LETTER

Optimal Algorithms for Finding the Longest Path with Length and Sum Constraints in a Tree*

Sung Kwon KIM[†a)], ***Member***

SUMMARY Let T be a tree in which every edge is associated with a real number. The sum of a path in T is the sum of the numbers associated with the edges of the path and its length is the number of the edges in it. For two positive integers $L_1 \leq L_2$ and two real numbers $S_1 \leq S_2$, a path is feasible if its length is between L_1 and L_2 and its sum is between S_1 and S_2. We address the problem: Given a tree T, and four numbers, L_1, L_2, S_1 and S_2, find the longest feasible path of T. We provide an optimal $O(n \log n)$ time algorithm for the problem, where $n = |T|$.

key words: *length constraint, longest path, sum constraint, tree*

1. Introduction

Let T be a tree. Each edge $e \in T$ is associated with a real number z_e. For two nodes $u, v \in T$, let $\pi(u, v)$ be the path between them in T. The sum of the numbers associated with the edges of $\pi(u, v)$, denoted by $s(u, v) = \sum_{e \in \pi(u,v)} z_e$, is called the *sum* of $\pi(u, v)$. The *length* of $\pi(u, v)$, denoted by $l(u, v)$, is the number of edges in it. For two positive integers $L_1 \leq L_2$ and two real numbers $S_1 \leq S_2$, a path $\pi(u, v)$ is *feasible* if $L_1 \leq l(u, v) \leq L_2$ and $S_1 \leq s(u, v) \leq S_2$.

We address the following problem and provide an optimal $O(n \log n)$ time algorithm for it, where $n = |T|$.

TreeLFP (longest feasible path of a tree): Given a tree T, and four numbers, L_1, L_2, S_1 and S_2, find the longest feasible path of T.

Kim [5] considered the same problem on sequences, and gave an optimal $O(n \log n)$ algorithm. Since a sequence can be considered as a "linear" tree, he provided a solution for a special case of TreeLFP.

In Sect. 2, we give a linear time algorithm for the case where T is a tree with a special structure, which will be described in detail later. In Sect. 3, we present an optimal $O(n \log n)$ time algorithm for the problem.

2. Twin-Rooted Trees

Let T' and T'' be two rooted trees, and let $g \in T'$ and $g' \in T''$ be their roots. Let T be the tree obtained by connecting T' and T'' with the addition of the edge $\hat{e} = (g, g')$. T is twin-rooted. Each edge $e \in T$, including $\hat{e}$, is associated with a real number z_e. Furthermore, we are given Λ' and Λ'', which are lists of sum-length pairs from g to every node u of T' and from g' to every node u of T'', respectively, i.e., $\Lambda' = \{(s(g, u), l(g, u)) \mid u \in T' - \{g\}\}$ and $\Lambda'' = \{(s(g', u), l(g', u)) \mid u \in T'' - \{g'\}\}$. We assume that Λ' and Λ'' are sorted in increasing order of sums.

TwinLFP (longest feasible path of a twin-rooted tree): Given a twin-rooted tree $T = T' \cup T''$, four numbers, L_1, L_2, S_1, S_2, and two sorted lists Λ', Λ'', find the longest feasible path of T with restriction that one of the end nodes of the path is in T' and the other is in T''.

In other words, among all pairs of nodes $u \in T'$ and $v \in T''$ such that $S_1 \leq s(u, v) \leq S_2$ and $L_1 \leq l(u, v) \leq L_2$, find one that maximizes $l(u, v)$.

To simplify TwinLFP, we first include $\hat{e}$ into T'' by letting $\Lambda'' = \{(s(g, u), l(g, u)) \mid u \in T''\} = \{(s(g', u) + z_{\hat{e}}, l(g', u) + 1) \mid u \in T''\}$. The new Λ'' is also sorted. Since $\pi(u, v)$ for $u \in T'$ and $v \in T''$ can be divided into $\pi(g, u)$ and $\pi(g, v)$, the condition $S_1 \leq s(u, v) \leq S_2$ is equivalent to $S_1 \leq s(g, u) + s(g, v) \leq S_2$, and the condition $L_1 \leq l(u, v) \leq L_2$ is equivalent to $L_1 \leq l(g, u) + l(g, v) \leq L_2$.

Each pair $(s(g, u), l(g, u))$ of Λ' can be represented as a point in the plane whose x- and y-coordinates are $s(g, u)$ and $l(g, u)$, respectively. Denote the set of these points by P. Similarly, another set of points Q is obtained from Λ''. The Minkowski sum $P \oplus Q$ of P and Q is defined to be $\{p \oplus q \mid p \in P, q \in Q\}$, where $p \oplus q = (x_p + x_q, y_p + y_q)$.

TwinLFP can be rewritten: Given P and Q, find a point of $P \oplus Q$ that maximizes the y-coordinate among all points of $P \oplus Q$ whose x-coordinates are between S_1 and S_2 and whose y-coordinates are between L_1 and L_2.

This is one of the constrained Minkowski sum problems in which we are to find a point maximizing an objective function among the points of $P \oplus Q$ which satisfy certain constraints. The constrained Minkowski sum problems were considered by Bernholt *et al.* [1]. In our problem, four constraints, namely, two x-bounds and two y-bounds, are involved. A method by Bernholt *et al.* [1] can be used to solve the problem in $O((|P| + |Q|) \log(|P| + |Q|))$ time.

If $S_1 \leq$ (minimum x-coordinate of $P \oplus Q$), then the x-lower bound is *ineffective* in the sense that it can be ignored. A linear-time algorithm for this case will be given in Sect. 2.1. Section 2.2 will deal with the case of effective x-lower bounds and present a linear-time algorithm.

2.1 Ineffective x-Lower Bounds

Since S_1 is ineffective, TwinLFP becomes: Given P and Q,

Manuscript received September 13, 2010.
Manuscript revised January 19, 2011.

[†]The author is with School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea.

*This work was supported by the KRF grant funded by the Korean government (MEST) (No.2009-0074076).

a) E-mail: skkim@cau.ac.kr.

DOI: 10.1587/transinf.E94.D.1325

find a point of $P \oplus Q$ maximizing the y-coordinate among all points of $P \oplus Q$ whose x-coordinates are less than or equal to S_2 and whose y-coordinates are between L_1 and L_2. Three bounds, i.e, an x-upper bound, and y-upper and -lower bounds, are involved. Our approach will first find, among all points of $P \oplus Q$ that satisfy two bounds, the x-upper and y-upper bounds, one that maximizes the y-coordinate. If the point also satisfies the third bound, the y-lower bound, then the point is one that we want to find. Otherwise, no point of $P \oplus Q$ satisfies the three bounds.

Since both Λ' and Λ'' are sorted in an increasing order of sums, $s(g, u)$, both P and Q are sorted in an increasing order of x-coordinates. Let $P = \{(a_i, b_i) \mid 1 \le i \le m\}$ with $a_1 \le \cdots \le a_m$ where $m = |\Lambda'|$, and $Q = \{(c_j, d_j) \mid 1 \le j \le n\}$ with $c_1 \le \cdots \le c_n$, where $n = |\Lambda''|$. TwinLFP with an ineffective x-lower bound is rephrased as: Given P and Q, among all pairs of i and j such that $a_i + c_j \le S_2$ and $L_1 \le b_i + d_j \le L_2$, find one that maximizes $b_i + d_j$.

Some points of P and Q may be deleted without affecting final solutions. If $a_i \le a_{i'}$ and $b_i = b_{i'}$ for some i, i', then the point $(a_{i'}, b_{i'})$ can be deleted from further consideration. Similarly, if $c_j \le c_{j'}$ and $d_j = d_{j'}$ for some j, j', then the point $(c_{j'}, d_{j'})$ can be deleted. So, we may assume that the y-coordinates of the points of P are all distinct, i.e., $\{b_1, \ldots, b_m\} = \{1, \ldots, m\}$, and similarly, the y-coordinates of the points of Q are distinct, i.e., $\{d_1, \ldots, d_n\} = \{1, \ldots, n\}$.

For $1 \le j \le n$, compute the largest integer $i(j)$ such that $a_{i(j)} + c_j \le S_2$. It is easy to see that $i(1) \ge \cdots \ge i(n)$ and they can be computed in linear time. Then, every pair j and $i \in \{1, \ldots, i(j)\}$ satisfy the x-upper bound, $a_i + c_j \le S_2$.

Next, for each j, find i that maximizes $b_i + d_j$ among all $i \in \{1, \ldots, i(j)\}$ such that $b_i + d_j \le L_2$. Let $i^*(j)$ denote this i. In other words, $i^*(j)$ for each j is the integer i that maximizes b_i among all $i \in \{1, \ldots, i(j)\}$ such that $b_i \le L_2 - d_j$. Among all pairs $(i^*(j), j)$, find one $(i^*(\hat{j}), \hat{j})$ maximizing $b_{i^*(j)} + d_j$. If $b_{i^*(\hat{j})} + d_{\hat{j}} \ge L_1$, then the pair $(i^*(\hat{j}), \hat{j})$ is the one we want to find. Otherwise, we conclude that no pair i, j satisfies $a_i + c_j \le S_2$ and $L_1 \le b_i + d_j \le L_2$.

We are to compute $i^*(j)$ for each j. Make a blue point (i, b_i) for each $1 \le i \le m$, and a red point $(i(j), L_2 - d_j)$ for each $1 \le j \le n$. A point is *dominated* by another point if the x- and y-coordinates of the former are less than or equal to those of the latter, respectively. For each red point $(i(j), L_2 - d_j)$, find its (blue) neighbor, which is a blue point with maximum y-coordinate among all blue points (i, b_i) dominated by the red point. Then, the x-coordinate of the neighbor is $i^*(j)$.

Let $V = \{v_i = (x_i, y_i) \mid 1 \le i \le k\}$ be a y-sorted list of the blue and red points in $P \cup Q$ such that $y_1 \le \cdots \le y_k$, where $k = m + n$. If a blue point and a red point have a same y-coordinate, then the blue one is placed before the red one. Since, as mentioned earlier, $\{b_1, \ldots, b_m\} = \{1, \ldots, m\}$ and $\{d_1, \ldots, d_n\} = \{1, \ldots, n\}$, the sorting can be done in $O(m+n)$ time. Let $(v_{\sigma_1}, \ldots, v_{\sigma_k})$ be a x-sorted list of V such that $x_{\sigma_1} \le \cdots \le x_{\sigma_k}$. Again, a blue point appears before a red point if they have a same x-coordinate. This sorting also takes linear time as $i(1) \ge \ldots \ge i(n)$ is already sorted. An integer i is

```
$K = (0, 1, \ldots, k)$;
for $j = k$ to 1
  if $\sigma_j$ is red
    $\beta(j) = \mathrm{FIND}(\sigma_j)$;      $(*)$
  else
    UNION(predecessor($\sigma_j$), $\sigma_j$);
    delete $\sigma_j$ from $K$;
```

Fig. 1 Algorithm for computing $\beta(j)$.

blue (*red*) if v_i is blue (red).

Consider the sequence $(\sigma_1, \ldots, \sigma_k)$. For each j such that σ_j is red, compute

$$\beta(j) = \max\{\sigma_i \mid i < j, \sigma_i < \sigma_j, \text{ and } \sigma_i \text{ blue }\} \qquad (1)$$

The condition $i < j$ implies that $x_{\sigma_i} \le x_{\sigma_j}$, and the condition $\sigma_i < \sigma_j$ implies that $y_{\sigma_i} \le y_{\sigma_j}$. These two together imply that v_{σ_i} is dominated by v_{σ_j}. Maximizing σ_i corresponds to maximizing the y-coordinate. So, $v_{\beta(j)}$ is the neighbor of v_{σ_j}. The problem of computing $\beta(j)$ was studied by Kim [5] and an $O(k \cdot \alpha(k))$ time algorithm was given, where $\alpha(k)$ is the inverse of the Ackermann function [2], [10]. We show that a careful analysis of the algorithm by Kim improves time complexity to $O(k)$.

To compute $\beta(j)$ for all j with σ_j red, Algorithm in Fig. 1 will be used. K, initially $K = (0, 1, \ldots, k)$, is a sorted list of the blue and red integers, where 0 is assumed to be blue. While scanning $(\sigma_1, \ldots, \sigma_k)$ backward, we compute $\beta(j)$ if σ_j is red, and delete σ_j from K, otherwise.

K will be implemented as a combined data structure of a doubly linked list D and a data structure for disjoint sets S. The blue integers are stored in D. The red integers are partitioned into disjoint sets, each of which is implemented as a rooted tree. S is the collection of the rooted trees. The root of each rooted tree points to exactly one blue integer of D, the *name* of the set. The name of a set is not an element of the set.

Initially, the blue integers, in an increasing order, are stored into D. Each blue integer k' will be associated with a (possibly empty) set whose name is k' in the following way: If k' is the largest blue integer in K, then the red integers larger than it are all contained in the set. Otherwise, if $k'' > k'$ is the blue integer such that no blue integer lies between k' and k'', the red integers between them are all contained in the set. The sets created in this way are disjoint.

A set and its name satisfy the following property. **Property of the name:** The name of a set is the largest blue integer of K that is less than the red integers of the set.

As the algorithm in Fig. 1 proceeds, sets are merged. If σ_j is blue, it is deleted from D. Before deleting it, its set is merged into the set of its predecessor in D by doing UNION(predecessor(σ_j), σ_j). The predecessor of σ_j is the largest integer that is less than σ_j in D. The predecessor of σ_j will be the name of the new set. Deleting σ_j itself from D can be done in constant time as D is a doubly linked list. It is easy to see that the property of the name still holds for the new set and its name. Note that σ_j and predecessor(σ_j) are next to each other in D.

Computing $\beta(j)$ for red σ_j in $(*)$ is done by FINDing the name of the set to which σ_j belongs. Whenever $(*)$ is executed, all blue integers in $(\sigma_{j+1}, \ldots, \sigma_k)$ have been removed from K. So, if we let σ_{i_0} be the name of the set, then $i_0 < j$. This together with the property of the name satisfies the definition of $\beta(j)$ of (1) and thus we have that $\beta(j) = \sigma_{i_0}$. The algorithm correctly computes $\beta(j)$ for all j with σ_j red.

As there are k operations of UNIONs and FINDs, the algorithm takes $O(k \cdot \alpha(k))$ time [2], [10]. The sequence $(\sigma_1, \ldots, \sigma_k)$ is given before the algorithm starts. Each of the UNIONs performed by the algorithm is of the form UNION$(\sigma_{j'}, \sigma_j)$, where $\sigma_{j'}$ and σ_j are adjacent each other in D. So, the order of the UNIONs can be predetermined by scanning the blue integers of $(\sigma_k, \ldots, \sigma_1)$. This corresponds to the *static tree set union* of Gabow and Tarjan [3]. They showed that k UNIONs and FINDs in the static tree set union can be done in $O(k)$ time. So, the time complexity is reduced to $O(k)$.

We have shown that TwinLFP with an ineffective x-lower bound can be solved in linear time.

Lemma 1: TwinLFP with an ineffective x-lower bound can be solved in linear time.

2.2 Effective x-Lower Bounds

TwinLFP with an effective x-lower bound is: Given P and Q, find a point of $P \oplus Q$ maximizing the y-coordinate among all points of $P \oplus Q$ whose x-coordinates are between S_1 and S_2 and whose y-coordinates are between L_1 and L_2.

Using a method by Bernholt *et al.* [1], the problem can be solved in $O((m+n)\log(m+n))$ time as four constraints are involved. We show that this can be reduced to $O(m+n)$. We borrow an idea from Sect. 4.1 of Bernholt *et al.* [1].

A *strip* in the plane is the region bounded by two vertical lines. The *width* of a strip is the distance between its boundaries. Let Σ be the strip whose boundaries are $x = S_1$ and $x = S_2$, and let $\delta = S_2 - S_1$ be its width. We say that a strip *covers* a point if the point is inside the strip.

Find a minimum number of disjoint strips with width $\delta/4$ that together cover P. This can be done in greedy manner by scanning P from left to right. Assume that m' strips have been found. Let P_i be the subset of P that is covered by the i-th strip for $1 \le i \le m'$. Then, $P_1, \ldots, P_{m'}$ is a partition of P. Similarly, find a partition of Q, $Q_1, \ldots, Q_{n'}$. We compute a solution for $P_i \oplus Q_j$ for every pair of i, j such that $1 \le i \le m'$ and $1 \le j \le n'$. The best of these solutions is the final solution for $P \oplus Q$.

Consider $P_i \oplus Q_j$ for some i, j. The points of $P_i \oplus Q_j$ are covered by $\Sigma_{i,j}$, a strip of width $\delta/2$, where $\Sigma_{i,j}$ is the Minkowski sum of the two strips of width $\delta/4$ that covered P_i and Q_j in the previous paragraph. Since Σ is δ wide and $\Sigma_{i,j}$ is $\delta/2$ wide, the possible relative positions of $\Sigma_{i,j}$ with respect to Σ are:

(1) $\Sigma_{i,j}$ does not intersect Σ.

(2) $\Sigma_{i,j}$ is totally inside Σ.

(3) The left boundary of Σ is contained in $\Sigma_{i,j}$.

(4) The right boundary of Σ is contained in $\Sigma_{i,j}$.

The pairs of P_i and Q_j falling under case (1) do not need to be further considered and thus are excluded. Case (4) corresponds to a case with an ineffective x-lower bound: Given P_i and Q_j, find a point of $P_i \oplus Q_j$ maximizing the y-coordinate among all points of $P_i \oplus Q_j$ whose x-coordinates are less than or equal to S_2 and whose y-coordinates are between L_1 and L_2. Case (4) can be solved in linear time, i.e, $O(|P_i| + |Q_j|)$ time by Lemma 1. Case (3) is symmetric to case (4), and case (2) corresponds to a case with no x-bounds and thus is easier to solve than case (4). So, each of cases (2) and (3) can also be solved in $O(|P_i| + |Q_j|)$ time.

To determine how many out of $m'n'$ pairs of P_i and Q_j fall under cases (2)–(4), let $i(j)$ for each j be the smallest integer i such that $\Sigma_{i,j}$ satisfies case (3). Since $\Sigma_{i,j} \cap \Sigma_{i+1,j}$ is $\delta/4$ wide for any i, six strips $\Sigma_{i(j),j}, \ldots, \Sigma_{i(j)+5,j}$ are sufficient to fully contain Σ. Thus, each subset Q_j is paired with at most six subsets $P_{i(j)}, \ldots, P_{i(j)+5}$.

Since $\sum_{j=1}^{n'} \sum_{i=i(j)}^{i(j)+5} (|P_i| + |Q_j|) \le 6(m+n)$, TwinLFP can be solved in linear time.

Lemma 2: TwinLFP can be solved in linear time.

3. General Trees

Let T be a (general) tree. T can be rooted by selecting a node of T and making it the root. A rooted tree can be transformed into a binary tree by adding dummy nodes and edges so that the longest feasible path of the tree can be induced from the longest feasible path of the binary tree [9], [11]. Each dummy edge has length zero and $z_e = 0$.

From now on, we assume that T is a binary tree with n nodes. In T, each edge is either original (existed before the transformation) or dummy (added in the transformation). Note that each original edge has length one and each dummy edge has length zero. The *length* of a path $\pi(u, v)$ in T is defined to be the number of the original edges of $\pi(u, v)$.

If a node is deleted from T, then three subtrees T_1, T_2, T_3 (some may be empty) are left. A node is a *centroid* of T if its deletion from T leaves three subtrees such that $|T_i| \le |T|/2$ for $i = 1, 2, 3$. A tree can have at most two centroids and, if there are two centroids, they are adjacent [7]. Centroids of a tree can be found in linear time [8], [11].

Let g be the centroid of T (take either one when two centroids exist). Assume that $|T_1| \ge |T_2|$ and $|T_1| \ge |T_3|$. Let g' be the node of T_1 that is adjacent to g. The edge $\hat{e} = (g, g')$ is the *wire* of T. Deleting the wire, but not its end nodes, from T leaves two subtrees T', T'' such that $g \in T'$ and $g' \in T''$.

TreeLFP on T is recursively solved as follows:

(i) Divide T into T' and T'' by deleting the wire.

(ii) Recursively solve TreeLFP on T' and on T''.

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

After step (ii), we have the longest feasible path π' of T', and the longest feasible path π'' of T''. In step (iii), we need to find the longest feasible path π''' such that one of its end nodes is in T' and the other is in T'', and return the longest one of $\{\pi', \pi'', \pi'''\}$ as the longest feasible path of T.

To locate π''', we apply Lemma 2 to $T = T' \cup T''$, which is twin-rooted at $\hat{e} = (g, g')$. The sorted lists Λ' and Λ'' required by the algorithm can be obtained in $O(|T'|)$ and $O(|T''|)$ time, respectively, by the method of Kim [6].

One point we need to be careful about when applying the algorithm of Sect. 2 is $l(g, u)$: Every edge of T in Sect. 2 has length one, whereas each edge of T here has either length one (original edge) or length zero (dummy edge). Only the original edges have to be taken into account in computing $l(g, u)$.

Theorem 1: TreeLFP can be solved in optimal $O(n \log n)$ time.

Proof: Correctness of our TreeLFP algorithm is obvious from our explanation given so far. Let $A(n)$ be the time complexity of the algorithm on a size n tree T. Step (i) takes linear time, and step (iii) also takes linear time by Lemma 2. Then, $A(n) \leq A(n') + A(n'') + O(n)$, where $n' = |T'|$ and $n'' = |T''|$. By the definitions of a centroid and a wire, we have $\frac{1}{3}n \leq n', n'' \leq \frac{2}{3}n$. So, $A(n) = O(n \log n)$. An $\Omega(n \log n)$ lower bound was proved in [4]. □

References

[1] T. Bernholt, F. Eisenbrand, and T. Hofmeister, "Constrained minkowski sums: A geometric framework for solving interval problems in computational biology efficiently," Discret. Comput. Geom., vol.42, no.1, pp.22–36, 2009.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 2nd ed., MIT Press and McGraw-Hill, 2001.

[3] H. Gabow and R.E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," 15th ACM STOC, pp.246–251, 1983.

[4] 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-ACM Trans. Comput. Biol. Bioinform., vol.5, no.2, pp.281–290, 2008.

[5] S.K. Kim, "Optimal online and offline algorithms 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.

[6] S.K. Kim, "Optimal algorithms for finding density-constrained longest and heaviest paths in a tree," IEICE Trans. Inf. & Syst., vol.E93-D, no.11, pp.2989–2994, Nov. 2010.

[7] D.E. Knuth, The Art of Computer Programming, Fundamental Algorithms, 2nd ed., Addison-Wesley, 1973.

[8] 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.

[9] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs," Oper. Res. Lett., vol.19, pp.59–64, 1996.

[10] R.E. Tarjan, "Efficiency of a good but not linear set union algorithm," J. ACM, vol.22, no.2, pp.215–225, 1975.

[11] 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.