ . Contents of a vertex  $t$  of  $T$  are as follows:

1.  $f(t)$  - a pointer to the father of  $t$  (if any).
2.  $lchild(t)$  - a pointer to the left child of  $t$ .
3.  $rchild(t)$  - a pointer to the right child of  $t$ .
4.  $c(t)$  - a pointer to the root of  $C$  corresponding to vertex  $t$ .
5.  $max(c(t))$  - The maximum field value in the vertex  $c_i$  of  $C$  (pointed to by  $c(t)$ ) where by searching the  $lch$  at  $c_i$  the search has selected  $p_i$  as the left end point of a segment.
6.  $p_j$  - right end point of a segment.
7.  $p_i$  - left end point of a segment with right end point  $p_j$ . As mentioned before, its value is selected by searching the vertex pointed by  $c(t)$ .
8.  $\rho$  - slope of  $p_i p_j$ .

Let Fredrickson's [6] algorithm wants to access the children of a node  $v$  of  $T$ . Accessing the left and right children is straightforward. To access the middle children, it creates them first. Let  $v$  corresponds to the tangent point on  $H_r^q$  w.r.t.  $p_j$ . First, we search for the next maximum density segment w.r.t.  $p_j$ . We search for the tangent point on  $lchs$  at lower levels than  $H_r^q$ . The search is conducted by the recursive algorithm discussed above. A child node of  $v$  is created for the tangent point on each of the lower level  $lchs$  from  $p_j$ . Then Fredrickson's [6] algorithm searches those nodes. The algorithm selects  $k$  maximum density segments in this way. The time at each node is blown up by a factor of  $\lg^2 m$ . We have:

**Lemma 5.** *After constructing the LCH Tree  $C$  of a group  $G$  of size  $m$ , the  $k$  maximum density segments can be found from  $G$  in  $O(k \lg^2 m)$  time.*

To find the  $k$  maximum density segments for the LR-pass the heap  $T$  is constructed from all the groups. Then Fredrickson's [6] algorithm is used to search the  $k$  maximum density segments from it. The CH Trees of the groups are searched as described above. For each pass of each batch, the  $k$  maximum density segments are updated using a linear time selection algorithm [3]. If  $k > k'(U - L)$ , where  $k'$  is some constant number, a single heap  $T$  is constructed for all the passes and all the batches. There will be  $(U - L + 1) \lg(U - L + 1)$  nodes in the tree.

Fredrickson's [6] algorithm is used to search the  $k$  maximum density segments from it as before. From Lemmas 4 and 5 we have the following theorem:

**Theorem 3.** *The above algorithm solves the  $k$  length-constrained maximum density segments problem with uniform length and arbitrary  $L$  and  $U$  in  $O(n \lg(U - L + 1) + k \lg^2(U - L + 2))$  time and  $O((U - L + 1) \lg(U - L + 2) + k)$  working space.*

### 3.3 Large $k$

By large  $k$  we mean  $k = f(n_0)$  for some  $n_0$  and some  $f(n) = \Omega(n(U - L + 1))$ . For such  $k$ , a brute force algorithm is more efficient. From the set of all feasible segments,  $k$  maximum density segments are selected, using a linear time selection algorithm [3]. Its time complexity is clearly in  $O(n(U - L + 1))$ . To minimize space, the sequence is scanned from left to right. For each element  $a_j \in A$ , all the feasible segments  $A[i, j]$  with right end element  $a_j$  are considered. The segments are inserted into a candidate set  $D$  of maximum density segments. As soon as  $k$  new segments are inserted into  $D$ ,  $k$  maximum density segments are selected from it using a linear time selection algorithm [3], and  $D$  is updated with the new set of  $k$  maximum density segments. Its space complexity is clearly in  $O(k)$ . Thus we have the following theorem:

**Theorem 4.** *For large  $k$ , there exists an algorithm for the  $k$  length-constrained maximum density segments problem with uniform length, and arbitrary  $L$  and  $U$  whose time and space complexities are in  $O(n(U - L + 1))$  and  $O(k)$  respectively.*

## 4 Conclusions

In this paper, we have presented linear time algorithm for the problem of length-constrained maximum density segments. We have extended our algorithm to find the  $k$  length constrained maximum density segments problem. The algorithms have already been extended to solve the corresponding problems with non-uniform length. We have indicated the extensions of our algorithms to higher dimensions. Our algorithms facilitate efficient solutions for these problems in higher dimensions.

It would be interesting to study if there is any linear time algorithm for the  $k$  length-constrained maximum density segments problem. It can also be investigated to find more efficient algorithms for the problems in higher dimensions. It remains open to improve the trivial lower bounds for these cases.

## References

1. Aho, A., Hopcroft, J., Ullman, J.: The Design and Analysis of Computer Algorithms. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Pub. Co., Boston (1974)

2. Bentley, J.: Programming pearls: perspective on performance. *Commun. ACM* **27**, 1087–1092 (1984)
3. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. *J. Comput. Syst. Sci.* **7**(4), 448–461 (1973)
4. Chung, K.-M., Lu, H.-I.: An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.* **34**(2), 373–387 (2005)
5. Duret, L., Mouchiroud, D., Gautier, C.: Statistical analysis of vertebrate sequences reveals that long genes are scarce in gc-rich isochores. *J. Mol. Evol.* **40**, 308–317 (1995)
6. Frederickson, G.N.: An optimal algorithm for selection in a min-heap. *Inf. Comput.* **104**(2), 197–214 (1993)
7. Goldwasser, M.H., Kao, M.-Y., Lu, H.-I.: Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics. In: Guigó, R., Gusfield, D. (eds.) *WABI 2002. LNCS*, vol. 2452, pp. 157–171. Springer, Heidelberg (2002)
8. Goldwasser, M.H., Kao, M.-Y., Lu, H.-I.: Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. *J. Comput. Syst. Sci.* **70**(2), 128–144 (2005)
9. Kim, S.K.: Linear-time algorithm for finding a maximum-density segment of a sequence. *Inf. Process. Lett.* **86**(6), 339–342 (2003)
10. Lee, D., Lin, T.-C., Lu, H.-I.: Fast algorithms for the density finding problem. *Algorithmica* **53**, 298–313 (2009)
11. Lin, Y.-L., Jiang, T., Chao, K.-M.: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci.* **65**(3), 570–586 (2002)
12. Nekrutenko, A., Li, W.H.: Assessment of compositional heterogeneity within and between eukaryotic genomes. *Genome Res.* **10**(12), 1986–1995 (2000)
13. Overmars, M.H., van Leeuwen, J.: Maintenance of configurations in the plane. *J. Comput. Syst. Sci.* **23**(2), 166–204 (1981)
14. Sharp, P.M., Averof, M., Lloyd, A.T., Matassi, G., Peden, J.F.: DNA Sequence evolution: the sounds of silence. *R. Soc. Lond. Philos. Trans. B* **349**, 241–247 (1995)
15. Stojanovic, N., Florea, L., Riemer, C., Gumucio, D., Slightom, J., Goodman, M., Miller, W., Hardison, R.: Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. *Nucleic Acids Res.* **27**(19), 3899–3910 (1999)
16. Tamaki, H., Tokuyama, T.: Algorithms for the maximum subarray problem based on matrix multiplication. In: *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1998*, pp. 446–452. Society for Industrial and Applied Mathematics, Philadelphia (1998)
17. Zoubak, S., Clay, O., Bernardi, G.: The gene distribution of the human genome. *Gene* **174**(1), 95–102 (1996)

Algorithms and Discrete Applied Mathematics  
Second International Conference, CALDAM 2016,  
Thiruvananthapuram, India, February 18-20, 2016,  
Proceedings  
Govindarajan, S.; Maheshwari, A. (Eds.)  
2016, XIII, 369 p. 76 illus. in color., Softcover  
ISBN: 978-3-319-29220-5