PAPER

Optimal Algorithms for Finding Density-Constrained Longest and Heaviest Paths in a Tree*

Sung Kwon KIM[†a)], ***Member***

SUMMARY Let T be a tree with n nodes, in which each edge is associated with a length and a weight. The density-constrained longest (heaviest) path problem is to find a path of T with maximum path length (weight) whose path density is bounded by an upper bound and a lower bound. The path density is the path weight divided by the path length. We show that both problems can be solved in optimal $O(n \log n)$ time.

key words: *algorithms, density-constrained paths, heaviest paths, longest paths*

1. Introduction

DNA sequences are strings of four letters, A, C, G, and T. The GC-ratio of a DNA sequence is the sum of the numbers of C and G in the sequence divided by the length of the sequence. It is known that subsequences of a DNA sequence with relatively high GC-ratios are biologically meaningful. A promoter of a gene is a subsequence of the DNA sequence containing the gene that facilitates the transcription of the gene, which is usually found near the gene. Promoters are often associated with one or more CpG islands. CpG islands are subsequences with a high frequency of GC residues. Therefore, identifying CpG islands (or subsequences with certain GC ratios) of a newly found DNA sequence is an important task in bioinformatics, which is usually done with the help of computer programs [3], [10].

The task of locating CpG islands can be generalized and formally formulated. Let $A = ((l_1, w_1), \ldots, (l_n, w_n))$ be a sequence of n pairs of reals, in which $l_i > 0$ and w_i are called *length* and *weight*, respectively. For a subsequence $((l_i, w_i), \ldots, (l_j, w_j))$ of A with $i \leq j$, its *length* and *weight* are $l_i + \cdots + l_j$ and $w_i + \cdots + w_j$, respectively, and its *density* is $\frac{w_i+\cdots+w_j}{l_i+\cdots+l_j}$.

A subsequence of A with maximum density can be found in $O(n)$ time [5], [6]. These algorithms can be used to identify CpG islands of a DNA sequence. Longest and shortest subsequences whose density is constrained by a lower bound can be located in linear time [3]. The problems of finding longest and shortest subsequences with upper and lower density bounds require $\Omega(n \log n)$, and [9] gives optimal algorithms for the problems. When both length and weight are constrained with both upper and lower bounds, longest and shortest subsequences can be computed in optimal $O(n \log n)$ time [12].

The problems defined on sequences can be defined on trees as more generalized forms. Let T be a tree with n nodes. Each edge $e \in T$ is associated with two reals $l_e > 0$ and w_e, called its *length* and *weight*, respectively. For two nodes u and v, let $\pi(u, v)$ be the path between them. The *length* (*weight*) of $\pi(u, v)$ is the sum of the lengths (weights) of the edges in it, that is, the length of $\pi(u, v)$ is $l(u, v) = \sum_{e\in\pi(u,v)} l_e$, and its weight is $w(u, v) = \sum_{e\in\pi(u,v)} w_e$. The *density* of $\pi(u, v)$ is defined as $w(u, v)/l(u, v)$.

Locating the longest path of T with non-negative weight can be done in $O(n \log n)$ time [11]. A path of T with maximum weight can be obtained in optimal $O(n \log n)$ time [2], and a path with maximum density can also be found in $O(n \log n)$ time [18]. Both of these algorithms work even if both upper and lower bound constraints are placed on length. When the lengths are restricted to positive integers, instead of reals, dynamic programming algorithms can be developed as in [7], [8], [17]

Related with these problems on a tree, another two problems are addressed in this paper. Given two reals D_1 and D_2, $D_1 \leq D_2$, a path is said to be *density-constrained* if its density is at least D_1 and at most D_2. A path of T is called a *longest* (*heaviest*) path if its length (weight) is the largest among the lengths (weights) of the paths in T.

Problem DCLP (density-constrained longest path): Given a tree T and density bounds D_1 and D_2, find a density-constrained longest path in T.

Problem DCHP (density-constrained heaviest path): Given a tree T and density bounds D_1 and D_2, find a density-constrained heaviest path in T.

Note that the answers to Problems DCLP and DCHP with the same input tree and density bounds are usually not identical. For example, consider a path with three edges, or a sequence of three pairs, $A = ((1, 1), (1, 1), (1, -1))$, and let $D_1 = 0$ and $D_2 = 1$. The DCLP of A is $((1, 1), (1, 1), (1, -1))$ whose density is $\frac{1}{3}$ and length is 3 and its DCHP is $((1, 1), (1, 1))$ whose density 1 and weight is 2.

Problem DCLP is a generalized version of the problems studied in [3], [9] (mentioned earlier), which are defined on sequences. A restricted version of Problem DCHP is studied in [4], which proposes an optimal $O(n \log n)$ time algorithm for the case where T is a path, i.e., a sequence. All of [3], [4] and [9] are motivated by the observation that constraining density with upper and lower bounds is necessary

Manuscript received April 7, 2010.
Manuscript revised July 8, 2010.
[†]The author is with School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea.
*This research was supported by the Chung-Ang University Research Grants in 2009.
a) E-mail: skkim@cau.ac.kr
DOI: 10.1587/transinf.E93.D.2989

to locate good-quality subsequences of DNA sequences, which are further analyzed to be confirmed as CpG islands.

In this paper, we present optimal $O(n \log n)$ time algorithms for both of Problems DCLP and DCHP. Our algorithms are based on divide-and-conquer approaches. In Sect. 2, centroid decomposition is reviewed, which is used as our method of partitioning trees. In Sects. 3 and 4, the algorithms for Problems DCLP and DCHP, respectively, are described. We conclude with final remarks and future works in Sect. 5.

2. Centroid Decomposition

In a binary tree every internal node has degree at most three. As in [18], an arbitrary tree can be transformed into a binary tree by introducing edges of zero length and zero weight so that a solution for the tree can be induced from a solution for the binary tree. From now on, we may assume that T is a binary tree with n nodes.

A *component* of T is a connected subgraph of T. Let C be a component of T. Define $|C|$ to be the number of nodes in C. Deleting a node and its adjacent edges from C leaves at most three components, C_1, C_2, and C_3. A node is called a *centroid* of C, if its removal results in that $|C_i| \le |C|/2$ for $i = 1, 2, 3$. A component has one or two centroids [13]. Let u be a centroid of C. Let C_1 be the one such that $|C_1| \ge |C_2|$ and $|C_1| \ge |C_3|$. Let $v \in C_1$ be the node that is adjacent to u. Deleting the edge (u, v), but not the nodes, from C leaves two components $C' = C_1$ and $C'' = C - C'$. Then, it is easy to verify that

$$\frac{1}{3}|C| \le |C'| \le |C''| \le \frac{2}{3}|C|. \tag{1}$$

The edge (u, v) is called the *wire* of C, and v and u are called the *connector* of C' and C'', respectively.

A centroid decomposition of T works as follows: If T consists of a single node only, then the process finishes. Otherwise, partition T into T' and T'' by locating the wire of T and recursively decompose T' and T''. This procedure of a centroid decomposition of T can be modeled as a rooted binary tree, CT_T. The root of CT_T represents T, and $CT_{T'}$ and $CT_{T''}$ are the left and right subtrees of the root, respectively. CT_T has n leaves and its height is $O(\log n)$ by (1). For each node $a \in CT_T$, let C_a be the component represented by a, and let q_a be the connector of C_a.

Assume that every edge $e \in T$ is assigned a real number s_e, called its *score*. The score of a path $\pi(u, v)$ is the sum of the scores of the edges in the path, i.e., $s(u, v) = \sum_{e \in \pi(u,v)} s_e$. Consider a node $a \in CT_T$. For a node $u \in C_a$, let $\{\langle s(u, v), v\rangle \mid v \in C_a\}$ be a list of score-destination pairs such that $s(u, v)$ is the score of the path from u to v. Sort the list on increasing order of scores, and let $S(u, C_a)$ denote it. If $u = q_a$, then simply $S_a = S(q_a, C_a)$. For a real number s, we define $s \oplus S(u, C_a) = \{\langle s + s', v\rangle \mid \langle s', v\rangle \in S(u, C_a)\}$. Note that $s \oplus S(u, C_a)$ is also sorted.

We show that S_a can be computed in linear time provided that S_b for all descendants b of a in CT_T have been

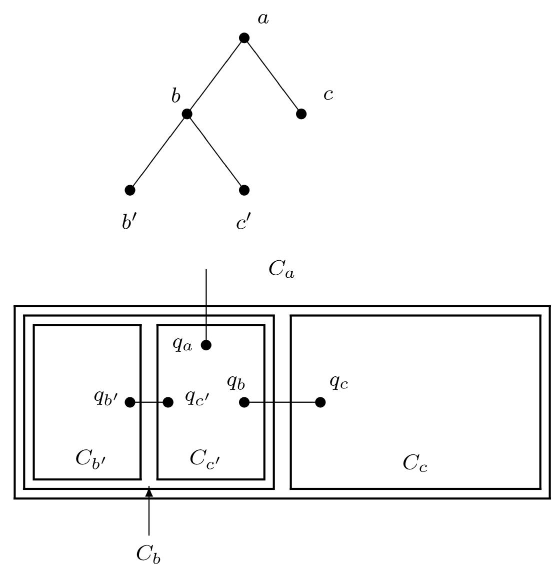

Fig. 1 A subtree of CT_T and a diagram showing inclusions between components.

computed and are stored for reference. Note that an unsorted version of S_a, $\{\langle s(q_a, v), v\rangle \mid v \in C_a\}$, can be obtained in linear time, without the help of the descendants of a, by traversing C_a in postorder after making C_a a rooted tree with root q_a. To get S_a (sorted) in linear time, we need to merge the sorted lists of some of the descendants of a. If a is a leaf in CT_T, then $S_a = \emptyset$. Let b and c be the children of a in CT_T. Wlog, assume that $q_a \in C_b$. Refer to Fig. 1. Then,

$$S_a = \mathrm{MERGE}(S(q_a, C_b), s(q_a, q_c) \oplus S_c),$$

where MERGE$(\cdot, \cdot)$ merges two sorted lists. $s(q_a, q_c)$ can be computed in $O(|C_b|)$ time, and since S_c is available, $s(q_a, q_c) \oplus S_c$ can be obtained in $O(|S_c|) = O(|C_c|)$ time. If C_b consists of a single node only, then $S(q_a, C_b) = \emptyset$. Otherwise, $S(q_a, C_b)$ is recursively computed: $S(q_a, C_b) = \mathrm{MERGE}(S(q_a, C_{c'}), s(q_a, q_{b'}) \oplus S_{b'})$, where b has two children b' and c', and $q_a \in C_{c'}$.

Let $L(|C_a|)$ be the time for computing S_a. Then, $L(1) = 1$, and $L(|C_a|) = L(|C_b|) + \alpha|C_a|$ for constant $\alpha > 0$. Since $\frac{1}{3}|C_a| \le |C_b| \le \frac{2}{3}|C_a|$ by (1), we have $L(|C_a|) \le L(\frac{2}{3}|C_a|) + \alpha|C_a|$ and $L(|C_a|) \ge L(\frac{1}{3}|C_a|) + \alpha|C_a|$. Solving these two inequalities gives $\frac{3}{2}\alpha|C_a| \le L(|C_a|) \le 3\alpha|C_a|$.

Lemma 1: i) S_a for some $a \in CT_T$ can be computed in linear time provided that S_b for all descendants b of a in CT_T are available for reference. ii) S_a for all $a \in CT_T$ can be computed in $O(n \log n)$ time.

Proof: We traverse CT_T in postorder and compute S_a whenever a is visited. Since CT_T has $O(\log n)$ levels and $\sum_{a \text{ at level } i} |S_a| \le n$, $\sum_{a \in CT_T} |S_a|$ is bounded by $O(n \log n)$. □

Note: In Lemma 1, *scores* could be lengths, weights, or any real-numbered values associated with edges if the score of a path is the sum of the scores of the edges on the path. The lemma says that a sorted list of the scores of the paths in C_a from its connector to every node in C_a can be obtained in linear time if the sorted lists for the descendants of a in CT_T are available.

3. Algorithm for Problem DCLP

Our algorithm for Problem DCLP on a binary tree T is a divide-and-conquer algorithm based on centroid decompositions.

(i) Decompose T into two components T_1 and T_2 by locating the wire $\hat{e} = (q, q')$ of T, $q \in T_1$ and $q' \in T_2$, and deleting it.

(ii) Recursively solve the subproblems on T_1 and T_2.

(iii) Combine the subsolutions from (ii) to find a solution for T.

After Step (ii), we have a density-constrained longest path π_1 (π_2), both of whose end nodes are in T_1 (T_2). In Step (iii), we have to find a density-constrained longest path π_3 such that one of its end nodes is in T_1 and the other is in T_2, and return the longest one of $\{\pi_1, \pi_2, \pi_3\}$ as a solution for T.

Let $M(n)$ be the execution time of our algorithm on a tree with n nodes. $M(1) = 1$ and, for $n > 1$, $M(n) = M(n_1) + M(n_2) + \text{Divide}(n) + \text{Combine}(n)$, where $n_1 = |T_1|$, $n_2 = |T_2|$, Divide(n) is time for Step (i), and Combine(n) is time for Step (iii). By (1), $\frac{1}{3}n \le n_1, n_2 \le \frac{2}{3}n$. We have $\text{Divide}(n) = O(n)$ as a centroid of a tree can be found in $O(n)$ time [15], [19]. If we are able to show that $\text{Combine}(n) = O(n)$, then we have $M(n) = O(n \log n)$. In the remainder of this section, we explain how to find π_3 in linear time.

To find π_3, consider two nodes, $u \in T_1$ and $v \in T_2$, in Fig. 2. For $\pi(u, v)$ to be a candidate for π_3, it has to be density-constrained, which states that

$$D_1 \le \frac{w(u,v)}{l(u,v)} \le D_2, \tag{2}$$

or equivalently,

$$D_1 l(u,v) \le w(u,v) \le D_2 l(u,v)$$

as $l(u, v) > 0$.

Define $d_1(u, v) = w(u, v) - D_1 l(u, v)$ and $d_2(u, v) = w(u, v) - D_2 l(u, v)$ for $u \in T_1$ and $v \in T_2$. Then, (2) can be written as

$$d_1(u,v) \ge 0 \qquad \text{and} \qquad d_2(u,v) \le 0. \tag{3}$$

Since $d_1(u, v) = d_1(q, u) + d_1(q, v)$ and $d_2(u, v) = d_2(q, u) + d_2(q, v)$, (3) is equivalent to

$$d_1(q,u) \ge -d_1(q,v) \quad \text{and} \quad -d_2(q,u) \ge d_2(q,v). \tag{4}$$

On a two-dimensional plane, a point may be defined by specifying its x- and y-coordinates. Define a *blue* point $b_u = (x(b_u), y(b_u)) = (d_1(q, u), -d_2(q, u))$ for each $u \in T_1$ and a *red* point $r_v = (x(r_v), y(r_v)) = (-d_1(q, v), d_2(q, v))$ for each $v \in T_2$. We say that a point $(x(p), y(p))$ *dominates* another point $(x(p'), y(p'))$ if $x(p) \ge x(p')$ and $y(p) \ge y(p')$. Then, (4) is equivalent to saying that b_u dominates r_v. In

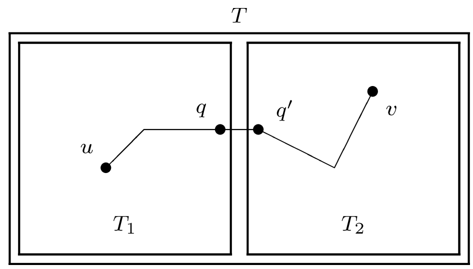

Fig. 2 Finding $\pi_3 = \pi(u, v)$.

other words, $\pi(u, v)$ for $u \in T_1$ and $v \in T_2$ is density-constrained if and only if b_u dominates r_v. Hence, π_3 can be found by locating a blue-red pair of points b_u and r_v such that b_u dominates r_v and $l(u, v)$ is as large as possible.

We need to enumerate, for each blue point, all red points that are dominated by it and to find one that maximizes the length of the path between them. Before this, some blue points and red points that are useless may be eliminated from further consideration.

Let $B = \{b_u \mid u \in T_1\}$ and $R = \{r_v \mid v \in T_2\}$. For $b_u \in B$, $x(b_u) + y(b_u) = d_1(q, u) - d_2(q, u) = w(q, u) - D_1 l(q, u) - (w(q, u) - D_2 l(q, u)) = (D_2 - D_1) l(q, u)$. Consider two blue points b_u and $b_{u'}$ such that b_u dominates $b_{u'}$. Since by the definition of dominance, $x_u \ge x_{u'}$ and $y_u \ge y_{u'}$, we have $x_u + y_u \ge x_{u'} + y_{u'}$ and thus, $(D_2 - D_1) l(q, u) \ge (D_2 - D_1) l(q, u')$, which implies that $l(q, u) \ge l(q, u')$. In other words, if b_u dominates $b_{u'}$, then $l(q, u) \ge l(q, u')$. Since every red point dominated by $b_{u'}$ is also dominated by b_u and $l(q, u) \ge l(q, u')$, $b_{u'}$ is useless in the sense that its elimination from B does not affect final solution.

Similarly, for $r_v \in R$, $x(r_v) + y(r_v) = -d_1(q, v) + d_2(q, v) = -(D_2 - D_1) l(q, v)$. If r_v dominates $r_{v'}$, then $x(r_v) + y(r_v) \ge x(r_{v'}) + y(r_{v'}) \iff -(D_2 - D_1) l(q, v) \ge -(D_2 - D_1) l(q, v') \iff l(q, v) \le l(q, v')$. In this case, r_v is useless and may be removed from R without affecting final solution. Useless points in B and R can be deleted in linear time provided that each of B and R is sorted on x-coordinates. This is called the maxima of a point set in the literature [14], [16].

Since both B and R have no useless points, each makes a "downward staircase" as in Fig. 3. If the red points, in increasing order of x-coordinates, are stored into an array, then the red points dominated by each blue point forms an interval or a subarray in the array. For example, in Fig. 3, b_u dominates three red points, i.e., the third, fourth, and fifth red points. The intervals can be found by merging the two downward staircases of B and R. For each blue point b_u, find a red point $r_v(u)$ such that $l(q, v(u)) = \max\{l(q, v) \mid b_u \text{ dominates } r_v\}$. This can be done by locating the maximum of the values $l(q, v)$ in each interval. Then, $\max\{l(q, u) + l(q, v(u)) \mid b_u \in B\}$ is the length of π_3. Except for sorting B and R on x-coordinates, the work for computing π_3 is linear.

To obtain x-sorted lists of B and R, we use Lemma 1. Define a score $s_e = w_e - D_1 l_e$ for each $e \in T$. The score

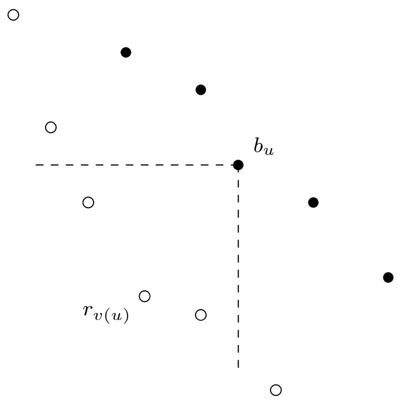

Fig. 3 Computing $r_{v(u)}$ for b_u.

of a path is $s(u,v) = \sum_{e\in\pi(u,v)} s_e = w(u,v) - D_1 l(u,v)$. Then, $d_1(u,v) = s(u,v)$. $S(q,T_1)$ is a sorted list of the x-coordinates of the blue points, and $s_{\hat{e}} \oplus S(q',T_2)$ is a sorted list of the x-coordinates of the red points. Remember that $\hat{e} = (q,q')$, and q and q' are the connectors of T_1 and T_2, respectively. By Lemma 1, $S(q,T_1)$ and $S(q',T_2)$ can be computed in $O(|T_1|)$ and $O(|T_2|)$ time, respectively. $s_{\hat{e}} \oplus S(q',T_2)$ can also be obtained in $O(|T_2|)$ time.

The recursion of our algorithm stops and returns a path of length $-\infty$ when the component has only one node or when the component consists of edges of zero length only. The former is obvious because no further partition of the component is possible, and the reason for the latter is that a component without an edge of positive length does not need to be further considered. Remember that the edges of an input tree have positive lengths and edges of zero length were added in the transformation of the input tree into a binary tree in Sect. 2.

Our algorithm for solving Problem DCLP is in Fig. 4. Since we have shown that Combine(n) = $O(n)$, the following theorem is proved.

Theorem 1: Problem DCLP on a tree with n nodes can be solved in $O(n \log n)$ time which is optimal.

Proof: An $\Omega(n \log n)$ lower bound proof is in [9]. □

4. Algorithm for Problem DCHP

Problem DCHP on T can be solved as follows.

(i) Decompose T into T_1 and T_2 by locating the wire $\hat{e} = (q,q')$ of T, $q \in T_1$ and $q' \in T_2$, and deleting it.

(ii) Recursively solve the subproblems on T_1 and T_2.

(iii) Combine the subsolutions from (ii) to find a solution for T.

Density-constrained heaviest paths π_1 and π_2 of T_1 and T_2, respectively, are recursively obtained by Step (ii). In Step (iii), π_3, a density-constrained heaviest path with one end node in T_1 and the other in T_2, has to be found, and the heaviest one among π_i, $i = 1,2,3$, is returned as a density-constrained heaviest path of T. Step (i) is the same as the one in Sect. 3. We show in the remainder of this section that

```
Algorithm DCLP
Input: A binary tree T with each edge e associated with w_e
    and l_e, and D_1 and D_2 with D_1 ≤ D_2.
Output: ⟨l, u, v⟩ such that π(u, v) is a density-constrained
    longest path of T and l is its length.

DCLP(T)
    for each e ∈ T
        s_e ← w_e − D_1 l_e.
    ⟨l, u, v⟩ ← computeDCLP(T).

computeDCLP(T)
    if T consists of a single node only or
      T has no edge of positive length,
        return ⟨−∞, NULL, NULL⟩.
    locate the wire ê = (q, q′) of T.
    decompose T into T_1 and T_2 so that q ∈ T_1 and q′ ∈ T_2.
    ⟨l_1, u_1, v_1⟩ ← computeDCLP(T_1).
    ⟨l_2, u_2, v_2⟩ ← computeDCLP(T_2).
    compute S_1 ← S(q, T_1) and S_2 ← S(q′, T_2).
    compute w(q, u) and l(q, u) for each u ∈ T_1.
    compute w(q′, v) and l(q′, v) for each v ∈ T_2.
    B ← ∅.
    for each ⟨s, u⟩ ∈ S_1   // scan S_1 in increasing order of s.
        s′ ← −w(q, u) + D_2 l(q, u).
        add blue point b_u = (s, s′) into B.
    R ← ∅.
    for each ⟨s, v⟩ ∈ S_2   // scan S_2 in decreasing order of s.
        s ← s + s_ê.
        s′ ← w(q′, v) − D_2 l(q′, v) + w_ê − D_2 l_ê.
        add red point r_v = (−s, s′) into R.
    eliminate useless points from B and from R.
    compute v(u) for all b_u ∈ B.
    l_3 ← −∞.
    for each b_u ∈ B
        l′ ← l(q, u) + l(q′, v(u)) + l_ê.
        if l′ > l_3
            ⟨l_3, u_3, v_3⟩ ← ⟨l′, u, v(u)⟩.
    l_i ← max{l_1, l_2, l_3}.
    return ⟨l_i, u_i, v_i⟩.
```

Fig. 4 Algorithm for Problem DCLP.

π_3 can be found in linear time, which results in an $O(n \log n)$ time algorithm for the problem.

For $u \in T_1$ and $v \in T_2$, $\pi(u,v)$ has to satisfy (2) to be density-constrained. We have five cases according to the signs of D_1 and D_2. Remember that $D_1 \le D_2$.

4.1 $D_1 > 0$

Since both D_1 and D_2 are positive, it has to be $w(u,v) > 0$ and thus (2) can be rewritten as

$$\frac{w(u,v)}{D_2} \le l(u,v) \le \frac{w(u,v)}{D_1}. \quad (5)$$

Define $h_1(u,v) = \frac{w(u,v)}{D_1} - l(u,v)$ and $h_2(u,v) = \frac{w(u,v)}{D_2} - l(u,v)$. Then, (5) is equivalent to $h_1(u,v) \ge 0$ and $h_2(u,v) \le 0$. Since $h_1(u,v) = h_1(q,u) + h_1(q,v)$ and $h_2(u,v) = h_2(q,u) + h_2(q,v)$, we have

$$h_1(q,u) \ge -h_1(q,v) \quad \text{and} \quad -h_2(q,u) \ge h_2(q,v).$$

Define a blue point $b_u = (h_1(q,u), -h_2(q,u))$ for each $u \in T_1$, and a red point $r_v = (-h_1(q,v), h_2(q,v))$ for each $v \in T_2$. $\pi(u,v)$ is density-constrained if and only if b_u dominates r_v. Moreover, if a blue point b_u dominates another blue point $b_{u'}$, then $b_{u'}$ is useless as

$$\begin{aligned} & x(b_u) + y(b_u) \geq x(b_{u'}) + y(b_{u'}) \\ \iff\quad & h_1(q,u) - h_2(q,u) \geq h_1(q,u') - h_2(q,u') \\ \iff\quad & (\tfrac{1}{D_1} - \tfrac{1}{D_2})w(q,u) \geq (\tfrac{1}{D_1} - \tfrac{1}{D_2})w(q,u') \\ \iff\quad & w(q,u) \geq w(q,u'). \end{aligned}$$

Similarly, if a red point r_v dominates another red point $r_{v'}$, then r_v is useless as

$$\begin{aligned} & x(r_v) + y(r_v) \geq x(b_{v'}) + y(b_{v'}) \\ \iff\quad & -h_1(q,v) + h_2(q,v) \geq -h_1(q,v') - h_2(q,v') \\ \iff\quad & -(\tfrac{1}{D_1} - \tfrac{1}{D_2})w(q,v) \geq -(\tfrac{1}{D_1} - \tfrac{1}{D_2})w(q,v') \\ \iff\quad & w(q,v) \leq w(q,v'). \end{aligned}$$

As in Sect. 3, useless points in B and R can be removed in linear time if both B and R are sorted on x-coordinates. After removing useless points, we merge B and R to find, for each blue point b_u, a red point $r_{v(u)}$ such that b_u dominates $r_{v(u)}$ and $w(u, v(u))$ is as large as possible. This can be done in linear time. Obtaining x-sorted lists of blue and red points can be done by using Lemma 1 as in Sect. 3.

Figure 5 shows our algorithm for Problem DCHP with $D_1 > 0$, which runs in $O(n \log n)$ time. The algorithm, as the algorithm in Fig. 4, stops its recursion when the component consists of a single node or the component has no edge of positive length.

Algorithm DCHP
Input: A binary tree T with each edge e associated with w_e
  and l_e, and D_1 and D_2 with $D_1 \leq D_2$.
Output: $\langle w, u, v\rangle$ such that $\pi(u,v)$ is a density-constrained
  heaviest path of T and w is its weight.

DCHP(T)
  for each $e \in T$
    $s_e \leftarrow w_e/D_1 - l_e$.
  $\langle w, u, v\rangle \leftarrow$ computeDCHP(T).

computeDCHP(T)
  if T consists of a single node only or
   T has no edge of positive length,
    return $\langle -\infty, \mathrm{NULL}, \mathrm{NULL}\rangle$.
  locate the wire $\hat{e} = (q, q')$ of T.
  decompose T into T_1 and T_2 so that $q \in T_1$ and $q' \in T_2$.
  $\langle w_1, u_1, v_1\rangle \leftarrow$ computeDCHP(T_1).
  $\langle w_2, u_2, v_2\rangle \leftarrow$ computeDCHP(T_2).
  compute $S_1 \leftarrow S(q, T_1)$ and $S_2 \leftarrow S(q', T_2)$.
  compute $w(q,u)$ and $l(q,u)$ for each $u \in T_1$.
  compute $w(q',v)$ and $l(q',v)$ for each $v \in T_2$.
  $B \leftarrow \emptyset$.
  for each $\langle s, u\rangle \in S_1$ // scan S_1 in increasing order of s.
    $s' \leftarrow -w(q,u)/D_2 + l(q,u)$.
    add blue point $b_u = (s, s')$ into B.
  $R \leftarrow \emptyset$.
  for each $\langle s, v\rangle \in S_2$ // scan S_2 in decreasing order of s.
    $s \leftarrow s + s_{\hat{e}}$.
    $s' \leftarrow w(q',v)/D_2 - l(q',v) + w_{\hat{e}}/D_2 - l_{\hat{e}}$.
    add red point $r_v = (-s, s')$ into R.
  eliminate useless points from B and from R.
  compute $v(u)$ for all $b_u \in B$.
  $w_3 \leftarrow -\infty$.
  for each $b_u \in B$
    $w' \leftarrow w(q,u) + w(q', v(u)) + w_{\hat{e}}$.
    if $w' > w_3$
      $\langle w_3, u_3, v_3\rangle \leftarrow \langle w', u, v(u)\rangle$.
  $w_i \leftarrow \max\{w_1, w_2, w_3\}$.
  return $\langle w_i, u_i, v_i\rangle$.

Fig. 5 Algorithm for Problem DCHP with $D_1 > 0$.

4.2 $D_1 = 0$

Since $D_1 = 0$, (2) can be expressed as

$$0 \leq w(u,v) \leq D_2 l(u,v). \tag{6}$$

Define $d(u,v) = w(u,v) - D_2 l(u,v)$. Then, (6) can be rewritten as $w(u,v) \geq 0$ and $d(u,v) \leq 0$, which is equivalent to

$$w(q,u) \geq -w(q,v) \qquad \text{and} \qquad -d(q,u) \geq d(q,v)$$

as $w(u,v) = w(q,u) + w(q,v)$ and $d(u,v) = d(q,u) + d(q,v)$.

Define a blue point $b_u = (w(q,u), -d(q,u))$ for each $u \in T_1$ and a red point $r_v = (-w(q,v), d(q,v))$ for each $v \in T_2$. Then, $\pi(u,v)$ is density-constrained if b_u dominates r_v. Hence, π_3 is $\pi(u,v)$ such that b_u dominates r_v and $w(u,v)$ is as large as possible. Since $w(u,v) = w(q,u) + w(q,v) = x(b_u) - x(r_v)$, maximizing $w(u,v)$ is equivalent to maximizing $x(b_u) - x(r_v)$, which is the x-distance between b_u and r_v.

If b_u dominates $b_{u'}$, then $b_{u'}$ is useless because every red point dominated by $b_{u'}$ is also dominated by b_u and $x(b_u) \geq x(b_{u'})$. Similarly, if r_v dominates $r_{v'}$, then r_v is useless because every blue point dominating r_v also dominates $r_{v'}$ and $x(r_{v'}) \leq x(r_v)$.

As in Sect. 3, useless points in B and R can be removed in linear time if the x-sorted lists of B and of R are available. With B and R having no useless points, we can find, for each $b_u \in B$, $r_{v(u)}$ in R such that b_u dominates $r_{v(u)}$ and $x(b_u) - x(r_{v(u)})$ is as large as possible. Getting the blue and red points sorted on x-coordinates can be done by using Lemma 1 as in Sect. 3. In this case, we set $s_e = w_e$ for each $e \in T$. An algorithm, similar to the one in Fig. 5, can be written (omitted) for this case, which also runs in $O(n \log n)$ time.

4.3 $D_2 < 0$

Since both D_1 and D_2 are negative, (2) is now equivalent to

$$-D_2 \leq \frac{-w(u,v)}{l(u,v)} \leq -D_1. \tag{7}$$

Since both $-D_1$ and $-D_2$ are positive, the algorithm in Sect. 4.1 can be used if, for each edge $e \in T$, its weight w_e is replaced by $-w_e$.

4.4 $D_2 = 0$

Since $D_2 = 0$, (2) can be written as $D_1 l(u, v) \le w(u, v) \le 0$, which is equal to

$$0 \le -w(u, v) \le -D_1 l(u, v).$$

Since $-D_1 \ge 0$, the algorithm in Sect. 4.2 can be used after w_e for $e \in T$ is replaced by $-w_e$.

4.5 $D_1 < 0$ and $D_2 > 0$

The range $[D_1, D_2]$ is divided into two subranges $[D_1, 0]$ and $[0, D_2]$. With $[D_1, 0]$ ($D_2 = 0$), the algorithm in Sect. 4.4 is applied to get π'_3, and, with $[0, D_2]$ ($D_1 = 0$), the algorithm in Sect. 4.2 is applied to get π''_3. π_3 is the heavier one of π'_3 and π''_3.

Combining the results from Sects. 4.1–4.5, we have completed our proof that π_3 can be found in linear time, which leads to the theorem.

Theorem 2: Problem DCHP on a tree with n nodes can be solved in $O(n \log n)$ time which is optimal.

Proof: [4] has an $\Omega(n \log n)$ lower bound proof. □

5. Conclusions

We have studied the problems of finding a longest or heaviest path of a tree with its density constrained by upper and lower bounds. The problems have been shown to be solved in optimal $O(n \log n)$ time, where n is the size of the input tree.

One possible future work is finding a longest path of a tree with both length and weight constrained by both upper and lower bounds. More generally, one may develop a general framework which can be used to solve the problems of finding paths that optimize a certain objective function under constraints on length, weight, or density, as Bernholt et al. [1] do on sequences.

References

[1] T. Bernholt, F. Eisenbrand, and T. Hofmeister, "Constrained Minkowski sums: A geometric framework for solving interval problems in computational biology efficiently," Discrete Computational Geometry, vol.42, pp.22–36, 2009.
[2] B. Bhattacharyya and F. Dehne, "Using spine decompositions to efficiently solve the length-constrained heaviest path problem for trees," Inf. Process. Lett., vol.108, pp.293–297, 2008.
[3] K.Y. Chen and K.M. Chao, "Optimal algorithms for locating the longest and shortest segments satisfying a sum or an average constraint," Inf. Process. Lett., vol.96, pp.197–201, 2005.
[4] C.H. Cheng, H.F. Liu, and K.M. Chao, "Optimal algorithms for the average-constrained maximum-sum segment problem," Inf. Process. Lett., vol.109, pp.171–174, 2009.
[5] K.M. Chung and H.I. Lu, "An optimal algorithm for the maximum density segment problem," SIAM J. Comput., vol.34, pp.373–387, 2004.
[6] M.H. Goldwasser, M.Y. Kao, and H.I. Lu, "Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications," J. Comput. Syst. Sci., vol.70, pp.128–144, 2005.
[7] S.Y. Hsieh and C.S. Cheng, "Finding a maximum density path in a tree under the weight and length constraints," Inf. Process. Lett., vol.105, pp.202–205, 2008.
[8] S.Y. Hsieh and T.Y. Chou, "The weight-constrained maximum-density subtree problem and related problems in trees," J. Supercomputing, published online, 2009.
[9] Y.H. Hsieh, C.C. Yu, and B.F. Wang, "Optimal algorithms for the interval location problem with range constraints on length and average," IEEE Trans. Computational Biology and Bioinformatics, vol.5, no.2, pp.281–290, 2008.
[10] X. Huang, "An algorithm for identifying regions of a DNA sequence that satisfy a content requirement," Computer Applications in the Biosciences, vol.10, pp.219–225, 1994.
[11] S.K. Kim, "Finding a longest nonnegative path in a constant degree tree," Inf. Process. Lett., vol.93, pp.275–279, 2005.
[12] S.K. Kim, "Optimal online and offline algorithm for finding longest and shortest subsequences with length and sum constraints," IEICE Trans. Inf. & Syst., vol.E93-D, no.2, pp.250–256, Feb. 2010.
[13] D.E. Knuth, The Art of Computer Programming, Fundamental Algorithms, Second ed., Addison-Wesley, 1973.
[14] H.T. Kung, F. Luccio, and F.P. Preparata, "On finding the maxima of a set of vectors," J. ACM, vol.23, no.4, pp.469–476, 1975.
[15] N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran, "An $O(n \log^2 n)$ algorithm for the k-th longest path in a tree with applications to location problems," SIAM J. Comput., vol.10, no.2, pp.328–337, 1981.
[16] F.P. Preparata and M.I. Shamos, Computational Geometry, an Introduction, Springer-Verlag, 1985.
[17] H.H. Su, C.L. Lu, and C.Y. Tang, "An improved algorithm for finding a length-constrained maximum-density subtree in a tree," Inf. Process. Lett., vol.109, pp.161–164, 2008.
[18] B.Y. Wu, "An optimal algorithm for the maximum-density path in a tree," Inf. Process. Lett., vol.109, pp.975–979, 2009.
[19] B.Y. Wu, K.-M. Chao, and C.Y. Tang, "An efficient algorithm for the length-constrained heaviest path problem on a tree," Inf. Process. Lett., vol.69, pp.63–67, 1999.

Sung Kwon Kim received his B.S. degree from Seoul National University, Korea, his M.S. degree from KAIST, Korea, and his Ph.D. degree from University of Washington, Seattle, U.S.A. He is currently with Department of Computer Science and Engineering, Chung-Ang University, Seoul, Korea.