Journal of Combinatorial Optimization, 9, 147–156, 2005

Finding a Length-Constrained Maximum-Density Path in a Tree

RUNG-REN LIN
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

WEN-HSIUNG KUO
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

KUN-MAO CHAO* kmchao@csie.ntu.edu.tw
Department of Computer Science and Information Engineering, Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan

Received July 17, 2003; Revised June 7, 2004; Accepted July 28, 2004

Abstract. Let $T = (V, E, w)$ be an undirected and weighted tree with node set V and edge set E, where $w(e)$ is an edge weight function for $e \in E$. The *density* of a path, say $e_1, e_2, \ldots, e_k$, is defined as $\sum_{i=1}^{k} w(e_i)/k$. The length of a path is the number of its edges. Given a tree with n edges and a lower bound L where $1 \leq L \leq n$, this paper presents two efficient algorithms for finding a maximum-density path of length at least L in $O(nL)$ time. One of them is further modified to solve some special cases such as full m-ary trees in $O(n)$ time.

Keywords: algorithm, computational biology, network design, tree

1. Introduction

Given a sequence of n real numbers and a lower bound L, Lin et al. (2002) recently proposed an $O(n \log L)$-time algorithm for finding a segment of length at least L with the maximum average. Improvements to $O(n)$ were given independently by Goldwasser et al. (2002) and Kim (2003). Lin et al. (2003) implemented an algorithm that delivers k non-overlapping maximum-average segments of a given sequence of real numbers, for any fixed $k > 0$. The maximum-average segment problem arises naturally in several areas of sequence analysis. For example, given a DNA sequence, which segment of the sequence of length at least L has the highest GC ratio (Gardiner-Garden and Frommer, 1987; Huang, 1994; Takai and Jones, 2002)? Given a multiple sequence alignment and the score for each column of the alignment, can we find a subalignment consisting of L or more consecutive columns of the alignment that has the highest cumulative average score (Arslan et al., 2001; Stojanovic et al., 1999)? On the other hand, Wu et al. (1999) studied the problem of finding a path of length no greater than a given upper bound, whose total weight is as large as possible.

*Corresponding author.

In this paper, we study the problem of finding a maximum-density path in a weighted tree. Let $T = (V, E, w)$ be an undirected and weighted tree with node set V and edge set E, where $w(e)$ is an edge weight function for $e \in E$. The *density* of a path, say $e_1, e_2, \ldots, e_k$, is defined as $\sum_{i=1}^{k} w(e_i)/k$. We propose two approaches for finding a maximum-density path in a tree. One approach is to compute for each node a maximum-density path starting at that node. The resulting algorithm can be easily adapted to a directed acyclic graph. The other approach is to locate for each internal node a maximum-density path that takes such an internal node as a least common ancestor. Both approaches run in $O(nL)$ time. The later is further modified to solve some special cases such as full m-ary trees in $O(n)$ time.

Chao et al. (1994) considered constrained alignments consisting of aligned pairs in nearly optimal alignments. These suboptimal alignments are represented as a directed acyclic graph (Chao, 1998). The algorithms developed in this paper are useful in selecting, from all high-scoring alignments, a subalignment that has the highest cumulative average score (Arslan et al., 2001).

The rest of the paper is organized as follows. Section 2 describes the first approach which searches all possible paths starting from a specific node. An alternative approach, which searches all combinations of the *downward paths* of a least common ancestor (LCA) internal node, is given in Section 3. Section 4 discusses a special case that can be solved in linear time, and Section 5 concludes the paper with a few remarks.

2. Finding a maximum-density path from its end nodes

To find a maximum-density segment of length at least L in a one-dimensional sequence, Huang (1994) observed that there exists an optimal solution of length at most $2L - 1$. This property holds for the tree problem as well. Let $\mu(X)$ denote the density of path X in a tree.

Lemma 1. *There exists a length-constrained maximum-density path of length at most* $2L - 1$.

Proof: It is proved by a counter argument. Suppose there does not exist a length-constrained maximum-density path of length at most $2L - 1$. Let B denote the shortest path of length at least L such that the density is maximized. Suppose $|B| \geq 2L$. Bisect B into two subpaths, say C and D, such that both C and D are of length at least L. Without loss of generality, assume that $\mu(C) \geq \mu(D)$. We have $\mu(C) \geq \mu(CD) = \mu(B)$. A contradiction. □

Let $D_K^1[i]$ and $D_K^2[i]$ denote the maximum density and the second-best density of those paths of length i starting from node K, respectively. The *downward table* of node K consists of $D_K^1[i]$ and $D_K^2[i]$ where i is from 1 to $2L - 1$ or to the maximum length of the possible path from node K.

The nodes that determine its neighbor's downward table entries are called *contributors*. We also record those contributors in the downward tables. Take the weighted tree in figure 1

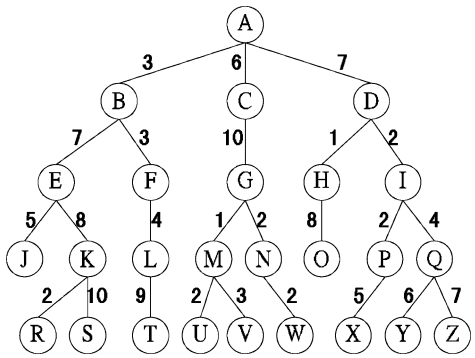

Figure 1. A weighted tree with 26 nodes.

for example. Let us focus on node A. $D_A^1[1] = 7$, and its contributor is node D. $D_A^2[1] = 6$, and its contributor is node C. $D_A^1[2] = 8$, and its contributor is node C.

We can construct the downward tables by dynamic programming. At the beginning, set $D_K^1[0]$ and $D_K^1[0]$ to zero for all nodes. Then compute the other entries from 1 to $2L - 1$ or to the maximum length. Suppose that node K has m neighbors $K_1, K_2, \ldots, K_m$. Let e_j denote the edge (K, K_j), we have

$$\begin{cases} D_K^1[i] \leftarrow \max1\{((i-1) * D_{K_j}^*[i-1] + w(e_j))/i, \quad 1 \le j \le m\} \\ D_K^2[i] \leftarrow \max2\{((i-1) * D_{K_j}^*[i-1] + w(e_j))/i, \quad 1 \le j \le m\}, \end{cases}$$

where $D_{K_j}^*[i-1] = D_{K_j}^2[i-1]$ if K is the contributor of K_j, and $D_{K_j}^*[i-1] = D_{K_j}^1[i-1]$ otherwise. Note that if K is a contributor of node K_j of length $i-1$, we can't take $D_{K_j}^*[i-1]$ into consideration. Furthermore, the function max1 always selects the maximum density. The function max2 selects the maximum density except the one chosen by the function max1. If function max1 or max2 has no candidate, we leave $D_K^1[i]$ or $D_K^2[i]$ undefined. Therefore, $D_K^1[i]$ and $D_K^2[i]$ denote the densities of the best two paths of the different contributors of length i which start from node K. If there is a tie, choose an arbitrary path.

Lemma 2. *The downward tables of all nodes can be constructed in $O(nL)$ time.*

Proof: Assume node K has m neighbors. For a fixed length i, it takes $O(m)$ time to determine $D_K^1[i]$ and $D_K^2[i]$ when $D_K^1[i-1]$ and $D_K^2[i-1]$ are known. In total, it takes $O(mL)$ time to construct the downward table of node K with m neighbors. By amortizing this cost to its m neighbors, we spend $O(L)$ time for each edge. Thus the time complexity of constructing the downward tables of all nodes are $O(nL)$, since there are n edges in the tree. □

Once the downward table of each node is constructed, a maximum-density path of the tree can be computed. Figure 2 gives the pseudo code for computing the density of the maximum-density path.

FMDP(T)
Input: A weighted tree T with n edges.
Output: The density of the maximum density path of length at least L of tree T.
1 **for** each node K **do**
2 $D^1_K[0] \leftarrow 0$
3 $D^2_K[0] \leftarrow 0$
4 **end for**
5 **for** $i \leftarrow 1$ **to** $2L-1$ **do**
6 **for** each node K with m neighbors $K_1, K_2, \ldots, K_m$ **do**
7 $D^1_K[i] \leftarrow \max1\{((i-1) * D^*_{K_j}[i-1] + w(e_j))/i,\ 1 \le j \le m\}$
8 $D^2_K[i] \leftarrow \max2\{((i-1) * D^*_{K_j}[i-1] + w(e_j))/i,\ 1 \le j \le m\}$
9 **end for**
10 **end for**
11 output $\max_{K \in V}\{D^1_K[i], L \le i \le 2L-1\}$

Figure 2. Algorithm for finding the density of the maximum-density path of length at least L.

Lemma 3. *The density of a maximum-density path of length at least L that starts from node K is* $\max\{D^1_K[i],\ L \le i \le 2L-1\}$.

Proof: By Lemma 1, the length of a maximum-density path is in the range from L to $2L-1$. The downward table of node K keeps the densities of the maximum-density paths for each length. Thus, the density of a maximum-density path starting from node K is $\max\{D^1_K[i],\ L \le i \le 2L-1\}$. □

Theorem 1. FMDP *runs in $O(nL)$ time.*

Proof: The time complexity of constructing downward tables for all nodes is analyzed in Lemma 2. Based on the downward tables, we can find the density of an optimal path starting from any specific node in $O(L)$ time. Therefore, in total it takes $O(nL)$ time to find the density of a length-constrained maximum-density in a tree. A simple backtracking procedure could be employed to deliver a maximum-density path. □

3. Finding a maximum-density path from its LCA nodes

This section presents an alternative approach for computing a maximum-density path. At the beginning, we root the tree at any node. It locates for each internal node a maximum-density path that takes such an internal node as a least common ancestor. The least common ancestor of two nodes is the node that is an ancestor of both which has the greatest depth in the tree. For instance, the common ancestors of nodes U and W in figure 1 are A, C, and G. And node G has the greatest depth, so it is the least common ancestor of nodes U and W.

We construct the downward tables as mentioned in Section 2. However, neither the contributors nor the parent of node K are considered. Table 1 is the downward table of

Table 1. The downward table of node A in figure 1.

Length	1	2	3	4
max1	7:D	8:C	6:C	7:B
max2	6:C	5:B	6:B	5:D

node A in figure 1. Each entry in Table 1 consists of two components, denoted as $a : b$, where a is the density and b is the contributor. Let the *downward path* of a given node be a path stretching to its children only. Basically, for each internal node we want to compute a maximum-density path taking that internal node as a least common ancestor. Such a path would be either a single downward path of length at least L, or a combination of two downward paths of length at least L in total. A naïve algorithm takes $O(m^2L^2)$ time to compute all combinations of a given node with m children. An improvement to $O(mL)$ is given in the following.

Define $Path(K, K_i)$ as the length-constrained maximum-density path which combines two non-overlapping downward paths starting from node K, and one of them is contributed by its child K_i. That is, $Path(K, K_i)$ is a maximum-density path that contains edge (K,K_i), and node K is the highest node of such path.

Now we show how to compute $Path(K, K_i)$ in $O(L)$ time. For a one-dimensional sequence A of length $2 * (2L - 1)$, the left half elements are numbered from right to left as $l_1, l_2, \ldots, l_{2L-1}$, and the right half elements are numbered from left to right as $r_1, r_2, \ldots, r_{2L-1}$. For the downward table of node K, we put all entries that are contributed by K_i into the position l_j of sequence A, where j is the corresponding length of those entries in the downward table. $A[l_j]$ will be "null" if node K_i does not contribute $D_K^1[j]$ or $D_K^2[j]$. We fill zero into $A[l_j]$ in this case. Then we put $D_K^1[j]$ into the position r_j of sequence A for each $1 \leq j \leq 2L - 1$, but if $D_K^1[j]$ is contributed by node K_i, we choose $D_K^2[j]$ instead. Now, the elements that are numbered as l_j or r_j in sequence A stand for the densities of length j. We can convert sequence A into *match sequence* B such that $(\sum_{i=1}^{k} B[l_i])/k = A[l_k]$ and $(\sum_{i=1}^{k} B[r_i])/k = A[r_k]$. The algorithm of converting a sequence to a match sequence is given in figure 3. The converting algorithm runs in $O(L)$ time.

For example, consider $Path(A, B)$ in figure 1. The sequence that is filled in for $Path(A, B)$ is shown in figure 4(A), and its match sequence is shown in figure 4(B).

Based on the match sequence corresponding to $Path(K, K_i)$, the problem of finding $Path(K, K_i)$ is reduced to the problem of locating a maximum-average segment of length at least L in a one-dimensional sequence. However, our problem requires some additional constraints. The desirable segment is always starting from a "non-null" element $l_i (1 \leq i \leq 2L - 1)$ in a match sequence, and the right-most element of such segment is always numbered as $r_i (1 \leq i \leq 2L - 1)$. The following lemma proves that the density of such segment equals to the density of $Path(K, K_i)$.

Lemma 4. *Under the constraints mentioned above, the average of the maximum-average segment of length at least L found in the match sequence equals the density of Path*(K, K_i).

```
CONVERT(A)
Input: A real sequence A= ⟨a_1, a_2, ..., a_{4L-2}⟩.
Output: Match sequence of A.
1 Init: sumL ← a_{2L-1}; sumR ← a_{2L};
2 for i ← 1 to 2L − 2 do
3     if (a_{2L-1-i} isn't null) then
4         a_{2L-1-i} ← a_{2L-1-i} * (i + 1) - sumL;
5         sumL ← sumL + a_{2L-1-i};
6     end if
7     a_{2L+i} ← a_{2L+i} * (i + 1) − sumR;
8     sumR ← sumR + a_{2L+i};
9 end for
```

Figure 3. Converting a given sequence into a match sequence.

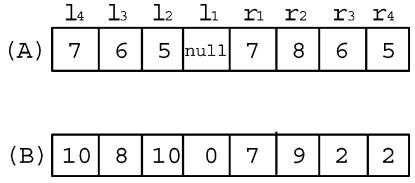

Figure 4. An example of a match sequence.

Proof: The maximum-density path of length at least L of a given LCA internal node is composed of either a single downward path of length at least L, or a combination of two downward paths of length at least L in total. Suppose that downward paths p and q are the best combination of such an LCA internal node. Let $|p|$ denote the length of p, and $|q|$ denote the length of q. If $|p| \neq |q|$, we must combine $D_K^1[|p|]$ and $D_K^1[|q|]$. Otherwise we will get a worse combination. If $|p| = |q|$, the best combination is to add up $D_K^1[|p|]$ and $D_K^2[|q|]$.

Under the constraints mentioned above, the segments of the match sequence corresponding to $Path(K, K_i)$ are actually the combinations of two downward paths, in which one of these two downward paths is guaranteed to contain edge (K, K_i). Thus, the maximum-density segment of such match sequence represents $Path(K, K_i)$. □

Now we show how to solve the reduced one-dimensional sequence problem in linear time. Lin et al. (2002) invented a data structure called a *right-skew segment*. A sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ is *right-skew* if and only if the average of any prefix $\langle a_1, a_2, \ldots, a_i \rangle$ is always less than or equal to the average of the remaining suffix subsequence $\langle a_{i+1}, a_{i+2}, \ldots, a_n \rangle$. Let $\mu(X)$ denote the average of segment X, and $\mu(a_i, a_j)$ denote the average of $A(i, j)$, where $A(i, j)$ stands for $\langle a_i, a_{i+1}, \ldots, a_j \rangle$. A partition $A = A_1 A_2 \ldots A_k$ is *decreasingly right-skew* if each segment A_i of the partition is right-skew and $\mu(A_i) > \mu(A_j)$ for any $i < j$. An example of the right-skew segments is shown in figure 5.

We keep the *right-skew pointer* $P[i]$ for each $1 \leq i \leq n$. $P[i]$ is the maximum index of the right-most elements of all maximum-density segments starting from i. In other words,

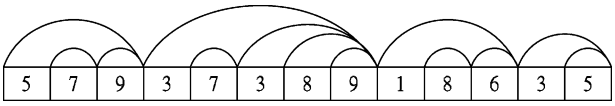

Figure 5. An example of the right-skew segments.

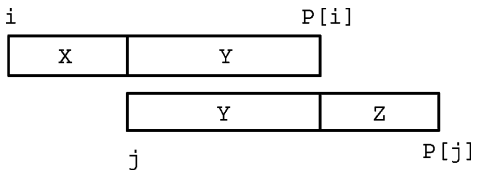

Figure 6. A contradictive example of right-skew segments.

$A(i, P[i])$ is the longest optimal segment of those starting from i. It is obvious that $A(i, P[i])$ is a right-skew segment. Some desirable features of the right-skew segments are described in the following two lemmas due to Goldwasser et al. (2002).

Lemma 5. *There cannot exist i and j such that $i < j \leq P[i] < P[j]$.*

Proof: $A(i, P[i])$ is the maximum-density of those segments that starts from i. Note that $P[i]$ should be maximized if there is a tie. If $i < j \leq P[i] < P[j]$, it means two right-skew segments partially overlapped (see figure 6). Since $P[i]$ stops at the end of the overlapped segment Y, it implies that $\mu(X) \leq \mu(Y)$. Likewise, $\mu(Y) \leq \mu(Z)$. Thus, we have $\mu(X) \leq \mu(Z)$, which implies $\mu(XY) \leq \mu(XYZ)$. It contradicts the statement that $A(i, P[i])$ is the longest optimal segment of those starting from i. □

Lemma 6. *For a given $i < j \leq P[j] < P[i]$, we have $\mu(i, P[i]) < \mu(j, P[j])$.*

Proof: Partition $A(i, P[i])$ into three segments X, Y, and Z as shown in figure 7, where Y is $A(j, P[j])$. Since $P[i]$ stops at the end of Z, it implies $\mu(Z) \geq \mu(XY)$. And $P[j]$ stops at the end of Y without stretching to the end of Z. It follows that $\mu(Y) > \mu(Z)$. So we have $\mu(Y) > \mu(Z) > \mu(X)$. Thus $\mu(i, P[i])$ is less than $\mu(j, P[j])$. □

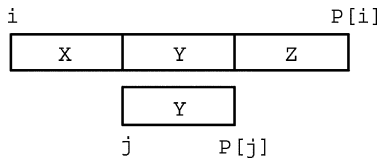

Figure 7. Overlapped right-skew segments.

Table 2. Recorded arrays for figure 5.

	1	2	3	4	5	6	7	8	9	10	11	12	13
a_i	5	7	9	3	7	3	8	9	1	8	6	3	5
$P[i]$	3	3	3	8	5	8	8	8	11	10	11	13	13
S_i			1, 2, 3		5			4, 6, 7, 8		10	9, 11		12, 13

```
FINDMAX(B)
Input: A match sequence B = ⟨b_1, b_2, ..., b_{4L-2}⟩ that corresponds to Path(K, K_i).
Output: The density of Path(K, K_i).
1 Init: y ← 4L − 2; k ← 4L − 2; flag ← true;
2 for i ← 2L − 1 downto 1
3     if (b_i isn't null) then
4         while (flag = true and k ≥ 2L)
5             k ← minimum but no less than i + L in S_y;
6             if (k exists and μ(k, y) < μ(i, k − 1)) then y ← k − 1;
7             else flag ← false;
8         end while
9     end if
10    G[i] ← y;
11    flag ← true;
12 end for
13 output the maximum μ(i, G[i]), where 1 ≤ i ≤ 2L − 1 and b_i isn't null
```

Figure 8. Finding *Path*(K, K_i) in linear time.

We have to record some additional information. Let S_k be a sorted list that contains all indices j for which $P[j] = k$. The arrays for figure 5 are illustrated in Table 2. For instance, $S_8 = \{4, 6, 7, 8\}$ implies that $P[4]$, $P[6]$, $P[7]$ and $P[8]$ are 8. Arrays P and S can be constructed in linear time as shown in Goldwasser et al. (2002) and Lin et al. (2003).

Let $G[i]$ represent the optimal index of a_i with a length constraint. That is, $(i, G[i])$ is of length at least L. The algorithm for finding *Path*(K, K_i) is shown in figure 8.

Lemma 7. *The* FINDMAX *algorithm runs in linear time. It takes a match sequence, which corresponds to Path*(K, K_i)*, as input, and outputs the density of Path*(K, K_i)*.*

Proof: As shown in figure 8, variables k and y in the **while** loop of lines 4 to 8 decrease only. Thus, the algorithm runs in linear time. In line 3, variable i is limited to start with a non-null element, and the rightmost element is always not less than the middle of the input sequence. Therefore, the algorithm outputs the maximum density which is equal to the density of *Path*(K, K_i). □

The algorithm for finding the density of the maximum-density path from LCA nodes is given in figure 9.

Theorem 2. FMDPLCA *runs in* $O(nL)$ *time.*

```
FMDPLCA(T)
Input: A weighted tree T with n edges.
Output: The density of the maximum density path of length at least L of tree T.
1 construct the downward tables for all nodes.
2  for each internal node K with m children K_1, K_2, ..., K_m do
3       for i ← 1 to m do
4            A ← a filled-in sequence for Path(K, K_i)
5            B ← CONVERT(A)
6            construct arrays P and S.
7            FINDMAX(B)
8       end for
9  end for
10 output the maximum value of line 7.
```

Figure 9. Algorithm for finding the density of the maximum-density path based on LCA nodes.

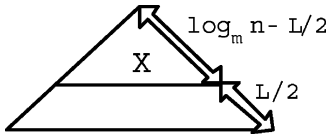

Figure 10. A full m-ary tree with n nodes.

Proof: By Lemma 7, the algorithm for finding the density of $Path(K, K_i)$ runs in $O(L)$ time. That is, each edge is handled in $O(L)$ time. Hence, the time complexity of finding the maximum-density paths of all internal nodes is $O(nL)$. □

4. A special case

Now consider the case where the tree is a full m-ary tree. Imitate the steps mentioned in the Section 3. In this special case, we only need to combine two downward paths when the internal node is of height at least $L/2$ as depicted in figure 10.

Theorem 3. *Finding a length-constrained maximum-density path in a full m-ary tree can be done in linear time.*

Proof: We discuss two trivial cases first. The tree becomes a linear array if $m = 1$, and we can solve the problem in linear time by the one-dimensional algorithm (Goldwasser et al., 2002; Lin et al., 2002). The other case is when $L = 1$. In this case, it suffices to output the maximum weight edge which is promised to be the maximum-density path. Let us consider a more general case in the following. The time for constructing the downward tables for all nodes is $(\sum_{i=0}^{\log_m n} m^i * (\log_m n - i)) = O(n)$. Note that only those internal nodes of height at least $L/2$ (the area X in figure 10) are qualified as a least common ancestor of a path of length at least L. For each qualified internal node, it takes $O(mL)$ time to find a length-constrained maximum-density path that takes such an internal node as a least

common ancestor. Therefore, in total we have

$$\left(m^{(\log_m n - L/2)}\right) \times O(mL) = O\left(\frac{nL}{m^{(L/2)-1}}\right) = O(n).$$

□

5. Concluding remarks

Though our algorithm is for the tree with weighted edges, a straight forward modification will work properly for the tree with weighted nodes. The concept of constructing the downward tables can also be used to solve the case where the graph is a directed acyclic graph. In the future, it would be interesting to consider some other variants such as the problem of finding a size-constrained maximum-density subtree in a tree.

Acknowledgments

We thank Kuan-Yu Chen and the reviewers for making several useful comments. Rung-Ren Lin and Kun-Mao Chao were supported in part by an NSC grant 91-2213-E-002-129 from the National Science Council, Taiwan.

References

A. Arslan, Ö. Eğecioğlu, and P. Pevzner, "A new approach to sequence comparison: normalized sequence alignment," *Bioinformatics*, vol. 17, pp. 327–337, 2001.

K.-M. Chao, R.C. Hardison, and W. Miller, "Recent developments in linear-space alignment methods: A survey," *Journal of Computational Biology*, vol. 1, pp. 271–291, 1994.

K.-M. Chao, "On computing all suboptimal alignments," *Information Sciences*, vol. 105, pp. 189–207, 1998.

M. Gardiner-Garden and M. Frommer, "CpG islands in vertebrate genomes," *J. Mol. Biol.*, vol. 196, pp. 261–282, 1987.

M.H. Goldwasser, M.-Y. Kao, and H.-I. Lu, "Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics," in *Proceedings of the 2nd Workshop on Algorithms in Bioinformatics (WABI)*, 2002, pp. 157–171.

X. Huang, "An algorithm for identifying regions of a DNA sequence that satisfy a content requirement," *CABIOS*, vol. 10, pp. 219–225, 1994.

S.K. Kim., "Linear-time algorithm for finding a maximum-density segment of a sequence," *Information Processing Letters*, vol. 86, pp. 339–342, 2003.

Y.-L. Lin, T. Jiang, and K.-M. Chao, "Efficient algorithms for locating the length-constrained heaviest segments," *Journal of Computer and System Sciences*, vol. 65, pp. 570–586, 2002.

Y.-L. Lin, X. Huang, T. Jiang, and K.-M. Chao, "MAVG, pp. Locating non-overlapping maximum average segments in a given sequence," *Bioinformatics*, vol. 19, pp. 151–152, 2003.

N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, and R.C. Hardison, "Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions," *Nucleic Acids Res.*, vol. 27, pp. 3899–3910, 1999.

D. Takai and P.A. Jones, "Comprehensive analysis of CpG islands in human chromosomes 21 and 22," *PNAS*, vol. 99, pp. 3740–3745, 2002.

B.Y. Wu, K.-M. Chao and C.Y. Tang, "An efficient algorithm for the length-constrained heaviest path problem on a tree," *Information Processing Letters*, vol. 69, pp. 63–67, 1999.