

Singapore Management University Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

12-2006

Finding a length-constrained maximum-sum or maximum-density subtree and its application to logistics

Hoong Chuin LAU

Singapore Management University, hclau@smu.edu.sg

Trung Hieu NGO


National University of Singapore

Bao Nguyen Nguyen

National University of Singapore

DOI: <https://doi.org/10.1016/j.disopt.2006.06.002>

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Numerical Analysis and Scientific Computing Commons](#), and the [Operations Research, Systems Engineering and Industrial Engineering Commons](#)

Citation

LAU, Hoong Chuin; NGO, Trung Hieu; and Nguyen, Bao Nguyen. Finding a length-constrained maximum-sum or maximum-density subtree and its application to logistics. (2006). *Discrete Optimization*. 3, (4), 385-391. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/1188

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Finding a length-constrained maximum-sum or maximum-density subtree and its application to logistics

Hoong Chuin Lau^{a,*}, Trung Hieu Ngo^b, Bao Nguyen Nguyen^b

^a Singapore Management University, Singapore 178902, Singapore

^b National University of Singapore, Singapore 119260, Singapore

Abstract

We study the problem of finding a length-constrained maximum-density path in a tree with weight and length on each edge. This problem was proposed in [R.R. Lin, W.H. Kuo, K.M. Chao, Finding a length-constrained maximum-density path in a tree, Journal of Combinatorial Optimization 9 (2005) 147–156] and solved in $O(nU)$ time when the edge lengths are positive integers, where n is the number of nodes in the tree and U is the length upper bound of the path. We present an algorithm that runs in $O(n \log^2 n)$ time for the generalized case when the edge lengths are positive real numbers, which indicates an improvement when $U = \Omega(\log^2 n)$. The complexity is reduced to $O(n \log n)$ when edge lengths are uniform. In addition, we study the generalized problems of finding a length-constrained maximum-sum or maximum-density subtree in a given tree or graph, providing algorithmic and complexity results.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Network design; Algorithm; Computational complexity; Logistics

1. Introduction

In this paper, we study the problem of finding a *length-constrained maximum-density path* in a tree (or **LDP**). Let $T = (V, E, w, l)$ be a rooted, undirected and weighted tree with set V of nodes, set E of edges, and a pair of functions $w(e)$ and $l(e)$ that represent the positive weight and the positive length of each edge e in E respectively. The *density* of the path $e_1 e_2 \dots e_k$ is defined as $\frac{\sum_{i=1}^k w(e_i)}{\sum_{i=1}^k l(e_i)}$. Given a lower bound L and an upper bound U , we propose an $O(n \log^2 n)$ -time algorithm to find a maximum-density path with length at least L and at most U . The complexity can be reduced to $O(n \log n)$ time when the edge lengths are uniform, i.e. $l(e) = 1$ for each edge e .

Two special cases of **LDP** are described as follows. Given a sequence of n pairs of real numbers (w_i, l_i) , a lower bound L and an upper bound U , find a segment, i.e. a consecutive subsequence, of length at least L and at most U with maximum sum or with maximum density. We denote these two problems as *LSS* and *LDS*, which stand

* Corresponding address: Singapore Management University, School of Information Systems, Singapore. Tel.: +65 68280229; fax: +65 68280919.

E-mail addresses: hclau@smu.edu.sg (H.C. Lau), ngochung@comp.nus.edu.sg (T.H. Ngo), nguyenna@comp.nus.edu.sg (B.N. Nguyen).

for *length-constrained maximum-sum segment* and *length-constrained maximum-density segment* respectively. Lin et al. [7] solved *LSS* in $O(n)$ time. For *LDS*, Huang [4] noted that the length of a maximum-density segment with length at least L is at most $2L - 1$, and thus provided an $O(nL)$ -time algorithm when there is no upper bound. Lin et al. [7] devised the concepts of right-skew segments and decreasing right-skew partitions, and used them to solve *LDS* in $O(n \log L)$ time. Goldwasser et al. [2,3] improved this time complexity to $O(n)$ for arbitrary L and U . In another track, Kim [5] transformed *LDS* to a geometric problem and applied geometric algorithms to solve it in $O(n)$ time. Recently, Chung and Lu [1] proposed the generalized problem of finding a length-constrained maximum-density segment in a sequence of number pairs (w_i, l_i) , where w_i and l_i are the weight and length of position i respectively, and the density between positions (or nodes) i and j is $d(i, j) = (w_i + w_{i+1} + \dots + w_j) / (l_i + l_{i+1} + \dots + l_j)$. They provided linear time algorithms for the new problem.

LDP is an interesting generalization of *LDS*; for when the tree is a path, the problem on a tree is equivalent to the problem on a sequence. In addition, **LDP** is a generalization of the problem proposed in [6], from the case when the edge lengths are positive integers to the case when the edge lengths are positive real numbers; and our $O(n \log n)$ -time algorithm is an improvement over the $O(nU)$ -time algorithm proposed in [6] when $U = \Omega(\log n)$. This also generalizes the result of [8] which solves the maximum-sum (they call it the heaviest-path) problem in $O(n \log^2 n)$ time.

Furthermore in this paper, we consider the generalization of finding a *length-constrained maximum-sum subtree* or a *length-constrained maximum-density subtree* in a graph (**LST-Graph** or **LDT-Graph**). That is, we would like to find a subtree such that the sum of its edge weights or the edge density (the sum of its edge weights divided by the sum of its edge lengths) is maximized over all length-constrained subtrees. These problems were stated as open problems in [8]. We prove that **LST-Graph** and **LDT-Graph** are NP-complete even when the edge lengths are uniform. We then provide $O(nL^2)$ -time algorithms to find a length-constrained maximum-sum or maximum-density subtree in a given tree (**LST-Tree** or **LDT-Tree**).

While **LDP** finds its applications in computational biology such as sequence alignment [6], the problems considered in this paper also have applications in computer, traffic or logistics network design. In a traffic network design for example, given a limited budget, we would like to upgrade a subgraph of the network that has the highest volume of traffic. When we model the traffic network as a graph, the length of each edge represents the length of its corresponding street segment and hence proportional to the upgrading cost. The weight of each edge represents the traffic load on the corresponding street. When the amount of traffic volume in a subgraph is measured as the amount of traffic load divided by the size of the subgraph or as the amount of traffic load only, we get the length-constrained maximum-density problem or the length-constrained maximum-sum problem respectively. Similarly in a global logistics network or multi-echelon supply network, the problem is often to identify the part of the network where the volume of flow of goods/passengers is the most dense within a geographical proximity.

The rest of the paper is organized as follows. Section 2 describes our improved algorithm for **LDP**. In Section 3, we prove that solving **LST-Graph** and solving **LDT-Graph** are NP-hard even when the edge lengths are uniform. We proceed to propose algorithms for **LST-Tree** and **LDT-Tree** when the edge lengths are uniform in Sections 4 and 5 respectively. Finally, in Section 6, we propose some future directions in extending our work.

2. Improved algorithm for LDP

We first consider a decomposition scheme on a tree and then use it to derive a recursive algorithm for solving **LDP**. This scheme is modeled after the idea proposed in [8]. Our purpose is to decompose the tree into two or three subtrees of “small size” rooted at the same node, such that they are disjoint except for the intersection at their root. Based on the decomposition, the algorithm finds an LDP (length-constrained maximum-density path) of the original tree based on the LDPs in each subtree or the LDPs crossing two subtrees.

2.1. Decompose a tree using centroid

For any tree $T = (V, E)$ of n nodes, a node $c \in V$ is called a *centroid* of T if when we delete c and all the edges incident with c , each resulting subtree contains no more than $n/2$ nodes. Each tree has at least one centroid. We can find a centroid in T in $O(n)$ time as follows. We root T at any node and visit the nodes in a postorder traversal. While visiting a node u , we let $T(u)$ be the subtree rooted at u and compute its number of nodes using the following

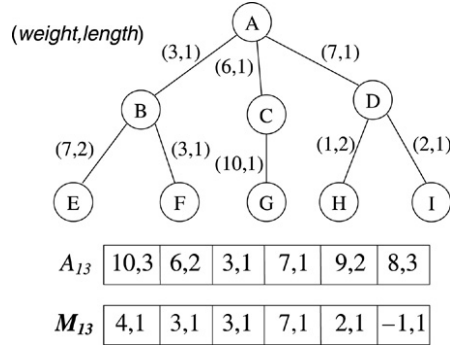


Fig. 1. Illustration of the construction of the match sequence M_{13} corresponding to the pair (T_1, T_3) . The original tree T is decomposed into three trees: T_1 , T_2 and T_3 . T_1 consists of 4 nodes A , B , F and E ; T_2 consists of 3 nodes A , C and G ; and T_3 consists of 4 nodes A , D , I and H . A_{13} is the intermediate sequence which is used to build M_{13} .

recurrence formula: $|T(u)| = 1$ if u is a leaf and $|T(u)| = \sum_{v \in \text{child}(u)} |T(v)| + 1$ otherwise, in which $\text{child}(u)$ is the set of children of u . In the traversal, the first node u that satisfies the inequality $|T(u)| \geq n/2$ is a centroid of T .

After finding a centroid c of T , we root the tree T at c and let c_1, c_2, \dots, c_k be the children of c in the order from left to right. Let $T(c_{i..j})$ be the subtree of T that consists of c and the subtrees $T(c_i), T(c_{i+1}), \dots, T(c_j)$. Let $j = \min\{l \in [1..k-1] \mid \sum_{i=1}^l |T(c_i)| \geq n/2 - 1\}$. Note that this definition is well-defined; in other words, j must exist because c is a centroid. If $j = k - 1$, we have two subtrees $T(c_{1..k-1})$ and $T(c_{k..k})$, each of which contains no more than $n/2 + 1$ nodes. If $j < k - 1$, we have three subtrees $T(c_{1..j-1})$, $T(c_{j..j})$ and $T(c_{j+1..k})$, each of which contains no more than $n/2 + 1$ nodes. Hence, we can decompose any tree T into two or three subtrees rooted at the same node, such that they intersect with each other only at their root and the size of each subtree is not greater than $n/2 + 1$.

2.2. Building a match sequence

From Section 2.1, we can decompose any tree T into three subtrees T_1, T_2 and T_3 , where T_3 could possibly be an empty tree, such that they intersect with each other only at their root u and the size of each subtree is not greater than $n/2 + 1$. Thus, an LDP is either a path in one of these subtrees or a path that consists of two downward paths starting from u to two nodes in two different subtrees. Let us call the later path a path crossing two subtrees. Besides, if we can find a path crossing T_1 and T_2 such that it satisfies the length constraints and its density is maximum among those paths crossing T_1 and T_2 (we denote this path as $\text{path}(T_1, T_2)$), we can do similarly with each pair (T_1, T_3) and (T_2, T_3) . The algorithm can therefore proceed recursively for each subtree to find an LDP in T .

For the pair (T_1, T_2) , we build the *match sequence* M_{12} as follows. We traverse T_1 to build a sequence $l(u_1) \leq l(u_2) \leq \dots \leq l(u_{n_1})$ consisting of all the path lengths from u to all the nodes u_i of T_1 , where n_1 is the number of nodes in T_1 . If two distinct nodes u_i and u_j have the same path length from u in T_1 , we just keep a node such that the weight of the path from u to that node is higher in the sequence and drop the other node. Without loss of generality, we assume that the resulting sequence is $l(u_1) < l(u_2) < \dots < l(u_{m_1})$ where $m_1 \leq n_1$. Similarly, we build a sequence $l(u'_1) < l(u'_2) < \dots < l(u'_{m_2})$ where $m_2 \leq n_2$ for T_2 , where n_2 is the number of nodes in T_2 . Let $f(u_{x..y}) = \sum_{i=x}^y f(u_i)$ and $f(u'_{x..y}) = \sum_{i=x}^y f(u'_i)$ where f may either be the length function l or the weight function w . We concatenate these two sequences to form M_{12} as follows: $M_{12} = u_{m_1} \dots u_2 u_1 u'_1 u'_2 \dots u'_{m_2}$. A new length function l' and a new weight function w' is assigned for the match sequence M , such that $l'(u_{1..i}) = l(u_i)$, $w'(u_{1..i}) = w(u_i)$ for $1 \leq i \leq n_1$, and $l'(u'_{1..j}) = l(u'_j)$, $w'(u'_{1..j}) = w(u'_j)$ for $1 \leq j \leq n_2$. According to our construction, l' is a positive-valued function. M_{13} and M_{23} are computed similarly. See Fig. 1 for the illustration. We have the following lemma, the proof of which is straightforward from the construction of the match sequences and therefore can be omitted.

Lemma 1. *The density of $\text{path}(T_i, T_j)$ is equal to the maximum density of all segments in M_{ij} that have length at least L and at most U , and contain two nodes u_1 and u'_1 , where $1 \leq i < j \leq 3$.*

2.3. Algorithm and time complexity

The algorithm can thus be described as the following procedure:

1. Root T at a centroid c and decompose it into three subtrees T_1 , T_2 and T_3 (see Section 2.1).
2. For each of the pairs (T_i, T_j) where $1 \leq i < j \leq 3$, we build a match sequence M_{ij} (see Section 2.2).
3. For each match sequence M_{ij} ($1 \leq i < j \leq 3$), we find a length-constrained maximum-density segment that contains two nodes u_1 and u'_1 . The path (T_i, T_j) and its density can be computed based on the segment that has been found in M_{ij} .
4. Repeat the algorithm recursively for each of three subtrees T_1 , T_2 and T_3 . The LDP in T is the path that has maximum density among the LDPs in T_i for $1 \leq i \leq 3$ and the paths $\text{path}(T_i, T_j)$ for $1 \leq i < j \leq 3$.

Let $T(n)$ be the time complexity for the above algorithm when it is applied for a tree of size n . Step 1 can be done in $O(n)$ time. Step 2 can be done in $O(n \log n)$ time because it consists of a sorting phase. If the edge lengths are uniform, the sorting phase can be implemented by the counting sort algorithm and therefore Step 2 can be done in $O(n)$ time. Step 3 can be done in $O(n)$ time by using the algorithm described in [1] with little modification. Thus we have the following recurrence relation:

$$T(n) = O(n \log n) + T(n_1) + T(n_2) + T(n_3)$$

where $n_1, n_2, n_3 \leq n/2 + 1$. We can easily derive that $T(n) = O(n \log^2 n)$. If the edge lengths are uniform, we have $T(n) = O(n \log n)$. Hence Theorem 2 follows.

Theorem 2. *An LDP in a tree of size n can be found in $O(n \log^2 n)$ when the edge lengths are positive numbers, and in $O(n \log n)$ when the edge lengths are uniform.*

3. Hardness of LST-graph and LDT-graph

The general case when the edge lengths are general positive numbers is easily proved to be NP-hard even when the graph is a tree using a reduction from the Knapsack problem, as pointed out in [8]. In this paper, we prove that **LST-Graph** and **LDT-Graph** are still NP-hard when the edge lengths are uniform.

3.1. LST-graph is NP-hard

We first present a polynomial-time reduction from the Minimum Set Cover problem to **LST-Graph**.

Minimum (Set) Cover (SC)

INSTANCE: A collection C of subsets of a finite set S and an integer m .

QUESTION: Is there a set cover for S , i.e. a subset $C' \subseteq C$ such that every element in S belongs to at least one member of C' and the cardinality of C' is not greater than m ?

Length-constrained maximum-sum subtree in graph (LST-Graph)

INSTANCE: A graph $\mathbb{G}(V, E)$, a length function $l : E \rightarrow \mathbb{N}$ and a weight function $w : E \rightarrow \mathbb{N}$ and two integers L, U .

SOLUTION: A subtree \mathbb{T} of \mathbb{G} whose length is bounded above by U and bounded below by L .

MEASURE: The weight of \mathbb{T} .

Consider an instance (S, C, m) of the Minimum Cover problem where $|S| = n$, $|C| = k$. Let c_i denote the i th set in C and s_j denote the j th element in S . We construct a graph $\mathbb{G}(V, E)$ where $V = \{u, u', c_1, c_2, \dots, c_k, s_1, s_2, \dots, s_n, s'_1, s'_2, \dots, s'_n\}$ and E consists of the following edges:

- $(u, u') \in E$.
- $(u', c_i) \in E$ for all i .
- $(c_i, s_j) \in E$ if the set C_i contains s_j .
- $(s_j, s'_j) \in E$ for all j .

The length function l is defined so that all edges of \mathbb{G} have uniform length.

The weighting function $w : E \rightarrow \mathcal{N}$ is defined as follows:

- $w(u, u') = N$.
- $w(s_j, s'_j) = N$ where N is some integer bigger than $(n + 1)k$.
- $w(u', c_i) = 1$ for all i .
- $w(c_i, s_j) = 1$ for all $(c_i, s_j) \in E$.

Lemma 3. $SC(S, C, m)$ has a solution if $LST\text{-Graph}(\mathbb{G}, 0, 2n + m + 1) > (n + 1)N$.

Proof. Let $\mathbb{T}(V', E')$ be the maximum-sum subtree of \mathbb{G} whose length does not exceed $2n + m + 1$. Denote the length and weight of \mathbb{T} by $l(\mathbb{T})$ and $w(\mathbb{T})$. We have: $w(\mathbb{T}) > (n + 1)N$.

Since the sum of the weight of (u', c_i) for all i and (c_i, s_j) for all i and j is less than N , any subgraph of \mathbb{G} whose weight is bigger than $(n + 1)N$ must contain all $(n + 1)$ edges that have weight N , i.e. (u, u') and (s_i, s'_i) for all i . Hence, $\{(u, u')\} \cup \{(s_i, s'_i) | 1 \leq i \leq n\} \subseteq E'$. This implies

$$\{u, u'\} \cup \{s_j | 1 \leq j \leq n\} \cup \{s'_j | 1 \leq j \leq n\} \subseteq V'.$$

In addition, that T 's length does not exceed $2n + m + 1$ implies $|V'| \leq 2n + m + 2$. Thus, V' contains at most m nodes from $\{c_i, 1 \leq i \leq k\}$. It is obvious that for each $s_j, 1 \leq j \leq n$, there exists an edge $(c_i, s_j) \in E'$. The collection of all subsets represented by $c_i \in V'$ hence covers S . \square

Lemma 4. If $SC(S, C, m)$ has a solution then $LST\text{-Graph}(\mathbb{G}, 0, 2n + m + 1) > (n + 1)N$.

Proof. Without loss of generality, let $\{C_1, C_2, \dots, C_m\}$ be a set cover of S . We construct a tree $\mathbb{T}(V', E')$ that has length $m + 1 + 2n$ and weight bigger than $(n + 1)N$ as follows:

- $V' = \{u, u'\} \cup \{c_i | 1 \leq i \leq m\} \cup \{s_j | 1 \leq j \leq n\} \cup \{s'_j | 1 \leq j \leq n\}$.
- $(u, u') \in E'$.
- $(u, c_i) \in E'$ for all $1 \leq i \leq m$.
- $(s_j, s'_j) \in E'$ for all $1 \leq j \leq n$.
- $(c_1, s_j) \in E' \forall s_j \in C_1$
- $(c_i, s_j) \in E' \forall s_j \in C_i \setminus \bigcup_{k=1}^{i-1} C_k$. \square

Lemmas 3 and 4 yield the following theorem:

Theorem 5. $LST\text{-Graph}$ is NP-hard even when the edge lengths are uniform and there is no lower bound.

3.2. $LDT\text{-Graph}$ is NP-hard

The same reduction with a new weighting function featuring N bigger than $((n + 1)k + 2n + 1)^2$ can be used to prove that finding a maximum-density subtree with length not smaller than L is also NP-hard. Lemma 6, Lemma 7 and Theorem 8 are derived similarly to Lemma 3, Lemma 4 and Theorem 5.

Length-constrained maximum-density subtree in graph ($LDT\text{-Graph}$)

INSTANCE: A graph $\mathbb{G}(V, E)$ with a length function $l : E \rightarrow \mathbb{N}$ and a weight function $w : E \rightarrow \mathbb{N}$, two positive integers $L < U$

SOLUTION: A subtree \mathbb{T} of \mathbb{G} whose length is at least L and at most U

MEASURE: The density of \mathbb{T} .

Lemma 6. $SC(S, C, m)$ has a solution if $LDT\text{-Graph}(\mathbb{G}, 2n + m + 1, \infty) \geq [(n + 1)N + n + m]/(2n + m + 1)$.

Lemma 7. If $SC(S, C, m)$ has a solution then $LDT\text{-Graph}(\mathbb{G}, 2n + m + 1, \infty) \geq [(n + 1)N + n + m]/(2n + m + 1)$.

Theorem 8. $LDT\text{-Graph}$ is NP-hard even when the edge lengths are uniform and there is no upper bound.

4. Algorithm for LST-Tree

Although the general problems of finding an LST of either a graph (even with uniform-length edges) and general edge-length tree are NP-hard, an LST of a tree with integer edge lengths can be efficiently computed.

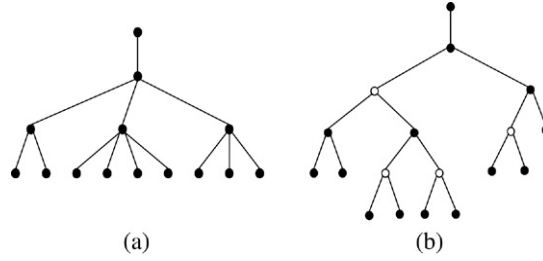


Fig. 2. (a) A general tree. (b) The binary tree transformed from the tree in (a); the black nodes are original nodes, the white nodes are newly created nodes.

4.1. Binary tree

We start by considering binary trees, in which each node has degree less than or equal to 3. The algorithm works by first choosing a leaf and rooting the tree at that leaf. It then annotates each node u of the tree with an array A_u of size U such that $A_u[i]$ stores the weight of the maximum-sum subtree of size i rooted at u .

Initially, for any leaf l , we set $A_l[0] = 0$ and $A_l[i] = -\infty$ for all $i \in \{1, 2, \dots, U\}$. The arrays are then updated in a bottom-up fashion. Given the arrays stored at its children, A_u for internal nodes u is computed using the formulas below:

1. If u has only one child v ,

$$A_u[i] = A_v[i - l(u, v)] + w(u, v). \quad (1)$$

2. If u has two children v and t ,

$$A_u[i] = \max \begin{cases} \max\{A_v[j] + A_t[i - l(u, v) - l(u, t) - j] + w(u, v) + w(u, t) \\ |0 \leq j \leq i - l(u, v) - l(u, t)\} \\ A_v[i - l(u, v)] + w(u, v) \\ A_t[i - l(u, t)] + w(u, t). \end{cases} \quad (2)$$

At each node, we have to update an array of length U . Each element of this array can be updated in $O(U)$ given the array stored at the node's children. Hence, each node is processed in $O(U^2)$. Each node is visited exactly once in this algorithm. The time complexity of the algorithm is $O(nU^2)$.

4.2. General tree

To deal with a general tree, we first transform it into a binary tree by the following procedure (see Fig. 2 for the illustration):

1. Root the tree at any leaf.
2. Traverse the tree in top-down fashion. At each node u , if it has more than 2 children, divide the set of children of u into pairs of two children p_1, p_2, \dots, p_k . for each pair p_i , create a new node u_i and set the two nodes in p_i to be children of u_i . The length and weight of the edge connecting each child to u_i are set to 0. If there are more than 2 new nodes, we repeat the process with them taking the place of children of u . After finishing processing u , we process its children in the original tree. Note that we do not visit the newly created nodes.

Lemma 9. *The above transformation takes $O(n)$ time.*

Proof. It is easy to see that at each node u of degree $d(u)$ in the original tree, there are $O(d(u))$ nodes and edges are created, taking time $O(d(u))$. Since $\sum_u d(u) = n$, the time complexity of this procedure is $O(n)$. Furthermore, the resulting binary tree has $O(n)$ nodes and edges. \square

It is clear that from an LST of the resulting binary tree, we can obtain an LST of the original tree by removing all newly created nodes and joint u and v if v is an ancestor of u and the path from u to v contains only newly created nodes. Hence, an LST of the original tree can be obtained by the following $O(nU^2)$ -time procedure:

1. Transform the tree to a binary tree.

2. Obtain an LST in the resulting binary tree.
3. Transform this subtree back to a subtree of the original tree.

5. Algorithm for LDT-Tree

The same algorithm above can be applied to find an LDT by replacing L by U and Eqs. (1) and (2) by Eqs. (3) and (4) as follows:

$$A_u[i] = \frac{(A_vi - l(u, v) + w(u, v))}{i}. \quad (3)$$

$$A_u[i] = \max \left\{ \begin{array}{l} \max \left\{ \frac{j A_v[j] + (i - l(u, v) - l(u, t) - j) A_t[i - l(u, v) - l(u, t) - j] + w(u, v) + w(u, t)}{i} \right. \\ \left. | 0 \leq j \leq i - l(u, v) - l(u, t) \right\} \\ \frac{A_v[i - l(u, v)] * [i - l(u, v)] + w(u, v)}{i} \\ \frac{A_t[i - l(u, t)] * [i - l(u, t)] + w(u, t)}{i} \end{array} \right\}. \quad (4)$$

This algorithm runs in $O(nU^2)$ by the same argument as above. Besides, if each edge length is either 0 or 1, we have:

Lemma 10. *The length of the smallest LST is at most $3L$.*

Proof. This lemma is clear since we can always separate a tree of length $3L$ into two subtrees of length at least L . The subtree with higher density has density higher than that of the original tree. \square

By Lemma 10, the algorithm can be easily modified to run in $O(nL^2)$.

6. Open problems and future research directions

Some future work can be proposed for the problems studied in this paper. It is an open question as to whether the above algorithms can be improved to have linear time complexity. More investigations on the maximum-sum or maximum-density problems in a graph should be conducted. In addition, the **LST-Tree** or **LDT-Tree** problems can be extended to solve for the case of general edge lengths. Finally, an interesting direction is to find either exact or good approximation algorithms for the problems that are proved to be NP-hard, namely, **LST-Graph** and **LDT-Graph**.

References

- [1] K.M. Chung, H.I. Lu, An optimal algorithm for the maximum-density segment problem, *SIAM Journal on Computing* 34 (2) (2004) 373–387.
- [2] M.H. Goldwasser, M.Y. Kao, 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, 2002*, pp. 157–171.
- [3] M.H. Goldwasser, M.Y. Kao, H.I. Lu, Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications, *Journal of Computer and System Sciences* 70 (2005) 128–144.
- [4] X. Huang, An algorithm for identifying regions of a dna sequence that satisfy a content requirement, *Computer Applications in the Biosciences* 10 (1994) 219–225.
- [5] S.K. Kim, Linear-time algorithm for finding a maximum-density segment of a sequence, *Information Processing Letters* 86 (2003) 339–342.
- [6] R.R. Lin, W.H. Kuo, K.M. Chao, Finding a length-constrained maximum-density path in a tree, *Journal of Combinatorial Optimization* 9 (2005) 147–156.
- [7] Y.L. Lin, T. Jiang, K.M. Chao, Efficient algorithms for locating the length-constrained heaviest segments, *Journal of Computer and System Sciences* 65 (3) (2002) 570–586.
- [8] B.Y. Wu, K.M. Chao, C.Y. Tang, An efficient algorithm for the length-constrained heaviest path problem on a tree, *Information Processing Letters* 69 (1999) 63–67.