Continuous Bottleneck Tree Partitioning Problems

Nir Halman[1] and Arie Tamir[2]

December 6, 1999

Revised: April 16, 2002

Abstract: We study continuous partitioning problems on tree network spaces whose edges and nodes are points in Euclidean spaces. A continuous partition of this space into p connected components is a collection of p subtrees, such that no pair of them intersect at more than one point, and their union is the tree space. An edge-partition is a continuous partition defined by selecting $p-1$ cut points along the edges of the underlying tree, which is assumed to have n nodes. These cut points induce a partition into p subtrees (connected components). The objective is to minimize (maximize) the maximum (minimum) "size" of the components (the min-max (max-min) problem). When the size is the length of a subtree, the min-max and the max-min partitioning problems are NP-hard. We present $O(n^2 \log(\min(p, n)))$ algorithms for the edge-partitioning versions of the problem. When the size is the diameter, the min-max problems coincide with the continuous p-center problem. We describe $O(n \log^3 n)$ and $O(n \log^2 n)$ algorithms for the max-min partitioning and edge-partitioning problems respectively, where the size is the diameter of a component.

Keywords: Tree partitioning; Continuous p-center problems; Bottleneck problems, Parametric search

[1]Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel. halman@post.tau.ac.il

[2]Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel. atamir@post.tau.ac.il, corresponding author.

1 Introduction

There are numerous papers in the literature studying discrete bottleneck tree partitioning problems, defined by deleting $p-1$ edges from a given tree graph and looking at the set of p discrete subtrees induced by the partition. (Of course, in the discrete cases $p < n$.) For example the reader is referred to [2, 3, 11, 12, 13, 14, 17, 22, 23] for the most recent studies. In this paper we focus on continuous partitioning problems on the metric space of the continuum set of points on the edges of a tree in Euclidean space, induced by the edge lengths. The objective is to partition this space (a tree network space) into p connected components (closed subtrees), optimizing the maximum or minimum "size" of the components. Typically, the size will refer to the length or the diameter of the component. (Informally, the length of a component is the sum of the lengths of its edges, and its diameter is the length of a longest simple path of the component.) In a continuous partition we require that no pair of components intersect at more than one point, and the union of the components is the tree network space. We are aware of only a few papers dealing with continuous tree partitioning problems. For example, in the continuous p-center problem on a tree the objective is to split the above space into p connected components, minimizing the maximum of the diameter (radius) of the components. Polynomial algorithms solving this problem appear in [8, 9, 20]. On the other hand, if we wish to minimize (maximize) the maximum (minimum) length of the components the problem is NP-hard, since the Partition problem is a special case, [16]. (Consider the case of a tree consisting of $n-1$ leaves, each one of them connected by an edge to a central node, and let $p = 2$.) The recent paper by Agnetis, Mirchandani and Pacifici [1] considers the model of partitioning a tree space into p subsets, not necessarily connected, of equal length, while minimizing the maximum diameter of the subsets. They provide polynomial algorithms only for the cases where $p = 2$ and $p = 3$.

In view of the above NP-hardness result, in this paper we focus mainly on a class of continuous partitions, where the $p-1$ cut points, splitting the space into p components are restricted to be on the edges of the tree. (See the next section for an exact definition of a cut.) We call such partitions continuous edge-partitions. As it will be clear from the definition, not every partition is an edge-partition. (For example, splitting a five node star tree, with one center node connected to four leaves, into two components, each having exactly two leaves can be achieved by a continuous partition, but not by a continuous edge-partition. See Figure 1 at the end of Section 3.) Such a model has been recently discussed by Becker, Simeone and Chiang [4, 5, 6]. They study the continuous max-min tree edge-partitioning problem, where the objective is to maximize the minimum length of the components. Assuming rational data, the authors present an $O(n^2p^2 + np^3)$ algorithm to solve the model. We are unaware of papers dealing with the continuous min-max tree partitioning problem. We adapt the general approach in [8, 9], to improve upon the above results. Specifically, for any real data we present $O(n^2 \log(\min(p, n)))$ algorithms to solve both the continuous max-min and min-max tree length edge-partitioning problems.

We also consider partitioning and edge-partitioning bottleneck problems involving the diameters of the components. The min-max model, where the goal is to minimize the maximum of the diameters of the p components is the continuous p-center problem, for which an $O(n \log^2 n)$ algorithm is already known, [10, 20]. For the max-min problems where the goal is to maximize the minimum of the diameters of the p components, we obtain the following results. For the edge-partitioning version we derive an $O(n \log^2 n)$ algorithm, while the partitioning version is solved in $O(n \log^3 n)$ time.

2 Formulation of the continuous bottleneck tree edge-partitioning problems

Let $T = (V, E)$ be an undirected tree with node set $V = \{v_1, ..., v_n\}$ and edge set $E = \{e_2, ..., e_n\}$. Each edge e_j, $j = 2, 3, ..., n$, has a positive length l_j. We assume that T is embedded in the Euclidean plane. Each edge e_j is considered to be a closed interval of length l_j so that we can (uniquely) refer to its interior points by their distances (along the edge) from the two nodes of e_j. Let $A(T)$ denote the continuum set of points on the edges of T. We view $A(T)$ as a connected and closed set which is the union of $n-1$ intervals. (A pair of intervals may intersect only at a common node.) Let $P[v_i, v_j]$ denote the unique simple closed path in $A(T)$ connecting v_i and v_j. Suppose that the tree T is rooted at some distinguished node, say v_1. For each node v_j, $j = 2, 3, ..., n$, let $p(v_j)$, the *parent* of v_j, be the node $v \in V$, $v \neq v_j$, which is closest to v_j on $P[v_1, v_j]$. v_j is a *child* of $p(v_j)$. e_j is the edge connecting v_j with its parent $p(v_j)$. A node v_i is a *descendant* of v_j if v_j is on $P[v_i, v_1]$. V_j will denote the set of all descendants of v_j, and C_j will denote the set of all children of v_j. $T(V_j)$ will denote the subtree induced by V_j.

As noted above we refer to interior points on an edge by their distances along the edge from the two nodes of the edge. The edge lengths induce a distance function on $A(T)$. For any pair of points $x, y \in A(T)$, we let $d(x, y)$ denote the length of $P[x, y]$, the unique simple path in $A(T)$ connecting x and y. If x and y are on the same edge, $P[x, y]$ is called a *sub-edge* or a *partial edge*, and its length is $d(x, y)$. Generally, the path $P[x, y]$ is also viewed as a collection of edges and at most two sub-edges (partial edges), which are not edges. $P(x, y)$ will denote the open path obtained from $P[x, y]$ by deleting the points x, y, and $P(x, y]$ will denote the half open path obtained from $P[x, y]$ by deleting the point x.

Also, for any subset $Y \subseteq A(T)$, and x in $A(T)$ we define $d(x, Y) = d(Y, x)$

= Infimum $\{d(x, y) | y \in Y\}$. $A(T)$ is a metric space with respect to the above distance function.

A subset $Y \subseteq A(T)$ is called a *subtree* if it is closed and connected. A subtree Y is also viewed as a finite (connected) collection of partial edges (closed subintervals), such that the intersection of any pair of distinct partial edges is empty or is a point in V. We call a subtree discrete when all its leaves, (relative boundary points), are nodes of T. If Y is a subtree we define the *length* of Y, $l(Y)$, to be the sum of the lengths of its partial edges. We define the *diameter* of Y, $d(Y)$, to be the length of a longest (simple) path in Y.

A *continuous partition* of $A(T)$ into p components is a set of p closed subtrees, such that no pair of them intersect at more than one point, and their union is $A(T)$. We now define an edge-partition.

A *cut* or a *break* of the tree T is defined by a point, called a *cut point*, as follows: Suppose first that x is an interior point along some edge (v_i, v_j), i.e., its distances from both v_i, and v_j are positive. An *interior cut* at x is a splitting of the edge into two closed partial edges, $P[v_i, x]$ and $P[x, v_j]$. (Note that x is in both partial edges.) Such an interior cut divides the tree into two closed subtrees of positive length, intersecting at x only. We also define two *boundary cuts* or *node cuts* along this edge: The $[v_i, v_j]$-*cut* ($[v_j, v_i]$-*cut*) is the splitting of the edge into two parts, one consisting of the node v_i (v_j) only, and the other consisting of the (closed) edge (v_i, v_j). (Again, note that v_i (v_j) is in both parts of the split.) v_i (v_j) is called the *marker* of the cut, and the edge (v_i, v_j) is its *direction*. If the marker of the node cut is not at a leaf of the tree it also partitions the tree into two subtrees of positive length, intersecting at the marker only. Note that an interior cut is uniquely defined by an interior point of an edge, while a node cut is characterized by a node and an edge which is incident to that node. In particular, there are at most $deg(v_i)$ distinct cuts associated with a node v_i. ($deg(v_i)$ is the number of edges incident to v_i.) To illustrate consider the example in Figure 1. A continuous 2-partition splitting the tree into two subtrees, each having exactly two leaves is defined by the two subtrees induced respectively by the node sets $\{v_1, v_2, v_3\}$ and $\{v_1, v_4, v_5\}$. No edge-partition can achieve this splitting. To split the tree into the two subtrees induced by the node sets $\{v_1, v_2, v_3, v_4\}$ and $\{v_1, v_5\}$, consider the edge-partition defined by the boundary cut $[v_1, v_5]$-cut. (Here v_1 is the marker and $e_5 = (v_1, v_5)$ is the direction.)

Since $p-1$ distinct cuts induce a partition of T into p subtrees, it is more convenient to add an artificial cut so that the number of cuts will be equal to the number of subtrees. Thus, we augment the original tree T by an artificial node v_0, a super root, by connecting it to v_1 with an artificial edge of unit length. We will assume without loss of generality that we select p cut points and one of the cuts must be the $[v_1, v_0]$-cut. These p cuts induce p subtrees on the original tree T. (The artificial edge takes no part in the partition of the original tree T.) Let p be a positive integer, and let X_p be a set of p distinct cut points, such that no node v_i is the marker of more than $deg(v_i) - 1$ boundary cuts in X_p. X_p is called a *p-cut.* It defines a *continuous edge-partition* of $A(T)$ into p closed connected components (subtrees) of positive length, $\{T_1(X_p), ..., T_p(X_p)\}$. The intersection of each pair of components is either empty or consists of exactly one point, which is either a marker or an interior point of an edge. For convenience we will define a one-to-one correspondence between the cut points and the subtrees. Each subtree $T_j(X_p)$, $j = 1, ..., p$, will correspond to the (unique) closest point to the root v_1 in $T_j(X_p) \cap X_p$. We will also say that this cut point *determines* or induces the subtree $T_j(X_p)$.

Note that an edge-partition is a partition with the additional following property. If two distinct subtrees of the partition have a common node, then this node is a leaf of at least one of the subtrees.

In this paper we study bottleneck optimization problems defined by p-cuts. Specifically, for each p-cut X_p we consider the lengths of the components, $\{l(T_1(X_p)), ..., l(T_p(X_p))\}$.

In the *continuous max-min tree length edge-partitioning problem*, the objective is to find a p-cut X_p maximizing

$$lMin(X_p) = \min\{l(T_1(X_p)), ..., l(T_p(X_p))\}.$$

In the *continuous min-max tree length edge-partitioning problem*, the objective is to find a p-cut X_p minimizing

$$lMax(X_p) = \max\{l(T_1(X_p)), ..., l(T_p(X_p))\}.$$

As mentioned above, the continuous max-min tree length edge-partitioning problem has been recently discussed by Becker, Simeone and Chiang [4, 5, 6]. Assuming rational data, the authors present $O(n^2p^2 + np^3)$ algorithm to solve the model. We are unaware of papers dealing with the continuous min-max tree length edge-partitioning problem.

We now define the related problems involving the diameters of the subtrees in the partition, $\{d(T_1(X_p)), ..., d(T_p(X_p))\}$.

In the *continuous min-max tree diameter edge-partitioning problem* , the objective is to find a p-cut X_p minimizing

$$dMax(X_p) = \max\{d(T_1(X_p)), ..., d(T_p(X_p))\}.$$

This is the continuous p-center problem on a tree. (Using location theory terminology it corresponds to the case where both the demand set and the supply set are equal to $A(T)$ [9].) Polynomial algorithms solving this problem appear in [8, 9, 20]. The algorithm with the lowest known complexity is the $O(n \log^3 n)$ procedure by Megiddo and Tamir (1983) [20]. The complexity can be further improved to $O(n \log^2 n)$ by using the modification in Cole (1987) [10]. Note that the continuous p-center problem is actually defined as the partitioning (not necessarily edge-partitioning) problem, minimizing the maximum of the diameters of the components. However, it can easily be shown that for this model the two versions coincide.

In the *continuous max-min tree diameter edge-partitioning problem*, the objective is to find a p-cut X_p maximizing

$$dMin(X_p) = \min\{d(T_1(X_p)), ..., d(T_p(X_p))\}.$$

We also consider the partitioning version of the model, the *continuous min-max tree diameter partitioning problem*, where we look for a partition (not necessarily edge-partition) maximizing the minimum of the diameters. The two versions are not identical. We present $O(n \log^2 n)$ and $O(n \log^3 n)$ algorithms for the edge-partitioning and the partitioning versions respectively.

Remark: We note that due to the nature of the above four bottleneck objective functions, the maxima of $lMin(X_p)$ and $dMin(X_p)$, as well as the minima of $lMax(X_p)$ and $dMax(X_p)$ do exist. (This follows from the fact that even if we allow X_p to be a multiset, or contain a node v_i which is a marker of $deg(v_i)$ boundary cuts, there is always a p-cut X'_p which dominates X_p.)

For convenience we summarize the notation introduced above that will be used throughout the paper.

Notation:

$A(T)$: The embedding of the tree in the Euclidean plane.

v_1: The root of T.

$p(v_j)$: The parent of v_j.

e_j: The edge $(v_j, p(v_j))$ connecting v_j with $p(v_j)$.

l_j: The length of e_j.

C_j: The set of children of v_j.

V_j: The set of descendants of v_j.

$T(V_j)$: The subtree induced by V_j.

$P[x, y]$: The unique simple path connecting x and y.

$l(Y)$: The length of a subtree Y.

$d(Y)$: The diameter of a subtree Y.

X_p: A p-cut, a set of p cut points.

$lMin(X_p)$: Minimum length of the p subtrees induced by X_p.

$lMax(X_p)$: Maximum length of the p subtrees induced by X_p.

$dMin(X_p)$: Minimum diameter of the p subtrees induced by X_p.

$dMax(X_p)$: Maximum diameter of the p subtrees induced by X_p.

3 The continuous max-min tree length edge-partitioning problem

To solve this max-min edge-partitioning problem, we use a parametric approach.

For a positive real l, $l \leq l(T)$, we define $M(l)$ to be the maximum number of cuts possible, such that there exists an edge-partition with $M(l)$ cuts for which the length of each one of the induced $M(l)$ subtrees is greater than or equal to l. It follows that if $l_1 < l_2$, then $M(l_2) \leq M(l_1)$. Therefore, l_p^*, the solution value to the continuous max-min tree length edge-partitioning problem into p components is the largest value of l such that $M(l) \geq p$.

We will present two approaches to compute l_p^*. Both approaches yield polynomial algorithms, based on the following linear time algorithm to compute $M(l)$ for a given real l.

3.1 Algorithm 1: Computation of $M(l)$

Given a positive real l, $l \leq l(T)$, our objective is to compute $M(l) = \max\{q | \exists X_q, lMin(X_q) \geq l\}$. (Recall our supposition that one of the cuts is the artificial cut, the $[v_1, v_0]$-cut.)

For any real z, $\lfloor z \rfloor$, will denote the largest integer bounded above by z.

We use a bottom-up approach, starting with the leaves of the rooted tree. Recall that for each node v_i, C_i (V_i) denote the set of all children (descendants) of v_i and $T(V_i)$ denotes the subtree of T induced by V_i. Define $V' = V'(l) = \{v_i | l(T(V_i)) \geq l > l(T(v_j)), \forall v_j \in C_i\}$. (Since $l \leq l(T)$, V' is nonempty, and if v_i is in V', then there is no cut point in $T(V_j)$ for $v_j \in C_i$.) To initiate the algorithm consider T", the subtree of the original tree, induced by the root v_1 and all the children of the nodes in V'. (These children will be the leaves of T", and v_1 will be its root.) A node v_i is called a *cluster node* of a rooted tree, if all its children are leaves of this tree. For each cluster node of T", v_i, and $v_j \in C_i$ add $l(T(V_j))$ to the current length of the (leaf) edge (v_i, v_j) of T".

Algorithm 1

To compute $M(l)$ we start with the rooted tree T". In a generic iteration of the algorithm we select a cluster node v_i of the (current) tree. (Initially we start with T", and set $M(l) = 0$.) Let $\{v_{i(1)}, ..., v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster

If $l_{i(k)} < l$ for all $k = 1, ..., t$, define $A = \sum_{k=1}^{t} l_{i(k)}$. If $v_i = v_1$ delete all edges $(v_i, v_{i(k)})$ from the current tree, and go to Step 3. If $v_i \neq v_1$ go to Step 2.

Otherwise, for each $k = 1, ..., t$, define $n_k = \lfloor l_{i(k)}/l \rfloor$, and add n_k to $M(l)$. Reduce the length of the edge $(v_i, v_{i(k)})$ from $l_{i(k)}$ to $a_k = l_{i(k)} - n_k l$. (This accounts for adding n_k cut points on the edge, where the distance between adjacent points is exactly l.) If $l_{i(k)} = n_k l$ delete the edge $(v_i, v_{i(k)})$, from the current tree. Let x_i denote a cut point closest to v_i, amongst all $M(l)$ cut points that have been added so far. Define $A = \sum_{k=1}^{t} a_k$. If all child edges are deleted, i.e., $A = 0$, and $v_i = v_1$, go to Step 3. If all edges are deleted and $v_i \neq v_1$, repeat starting with a cluster of the updated tree.

Step 2: Deleting a cluster

Delete all remaining edges of the type $(v_i, v_{i(k)})$, $k = 1, ..., t$, from the current tree. If $v_i = v_1$ go to Step 3. If $v_i \neq v_1$ and $A \geq l$, add 1 to $M(l)$, (corresponding to the $[v_i, p(v_i)]$-cut). Repeat starting with a cluster of the updated tree. If $v_i \neq v_1$ and $A < l$, increase the length of the edge $(v_i, p(v_i))$ from l_i to $l_i + A$, and repeat starting with a cluster of the updated tree.

Step 3: Termination at the root v_1

If $A \geq l$ add 1 to $M(l)$, (corresponding to the $[v_1, v_0]$-cut). Stop and return the current value of $M(l)$. If $A < l$ consider the cut point x_1 which is the closest to v_1 amongst all $M(l)$ cut points that have been established. Replace x_1 by the $[v_1, v_0]$-cut. Stop and return the current value of $M(l)$. (The last case corresponds to replacing the subtree previously determined by x_1 by its union with the current remaining subtree (cluster) rooted at v_1.)

End of Algorithm 1.

In the next lemma we prove the validity of the above procedure to compute $M(l)$.

Lemma 3.1 *Algorithm 1 correctly computes $M(l)$.*

Proof: Using the (inductive) nature of the algorithm, it is sufficient to prove its validity for a tree consisting of a single cluster, rooted at v_i.

Consider such a cluster with leaves $\{v_{i(1)}, ..., v_{i(t)}\}$. Since $l \leq l(T)$, the sum of the lengths of the edges is at least l. If the length of each edge is smaller than l, there is only one cut, (the $[v_i, v_0]$-cut). Indeed, this is the cut generated by the algorithm.

Suppose that there is at least one edge of length greater than or equal to l. For each such edge $(v_i, v_{i(k)})$ compute $a_k = l_{i(k)} - n_k l$, where $n_k = \lfloor l_{i(k)}/l \rfloor$. Without loss of generality suppose that the edge $(v_i, v_{i(1)})$ is one of these edges and a_1 is the smallest of all these coefficients. Then it is easy to see that there is a set of $M(l)$ cuts, such that for each $k \neq 1$, $n'_k = n_k = \lfloor l_{i(k)}/l \rfloor$ of them are equally spaced on the edge $(v_i, v_{i(k)})$, and $n'_1 = n_1 - 1 = \lfloor l_{i(1)})/l \rfloor - 1$, of them are equally spaced on the edge $(v_i, v_{i(1)})$. (The first cut on each edge is at a distance l from the leaf, and the distance between consecutive cuts on the same edge is also l.) Therefore, if we trim the length of each edge $(v_i, v_{i(k)})$, $k = 1, ..., t$, by $n'_k l$, the remaining cluster (whose length is at least l), will require at least one and at most two additional cuts. One of the cuts is the $[v_i, v_0]$-cut. There is a second cut if and only if the length of the above remaining cluster is at least $2l$. If there is a second cut point it is necessarily on the edge $(v_i, v_{i(1)})$ at a distance of a_1 from v_i. It is easily checked that Algorithm 1 does exactly that.

■

It is easy to verify that it takes $O(n)$ time to compute the integer $M(l)$, for a given value of l. We also note that with the same complexity the algorithm also generates the set of $M(l)$ cuts. (The distance between consecutive interior cuts on an edge is equal to l. Therefore, we can output this set by specifying only the location of the cut on the edge closest to v_1, and the total number of cut points on this edge.)

3.2 Characterizing the optimal value

In the next theorem we characterize the optimal solution value l_p^*. This characterization, combined with Algorithm 1 will yield our first polynomial algorithm to compute l_p^*.

Theorem 3.2 *Let l_p^* be the solution value for the continuous max-min tree length edge-partitioning problem. Then there exists a discrete subtree T' of T, and an integer $q \leq p$, such that*

$$l_p^* = l(T')/q.$$

Proof: We prove the result by induction on the number of nodes in T. The result clearly holds if T consists of a single edge.

Let T be a general tree, and suppose that we apply the above bottom-up algorithm to compute $M(l_p^*)$, using v_1 as the root of T. Consider the first node cut selected by the algorithm. (Recall that the final cut, on the augmented edge connecting the root v_1 to the artificial node v_0, is a node cut. Hence the first node cut is well defined.)

Suppose that this cut is the q-th cut, and it is either the $[v_i, v_j]$-cut, for some child v_j of v_i, or it is the $[v_i, p(v_i)]$-cut. In the first case there is an integer $q' \leq q$ such that the length of the discrete subtree T_j^+, induced by v_i and V_j, is equal to $q' l_p^*$. (All the prior cuts are interior, and therefore the length of each subtree associated with one of the first q cuts is equal to l_p^*.)

Suppose that the cut is the $[v_i, p(v_i)]$-cut. Let $q" \leq q$ denote the total number of cuts in V_i, including the $[v_i, p(v_i)]$-cut. Consider the following two cases. First, suppose that the length of the subtree determined by this cut point is equal to l_p^*. In this case we have $l(T(v_i)) = q" l_p^*$, where $T(v_i)$ is the subtree induced by V_i.

Next suppose that the length of the subtree defined by the q-th cut is strictly greater than l_p^*. Since all prior cuts in V_i are interior, we can slightly perturb all of these $q" - 1$ cuts towards v_i, ensuring that the length of each one of the respective $q"$ subtrees is strictly greater than l_p^*. From the optimality of l_p^* for the problem defined on T, it now follows that $v_i \neq v_1$, and the optimal solution value to the continuous edge-partitioning problem (with $p - q"$ cuts), on the subtree induced by v_i and the nodes in $V - V_i$ must also be l_p^*. The result follows from the induction hypothesis. ∎

Theorem 3.2, combined with the procedure to compute $M(l)$ implies the following polynomial (but not strongly polynomial) algorithm. Suppose that each edge length of the tree is integer, and let K be the longest edge length. Then from the above result it follows that l_p^* is a rational number, where both numerator and denominator are bounded above by $p + (n-1)K$. Since l_p^* is the largest value of l such that $M(l) \geq p$, we can now directly apply a search over the rationals, as described in [21, 24], and find l_p^* in $O(n\log(p+nK))$. If we relax the integrality assumption, and suppose that each edge length is rational, where all the integer numerators and denominators are bounded above by an integer K, the total running time will increase to $O(n^2\log K + n\log p)$.

3.3 The parametric algorithm

We will next show how to obtain a strongly polynomial algorithm using the parametric approach in Megiddo (1979) [18]. We start by bounding the number of cuts on each edge in an optimal solution.

Theorem 3.3 *Let X_p be an optimal solution to the continuous max-min tree length edge-partitioning problem. For each edge $e_j \in E$, let $n(j)$ be the number of cuts in X_p which are on e_j. Then,*

$$(l_j/l(T))(p-(n-1))-1 \leq n(j) \leq (l_j/l(T))(p+(n-1))+1.$$

Proof: Let l be a positive real. Consider an edge $e_j \in E$. Then it is easy to see that $l_j/l+1$ is an upper bound on the number of cuts that can be established on e_j, such that the length of each partial edge (subtree) is at least l. Similarly, it is easy to see that we can establish $\lfloor (l_j-l)/l\rfloor+1 = \lfloor l_j/l\rfloor$ cuts on each edge e_j, such that the length of each one of the induced subtrees will be at least l. Therefore, $M(l) \geq \sum_{e_j\in E}\lfloor l_j/l\rfloor \geq \sum_{e_j\in E}(l_j/l-1)$.

Combined with the above upper bounds we obtain,

$$l(T)/l-(n-1) \leq M(l) \leq l(T)/l+(n-1).$$

In particular, for l_p^* we obtain, $p \leq M(l_p^*) \leq (l(T)/l_p^*)+(n-1)$. Equivalently,

$$(p-(n-1))/l(T) \leq 1/l_p^*. \tag{1}$$

Next define $l' = l(T)/(p+(n-1))$. From the above we obtain $M(l') \geq l(T)/l'-(n-1) = p$. Since l_p^* is the largest l such that $M(l)\geq p$, it follows that

$$1/l_p^* \leq 1/l' = (p + (n-1))/l(T). \tag{2}$$

Finally, for each edge e_j we have

$$l_j/l_p^* - 1 \leq n(j) \leq l_j/l_p^* + 1.$$

Substituting the upper and lower bounds from (1)-(2) above yields,

$$(l_j/l(T))(p - (n-1)) - 1 \leq n(j) \leq (l_j/l(T))(p + (n-1)) + 1.$$

■

We now have all the ingredients necessary to apply the general parametric approach of Megiddo [18, 19] to obtain an $O(n^2 \log(\min(p, n)))$ algorithm for the solution of the max-min tree length edge-partitioning model for any real data. (We note that a similar framework is used in [8] to obtain the first polynomial algorithm for the continuous p-center and p-dispersion problems on tree graphs. Faster algorithms for these center/dispersion models appear in [20].)

The approach is to apply Algorithm 1 parametrically, using l as the single parameter, to compute $M(l_p^*)$ without specifying the value of l_p^* a priori. Note that for a fixed value of the parameter, Algorithm 1 is executed in $O(n)$ steps. At each step we possibly trim the lengths of some edges of the cluster by an integer multiple of the parameter l, and perform some additions and comparisons with the updated lengths of the edges. Imagine that we start the algorithm without specifying a value of the parameter l. The parameter is restricted to some interval which is known to contain the optimal value l_p^*. (Initially, we may start with the interval $[0, l(T)]$.) As we go along, at each step of the algorithm we update and shrink the size of the interval, ensuring that it includes the optimal value.

Before we formally present the parametric version of Algorithm 1 which computes the optimal value l_p^*, consider the preprocessing phase of Algorithm

1 above, where the set of nodes $V'(l)$ is computed. Since this set depends on the value of the parameter l, which is not specified, we need to determine what is $V'(l_p^*)$ without knowing l_p^*. We first use a bottom-up approach and compute in $O(n)$ time the set $L(T) = \{l(T(V_i))|i = 1, ..., n\}$. Next, using Algorithm 1 above, we apply a binary search on the set $L(T)$ to identify the largest (smallest) element of the set, say l_-, (l_+) such that $M(l_-) \geq p$ ($M(l_+) < p$). We conclude that $l_- \leq l_p^* < l_+$. To check whether l_p^* is actually bigger than l_-, we continue with the hypothesis that $l_- < l_p^* < l_+$. Specifically, with this supposition, we know that $V'(l_p^*) = V'(l_+)$, and we can construct the subtree T" (for the parameter value $l = l_p^*$), and turn to Step 1 of Algorithm 1 without knowing l_p^*.

Next we need to compare the lengths of the cluster edges with an unspecified value of the parameter in the open interval (l_-, l_+). We resolve all these comparisons simultaneously by applying a binary search, (using Algorithm 1), to identify the largest (smallest) element of the set, $\{l_{i(1)}, ..., l_{i(t)}\}$, say l', (l'') such that $M(l') \geq p$ ($M(l'') < p$). We conclude that $l' \leq l_p^* < l''$. We update the bounds by setting $l^- = \max(l', l_-)$ and $l^+ = \min(l'', l_+)$. $l^- \leq l_p^* < l^+$. Again to check whether l_p^* is actually bigger than l^-, we continue to the trimming phase with the hypothesis that $l^- < l_p^* < l^+$, and proceed along these lines. Following is a formal description of the parametric algorithm.

Algorithm 1: Parametric version

For $i = 1, ..., n$, define $n_i' = (l_i/l(T))(p-(n-1))-1$ and $n_i'' = (l_i/l(T))(p+(n-1))+1$.

In a generic iteration of the algorithm we select a cluster node v_i of the (current) tree. (Initially we start with T", and set $l^- = l_-$, $l^+ = l_+$, where T", l_- and l_+ are defined above. Also set $l_j(l) = l(T(V_j))$ for any leaf v_j of T", and $l^- \leq l \leq l^+$.) Let $\{v_{i(1)}, ..., v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster

For $k = 1, ..., t$, let $l(k)$ be defined by the solution to the linear equation $l_{i(k)}(l) = l$. Using Algorithm 1 above, apply a binary search on the set $\{l(1), ..., l(t)\}$ to identify the largest (smallest) element of the set, say l', (l'') such that $M(l') \geq p$ ($M(l'') < p$). (If $M(l(k)) \geq p$ for all $k = 1, ..., t$, define $l'' = L(T)$, and if $M(l(k)) < p$ for all $k = 1, ..., t$, define $l' = 0$.) Update the bounds on l_p^* by setting $l^- = \max(l', l^-)$ and $l^+ = \min(l'', l^+)$. ($l^- \leq l_p^* < l^+$.)

If $l_{i(k)}(l^-) \leq l^-$ for all $k = 1, ..., t$, define $A(l) = \sum_{k=1}^{t} l_{i(k)}(l)$. If $v_i = v_1$ delete all edges $(v_i, v_{i(k)})$ from the current tree, and go to Step 3. If $v_i \neq v_1$ go to Step 2.

Otherwise, (i.e., $l_{i(q)}(l^-) > l^-$ for some $q = 1, ..., t$), for each $k = 1, ..., t$, define

$$L'(k) = \{l | \exists q, q \in \{1, ..., p\}, n'_{i(k)} \leq q \leq n''_{i(k)}, l_{i(k)}(l) = ql\}.$$

Using Algorithm 1 above, apply a binary search on the set $L'(k)$ to identify the largest (smallest) element of the set, say l', (l'') such that $M(l') \geq p$ ($M(l') < p$). (If $M(l(k)) \geq p$ for all $k = 1, ..., t$, define $l'' = L(T)$, and if $M(l(k)) < p$ for all $k = 1, ..., t$, define $l' = 0$.) Update the bounds on l_p^* by setting $l^- = \max(l', l^-)$ and $l^+ = \min(l'', l^+)$. ($l^- \leq l_p^* < l^+$.)

Define $n_k = \lfloor l_{i(k)}(l^+)/l^+ \rfloor$. Reduce the length of the edge $(v_i, v_{i(k)})$ by setting $l_{i(k)}(l) = l_{i(k)}(l) - n_k l$. If $l_{i(k)}(l) = 0$ delete the edge $(v_i, v_{i(k)})$, from the current tree.

Define $A(l) = \sum_{k=1}^{t} l_{i(k)}(l)$. If all child edges are deleted, i.e., $A(l) = 0$, and $v_i = v_1$, go to Step 3. If all edges are deleted and $v_i \neq v_1$, repeat starting with a cluster of the updated tree.

Step 2: Deleting a cluster

Delete all remaining edges of the type $(v_i, v_{i(k)})$, $k = 1, ..., t$, from the current tree. If $v_i = v_1$ go to Step 3. Otherwise, ($v_i \neq v_1$), let l^* be the solution to the linear equation $A(l) = l$. Using Algorithm 1 compute $M(l^*)$, and determine whether $l^* \leq l_p^*$ or $l^* > l_p^*$. In the former case, update the lower bound by setting $l^- = \max(l^-, l^*)$, increase the length of the edge $(v_i, p(v_i))$ by setting $l_i(l) = l_i + A(l)$, and repeat starting with a cluster of the updated tree. If $l^* > l_p^*$, update the upper bound by setting $l^+ = \min(l^+, l^*)$, and repeat starting with a cluster of the updated tree.

Step 3: Termination at the root v_1

Let l^* be the solution to the linear equation $A(l) = l$. Using Algorithm 1 compute $M(l^*)$, and determine whether $l^* \leq l_p^*$ or $l^* > l_p^*$. In the former case, update the lower bound by setting $l^- = \max(l^-, l^*)$, and declare $l_p^* = l^-$. If $l^* > l_p^*$, update the upper bound by setting $l^+ = \min(l^+, l^*)$, and declare $l_p^* = l^-$.

End of Algorithm 1: Parametric version.

The validity of the above algorithm follows directly from the general analysis in [18]. The most important observation is that for all values of l in the interior of the final interval (l^-, l^+), the value of $M(l)$ is fixed and smaller than p. Moreover, for all these values of l, Algorithm 1, which computes $M(l)$, will execute exactly the same sequence of operations, e.g., each comparison will result in the same answer, regardless of the specific value of l.

To analyze the complexity of the above algorithm note that in a generic step of the (parametric) version of Algorithm 1, we select a cluster node v_i of the current tree. The lengths of all the edges connecting v_i to its children are linear functions of l. (Initially, the length of each edge e_j is l_j, a constant independent of l.) Consider for example, a child (leaf) $v_{i(k)}$. $l_{i(k)}(l)$, the (updated) length of the edge $e_{i(k)} = (v_i, v_{i(k)})$ is linear in l. Now we need to compute $n_{i(k)}(l) = \lfloor l_{i(k)}(l)/l \rfloor$, the number of cuts to be used on the edge $e_{i(k)}$.

Although l is not specified, and the optimal value l_p^* is obviously unknown at this stage, we can use Theorem 3.3 to conclude that for all relevant values of the parameter

$$n'_{i(k)} \leq n_{i(k)}(l) \leq n''_{i(k)}.$$

There are only $\min(p, n''_{i(k)} - n'_{i(k)}) = O(\min(p, n))$ possible values that the integer function $n_{i(k)}(l)$ can take on. They correspond to the critical values of the parameter l, which satisfy the linear equations $l_{i(k)}(l) = ql$, where q is an integer between $n'_{i(k)}$ and $n''_{i(k)}$.

We now apply a binary search on this set of critical values of l to identify a consecutive pair of values bounding the optimal value l^*_p. (Note that the binary search amounts to computing $M(l)$ for $O(\log(\min(p, n)))$ critical values of l.) To conclude, in $O(n \log(\min(p, n)))$ time we identify an interval, containing l^*_p, such that either l^*_p is the left endpoint of the interval, or for any l in the interior of this interval $n_{i(k)}(l) = n_{i(k)}(l^*_p)$. We can now trim the length of the edge $e_{i(k)}$ by the linear factor $n_{i(k)}(l^*_p)l$. We apply the trimming to all edges of the cluster, and downsize the updated interval of the parameter l.

To complete the processing of the cluster, we now need to add the linear functions corresponding to the (trimmed) lengths of its edges, and compare the sum with l. (See Algorithm 1.) We get a critical value of l, say l^*, for which the two terms are equal. We compute $M(l^*)$ to find whether $l^*_p < l^*$ or $l^*_p \geq l^*$. With this information we can proceed with the algorithm without having to specify a value (within the current valid interval) of the parameter.

If $t = t(v_i)$ denotes the number of children of the cluster node v_i, the total effort to process the cluster in the parametric algorithm is $O(t(v_i)n \log(\min(p, n)))$. Since the total number of children of all nodes is n, the total time of the parametric algorithm is $O(n^2 \log(\min(p, n)))$.

Theorem 3.4 *The continuous max-min tree length edge-partitioning problem can be solved in $O(n^2 \log(\min(p, n)))$ time.*

We note in passing that for certain tree topologies the above algorithm can be sped up. For example, if the depth of the tree (maximum number of nodes on a simple path) is k, then by applying the ideas in [19], the algorithm can be implemented in $O(kn \log n)$ time.

A possible approach to reduce the above $O(n^2 \log(\min(p, n)))$, bound is to parallelize Algorithm 1. Specifically, if there is a parallel algorithm which computes $M(l)$ in $O(poly(\log n))$ time using $O(n)$ processors, we can apply the ideas in [19] to design an $O(n\ poly(\log n))$ serial algorithm to compute l_p^*.

3.4 An Example

Consider the example in Figure 1. Let $l = 2$. To implement Algorithm 1, note that $V'(l) = \{v_1\}$, and therefore $T" = T$. We then trim the edges (v_1, v_3) and (v_1, v_5) by introducing two cut points at a distance of 2 from v_3 and v_5 respectively. (The edge (v_1, v_3) is deleted and v_1, the marker of the $[v_1, v_3]$-cut, is defined as x_1.) After the trimming we have $A = 3$. Since $A > 2$, we add the $[v_1, v_0]$-cut to conclude that $M(2) = 3$. (There is one interior cut on the edge (v_1, v_5) and two boundary cuts, the $[v_1, v_3]$-cut and the $[v_1, v_0]$-cut.)

To illustrate the parametric version consider the 3-edge-partitioning problem, i.e., $p = 3$. We first need to determine $V'(l_3^*)$. Since $L(T) = \{0, 7\}$, we trivially conclude that $l_- = 0$, $l_+ = 7$, $0 < l_3^* < 7$, and $V'(l_3^*) = \{v_1\}$. We start at Step 1, and let $i(k) = k+1$ for $k = 1, 2, 3, 4$. Initially, $l_{i(1)}(l) = l_{i(3)}(l) = 1$, $l_{i(2)}(l) = 2$ and $l_{i(4)}(l) = 3$. We obtain $l(1) = l(3) = 1$, $l(2) = 2$ and $l(4) = 3$. By Algorithm 1 we compute $M(1) = 7$, $M(2) = 3$ and $M(3) = 2$. We conclude that $l^- = 2$, $l^+ = 3$ and $2 \leq l_2^* < 3$. We observe that $l_{i(k)}(l^-) \leq l^-$ for $k = 1, 2, 3$, and $l_{i(4)}(l^-) = 3 > 2 = l^-$.

Next we obtain $L'(1) = L'(3) = \{1, 1/2, 1/3\}$, $L'(2) = \{2, 1, 2/3\}$ and $L'(4) = \{3, 3/2, 1\}$. Thus, the new critical values of l for which we need to know $M(l)$ are $\{1/3, 1/2, 2/3, 3/2\}$. However, since all these values are smaller than l^- they are ignored. We proceed by calculating $n_k = 0$ for

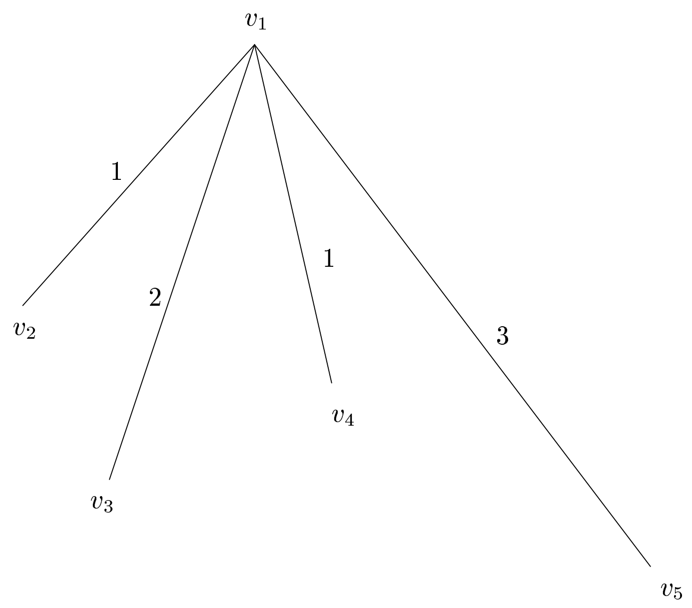

Figure 1: Example

$k = 1, 2, 3$, and $n_4 = 1$. The trimmed edge lengths, (parameterized over the interval (2,3)), are now $l_{i(1)}(l) = l_{i(3)}(l) = 1$, $l_{i(2)}(l) = 2$, and $l_{i(4)}(l) = 3 - l$. We set $A(l) = 7 - l$, and go to Step 3.

We now obtain the new critical value $l^* = 3.5$, which is the solution to the equation $A(l) = l$. Since $l^+ = 3 < 3.5$, we can ignore this critical value, and conclude that the optimal value is $l_3^* = l^- = 2$.

4 The continuous min-max tree length edge-partitioning problem

First we note that the max-min and min-max tree length problems can have distinct optimal solutions. Consider, for example, the tree in Figure 1, and change the length of the edge (v_1, v_3) to 1. Let $p = 3$. The unique solution to the max-min problem is attained by the selecting the $[v_1, v_0]$-cut, the $[v_1, v_5]$-cut, and the midpoint of the edge (v_1, v_5) as the third cut point. This solution is not optimal for the min-max model. The optimal solution value to the later problem is 2.5. It is attained by selecting the $[v_1, v_0]$-cut, the $[v_1, v_2]$-cut, and the point x, on (v_1, v_5), satisfying $d(x, v_1) = .5$, as the third cut point.

To solve the min-max model we can use exactly the same approach as in the previous section. For the sake of brevity we only present the necessary ingredients and results needed for the implementation, but skip the description of the algorithms. We conclude that the time needed to solve the continuous min-max tree edge-partitioning problem is also $O(n^2 \log(\min(p, n)))$.

4.1 Algorithm 2: Computation of $m(l)$

Given a positive real l, $l \leq l(T)$, our objective is to compute $m(l) = \min\{q | \exists X_q, lMax(X_q) \leq l\}$, the minimum number of cuts on the edges of the tree needed to ensure that the length of each one of the subtrees induced by this partition is at most l. (Recall our supposition that one of the cuts is the artificial cut, the $[v_1, v_0]$-cut.) We note that $m(l)$ is monotone. If we let L_p^* denote the optimal solution value to the min-max model, then L_p^* is the smallest value of l such that $m(l) \leq p$.

As above we use a bottom-up approach, starting with the leaves of the rooted tree.

Algorithm 2

In a generic iteration of the algorithm we select a cluster node v_i of the (current) tree. (Initially we start with T, and set $m(l) = 0$.) Let $\{v_{i(1)}, ..., v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster

For each $k = 1, ..., t$, define $n_k = \lfloor l_{i(k)}/l \rfloor$, and add n_k to $m(l)$. Reduce the length of the edge $(v_i, v_{i(k)})$ from $l_{i(k)}$ to $a_k = l_{i(k)} - n_k l$. (This accounts for adding n_k cut points on the edge, where the distance between adjacent points is exactly l.) If $l_{i(k)} = n_k l$ delete the edge $(v_i, v_{i(k)})$, from the current tree. Define $A = \sum_{k=1}^{t} a_k$. If $v_i = v_1$, go to Step 3. If all edges are deleted, $(A = 0)$, and $v_i \neq v_1$, repeat starting with a cluster of the updated tree.

Step 2: Deleting a cluster

If $A = l$, delete all remaining edges of the type $(v_i, v_{i(k)})$, $k = 1, ..., t$, from the current tree. Add 1 to $m(l)$, (corresponding to the $[v_i, p(v_i)]$-cut), and repeat starting with a cluster of the updated tree.

If $A < l$, delete all remaining edges of the type $(v_i, v_{i(k)})$, $k = 1, ..., t$, from the current tree. Increase the length of the edge $(v_i, p(v_i))$ from l_i to $l_i + A$, and repeat starting with a cluster of the updated tree.

If $A > l$, find q, and the subset V_i' of children of v_i, corresponding to the q smallest non-zero elements in the multiset $\{a_k | k = 1, ..., t\}$, such that the sum of these q smallest elements, denoted by B, is at most l, and the sum of the smallest nonzero $q + 1$ elements is greater than l. For each node $v_{i(k)} \notin V_i'$, such that $a_k > 0$, delete the edge $(v_i, v_{i(k)})$, and add 1 to $m(l)$. (Note that the particular definition of edge-partitioning implies that a cut must be used for each child not in V_i'.) For each node $v_{i(k)} \in V_i'$, delete the edge $(v_i, v_{i(k)})$. Increase the length of the edge $(v_i, p(v_i))$ from l_i to $l_i + B$, and repeat starting with a cluster of the updated tree.

Step 3: Termination at the root v_1

If $A = 0$, stop and return the current value of $m(l)$.

If $0 < A \leq l$, add 1 to $m(l)$, (corresponding to the $[v_1, v_0]$-cut). Stop and return the current value of $m(l)$.

If $A > l$, find q, and the subset V_1' of children of v_1, corresponding to the q smallest non-zero elements in the multiset $\{a_k | k = 1, ..., t\}$, such that the sum of these q smallest elements, denoted by B, is at most l, and the sum of the smallest nonzero $q + 1$ elements is greater than l. Let $t' = |\{k | a_k > 0, k = 1, ..., t\}|$. Add $t' - q + 1$ to $m(l)$. Stop and return the current value of $m(l)$.

End of Algorithm 2.

We note that the running time of the above algorithm is linear, since the time to process a cluster is proportional to its number of nodes. (We can apply the linear time median finding algorithm in [7] successively to find the term B defined above.)

In the next lemma we prove the validity of Algorithm 2.

Lemma 4.1 *Algorithm 2 correctly computes* $m(l)$.

Proof: Again, from the inductive nature of the algorithm it is sufficient to prove its validity for a tree consisting of a cluster rooted at some node v_i.

Consider such a cluster with leaves $\{v_{i(1)}, ..., v_{i(t)}\}$. Using the notation in Algorithm 2, it is easy to verify that for each $k = 1, ..., t$, there will be at least n_k cuts on the edge $(v_i, v_{i(k)})$. Moreover, the cuts on a given edge are equally spaced, with the first one being at a distance of l from the leaf. Hence, we can assume without loss of generality that a_k, the length of $(v_i, v_{i(k)})$, $k = 1, ..., t$, is less than l.

Let $A = \sum_{k=1}^{t} a_k$. If $A = 0$, there are no more cuts. If $0 < A \leq l$, there is one cut, i.e., the $[v_i, v_0]$-cut. This is exactly what Algorithm 2 does.

Suppose that $A > l$. (We assume without loss of generality that $a_t \geq a_{t-1} ... \geq a_1 > 0$.) Then there are $m(l)$ cuts, $m(l) > 1$, and one of them is the $[v_1, v_0]$-cut. Moreover, since the length of an edge is smaller than l, we can assume without loss of generality that if there is a cut on an edge $(v_i, v_{i(k)})$, it is the $[v_i, v_{i(k)}]$-cut.

Finally we note that if there is no cut on the longest edge, $(v_i, v_{i(t)})$, the optimal solution value $m(l)$ is not affected if we replace one of the $m(l) - 1$ cuts on the edges of the cluster by the $[v_i, v_{i(t)}]$-cut. Thus, we can assume without loss of generality that the $m(l)$ cuts are: the $[v_i, v_0]$-cut, and the $[v_i, v_{i(k)}]$-cut, for all $k = t, t-1, ..., t-(m(l)-2)$.

It is easy to verify that Algorithm 2 selects exactly these cuts. ■

Theorem 4.2 *Let* L_p^* *be the solution value for the continuous min-max tree length edge-partitioning problem. Then there exists a discrete subtree* T' *of*

T, and an integer $q \leq p$, such that

$$L_p^* = l(T')/q.$$

Proof: The proof is very similar to that of Theorem 3.2. ∎

As mentioned in the previous section, using the above theorem and Algorithm 2, we can now directly apply the search in [21, 24] to find L_p^* in polynomial time. We can also obtain a strongly polynomial time algorithm of complexity $O(n^2 \log(\min(p, n)))$ time by mimicking the approach in the previous section, and designing a parametric version of Algorithm 2. For the sake of brevity we skip the details.

Theorem 4.3 *The continuous min-max tree length edge-partitioning problem can be solved in $O(n^2 \log(\min(p, n)))$ time.*

5 Continuous bottleneck models involving the component diameters

We have already illustrated above that the bottleneck continuous partitioning problems, involving the lengths of the components are NP-hard, while the respective edge-partitioning problems are polynomially solvable.

We will show that if we use the diameter, instead of the length, as a measure, both the partitioning and the edge-partitioning problems are solvable in polynomial time.

We start with the min-max models, defined as follows. Find a partition (edge-partition) of the tree space into p subtrees minimizing the maximum diameter of the subtrees. The partitioning version is mentioned above, and it is known as the continuous p-center problem on a tree. An efficient $O(n \log^2 n)$ algorithm can be found in [10, 20]. It is known [9] that the optimal value of this problem, D_p^*, is of the form $D_p^* = d(v_i, v_j)/q$, where v_i, v_j are a pair of leaf nodes, and q is an integer bounded above by p. It can easily be shown that there is an optimal solution to the p-center problem, where the partition is actually an edge-partition. Therefore, the optimal solution values of the two versions are identical.

Turning to the max-min models, the objective is to find a partition (edge-partition) of the tree space into p subtrees maximizing the minimum diameter of the subtrees. Unlike the min-max model, simple examples illustrate that the two versions of this max-min problem are not identical. Consider, for example, the tree in Figure 1, and change the length of edge (v_1, v_5) to 2. For $p = 2$, the solution values to the partition and the edge-partition problems are 3 and 2 respectively.

We claim that both versions can be solved in subquadratic time using the method in [10, 20].

5.1 Maximizing the minimum diameter using edge-partitioning

Given a positive real d, $d \leq d(T)$, our objective is to compute $N(d) = \max\{q | \exists X_q, dMin(X_q) \geq d\}$, the maximum number of cuts on the edges of the tree needed to ensure that the diameter of each one of the subtrees induced by this partition is at least d. (Recall our supposition that one of the cuts is the artificial cut, the $[v_1, v_0]$-cut.)

We use a bottom-up approach, starting with the leaves of the rooted tree.

Algorithm 3

In a generic iteration of the algorithm we select a cluster node v_i of the (current) tree. (Initially we start with T, and set $N(d) = 0$.) Let $\{v_{i(1)}, ..., v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster

For each $k = 1, ..., t$, define $n_k = \lfloor l_{i(k)}/l \rfloor$, and add n_k to $N(d)$. Reduce the length of the edge $(v_i, v_{i(k)})$ from $l_{i(k)}$ to $a_k = l_{i(k)} - n_k l$. (This accounts for adding n_k cut points on the edge, where the distance between adjacent points is exactly l.) If $l_{i(k)} = n_k l$ delete the edge $(v_i, v_{i(k)})$, from the current tree.

Let x_i denote a cut point closest to v_i, amongst all $N(d)$ cut points that have been added so far.

If $v_i = v_1$, go to Step 3. If all edges are deleted and $v_i \neq v_1$, repeat starting with a cluster of the updated tree.

Step 2: Deleting a cluster

Let $S = \{k | k = 1, ..., t, a_k > 0\}$, and $A = \{a_k | k \in S\}$. (A is a multiset.) Define B_1 and B_2 to be the largest and the second largest elements in A respectively. (If $|S| = 1$ define $B_2 = 0$.) If $B_1 + B_2 \geq d$, delete all remaining edges of the type $(v_i, v_{i(k)}), k = 1, ..., t$, from the current tree. Add 1 to $N(d)$, (corresponding to the $[v_i, p(v_i)]$-cut). Repeat starting with a cluster of the updated tree.

If $B_1 + B_2 < d$, delete all remaining edges of the type $(v_i, v_{i(k)}), k = 1, ..., t$, from the current tree. Increase the length of the edge $(v_i, p(v_i))$ from l_i to $l_i + B_1$, and repeat starting with a cluster of the updated tree.

Step 3: Termination at the root v_1

Let $S = \{k | k = 1, ..., t, a_k > 0\}$, and $A = \{a_k | k \in S\}$. Define B_1 and B_2 to be the largest and the second largest elements in A respectively. (If $|S| = 0$ define $B_1 = B_2 = 0$ and if $|S| = 1$ define $B_2 = 0$.) If $B_1 + B_2 \geq d$, delete all remaining edges of the type $(v_i, v_{i(k)}), k = 1, ..., t$, from the current tree. Add 1 to $N(d)$, (corresponding to the $[v_1, v_0]$-cut). Stop and return the current value of $M(l)$.

If $B_1 + B_2 < d$, delete all remaining edges of the type $(v_i, v_{i(k)}), k = 1, ..., t$, from the current tree.

Consider the cut point x_1 which is the closest to v_1 amongst all $N(d)$ cut points that have been established. Replace x_1 by the $[v_1, v_0]$-cut. Stop and return the current value of $N(d)$. (The last case corresponds to replacing the subtree previously determined by x_1 by its union with the current remaining subtree (cluster) rooted at v_1.)

End of Algorithm 3.

The validity of Algorithm 3 follows from arguments similar to those used in the proof of Lemma 3.1. We skip the details. It is easy to see that the running time of Algorithm 3 is $O(n)$.

The next result characterizes the solution value.

Theorem 5.1 *Let d_p^* be the optimal solution value of the continuous max-min tree diameter edge-partitioning problem. Then there exists a pair of nodes v_i, v_j, and an integer $q \leq p$, such that*

$$d_p^* = d(v_i, v_j)/q.$$

Proof: We prove the result by induction on the number of nodes in T. The result clearly holds if T consists of a single edge.

Let T be a general tree, and suppose that we apply the above bottom-up algorithm to compute $N(d_p^*)$, using v_1 as the root of T. Consider the first node cut selected by the algorithm. (Recall that the final cut, on the augmented edge connecting the root v_1 to the artificial node v_0, is a node cut. Hence the first node cut is well defined.)

Suppose that this cut is the q-th cut, and it is either the $[v_i, v_j]$-cut for some child v_j of v_i, or it is the $[v_i, p(v_i)]$-cut. Let q' denote the total number of cuts in V_i. In the first case there is a leaf node $v_k \in V_j$ such that $d(v_k, v_i) = q'' d_p^*$ for some integer $q'' \leq q'$.

If the cut is the $[v_i, p(v_i)]$-cut, consider the following two cases. First, suppose that the diameter of the subtree defined by this cut is equal to d_p^*. In this case we have a pair of leaves in V_i, say v_k, v_s, such that $d(v_k, v_s) = d(v_k, v_i) + d(v_i, v_s) = q'' d_p^*$, for some $q'' \leq q'$

Next suppose that the diameter of the subtree defined by the cut is strictly greater than d_p^*. Since all prior cuts in V_i are interior, we can slightly perturb all of these $q' - 1$ cuts towards v_i, ensuring that the diameter of each one of the respective q' subtrees is strictly greater than d_p^*. From the optimality of d_p^* for the problem defined on T, it now follows that $v_i \neq v_1$, and the optimal solution value to the continuous edge-partitioning problem (with $p-q'$ cuts), on the subtree induced by v_i and the nodes in $V - V_i$ must also be d_p^*. The result follows from the induction hypothesis. ∎

We note in passing that unlike the min-max version of the model, discussed above, (the p-center problem), where the solution value D_p^* satisfies $D_p^* = d(v_i, v_j)/q$ for some pair of leaf nodes, the pair of nodes defining d_p^* are not necessarily leaves of the tree. To illustrate, consider the example of a single cluster consisting of three edges of lengths $2, 3/2, 3/2$ and set $p = 2$.

With the representation of the optimal solution value in Theorem 5.1, and the linear time procedure to compute $N(d)$, we can now directly apply the algorithmic framework in [10, 20], to obtain an $O(n \log^2 n)$ algorithm for the problem of maximizing the minimum diameter of an edge-partition of p subtrees. We also note in passing that when p is relatively small, i.e., $p = o(\log n)$, we can improve the complexity to $O(pn \log n)$ by using the search procedures in [15].

Theorem 5.2 *The continuous max-min tree diameter edge-partitioning problem can be solved in $O(n \log^2 n)$ time.*

5.2 Maximizing the minimum diameter using partitioning

For each real d we define $N'(d)$ to be the maximum number of subtrees defined by a partition, such that the diameter of each one of them is at least d. The following is a bottom-up algorithm (similar to Algorithm 3) to compute $N'(d)$. (Since the concept of a cut point has been defined with respect to edge-partitioning only, we now define the concept of a root point for general partitions. Each subtree of a partition will be associated with the closest point of the subtree to the root of the tree v_1. This point is called the *root point* of the subtree.)

Algorithm 4

In a generic iteration of the algorithm we select a cluster node v_i of the (current) tree. (Initially we start with T, and set $N'(d) = 0$.) Let $\{v_{i(1)}, ..., v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster

For each $k = 1, ..., t$, define $n_k = \lfloor l_{i(k)}/l \rfloor$, and add n_k to $N'(d)$. Reduce the length of the edge $(v_i, v_{i(k)})$ from $l_{i(k)}$ to $a_k = l_{i(k)} - n_k l$. (This accounts for adding n_k root points on the edge, where the distance between adjacent points is exactly l.) If $l_{i(k)} = n_k l$ delete the edge $(v_i, v_{i(k)})$, from the current tree.

Let x_i denote a root point closest to v_i, amongst all $N'(d)$ root points that have been added so far.

If $v_i = v_1$, go to Step 3. If all edges are deleted and $v_i \neq v_1$, repeat starting with a cluster of the updated tree.

Step 2: Deleting a cluster

Let $S = \{k | k = 1, ..., t, a_k > 0\}$, and $A = \{a_k | k \in S\}$. (A is a multiset.)

If S contains at least two elements, and the sum of the two largest elements in A is at least d, find n^+, the maximum number of disjoint pairs (j, l), $j \neq l$, of indices in S such that for each such pair $a_j + a_l \geq d$. If $n^+ = |S|/2$, delete all edges of the type $(v_i, v_{i(k)})$, $k = 1, ..., t$, from the current tree. Add n^+ to $N'(d)$. Repeat starting with a cluster of the updated tree.

If $n^+ < |S|/2$, find the index $s \in S$, corresponding to a largest element in A, such that n^+ is equal to the maximum number of disjoint pairs (j, l), $j \neq l$, of indices in $S - \{s\}$, with the property that for each such pair $a_j + a_l \geq d$. Delete all edges of the type $(v_i, v_{i(k)})$, $k = 1, ..., t$, from the current tree. Add n^+ to $N'(d)$. Increase the length of the edge $(v_i, p(v_i))$ from l_i to $l_i + a_s$, and repeat starting with a cluster of the updated tree. (Note that each of the new n^+ subtrees selected for the partition is a path containing v_i, which is also the root point of the path.)

If S contains one element, or if the sum of the two largest elements in A is smaller than d, delete all edges of the type $(v_i, v_{i(k)}), k = 1, ..., t$, from the current tree. Increase the length of the edge $(v_i, p(v_i))$ from l_i to $l_i + B_1$, where B_1 is the largest element in A, and repeat starting with a cluster of the updated tree.

Step 3: Termination at the root v_1

Let $S = \{k|k = 1, ..., t, a_k > 0\}$, and $A = \{a_k|k \in S\}$. (A is a multiset.) Find n^+, the maximum number of disjoint pairs (j, l), $j \neq l$, of indices in S such that for each such pair $a_j + a_l \geq d$. If $n^+ \geq 1$, delete from the cluster the $2n^+$ edges corresponding to this maximum solution. Add n^+ to $N'(d)$. If the remaining cluster consists of the root only, stop and return the current value of $N'(d)$. Otherwise, consider the root point x_1 which is the closest to v_1 amongst all $N'(d)$ root points that have been established. Stop and return the current value of $N'(d)$. (The last case corresponds to replacing the subtree previously associated with x_1 by its union with the current remaining subtree (cluster) rooted at v_1. We also declare v_1 to be the root point of this augmented subtree.)

End of Algorithm 4.

The validity of Algorithm 4 follows from arguments similar to those used in the proof of Lemma 3.1. We skip the details.

To evaluate the running time of Algorithm 4, consider the time needed to process a cluster with t leaves. We first sort the elements in A. Assume, without loss of generality that $0 < a_1 \leq ... \leq a_t$. Consider the smallest element in the (current) sorted list, say a_j. (Initially set $j = 1$.) Find the smallest index $k > j$ such that $a_j + a_k \geq d$. (If there is no such index k, set $X = a_j$, and remove a_j from the list.) Match the pair (j, k), and remove both a_j and a_k from the list. If the list is empty, the index s, defined above in Step 2, is the one corresponding to the current value of X. n^+ is the total number of pairs matched. If the list is nonempty repeat.

The total time to process a cluster with t leaves is clearly $O(t \log t)$. Therefore, the total running time of Algorithm 4 is $O(n \log n)$.

The next result characterizes the solution value.

Theorem 5.3 *Let d_p^{**} be the optimal solution value of the continuous max-min tree diameter partition problem. Then there exist a pair of nodes v_i, v_j, and an integer $q \leq p$, such that*

$$d_p^{**} = d(v_i, v_j)/q.$$

Proof: The proof is very similar to the proof of Theorem 5.1. We skip the details. ∎

With the representation of the optimal solution value in Theorem 5.3, and the $O(n \log n)$ algorithm to compute $N'(d)$, we can now directly apply the algorithmic framework in [10, 20], to obtain an $O(n \log^3 n)$ algorithm for the problem of maximizing the minimum diameter of a partition into p subtrees. Again, if $p = o(\log n)$, the complexity can be further improved to $O(pn \log^2 n)$ by using the search procedure in [15].

Theorem 5.4 *The continuous max-min tree diameter partitioning problem can be solved in $O(n \log^3 n)$ time.*

6 Remarks on non-crossing problems

We have shown above that the partitioning problems of a tree, minimizing (maximizing) the maximum (minimum) length of the components are both NP-hard, while the edge-partitioning versions are polynomially solvable. Our definitions of partitioning and edge-partitioning are independent of the particular planar embedding of the tree that is used. However, in many practical and real situations, we deal with physical networks, as in tree-like highway networks, where the embedding is already fixed. In physical networks not every partition is feasible, and we may have to impose further restrictions. For example, suppose that node v_1 has exactly four neighbors, $\{v_2, v_3, v_4, v_5\}$, each one of them connected to v_1 with an edge, and the embedding induces the cyclic ordering $(v_2, v_3, v_4, v_5, v_2)$. It might be the case that due to traffic considerations, no two components of a partition can "cross", e.g., the partition of the cluster into the two components induced by the two sets (paths) (v_1, v_2, v_3), (v_1, v_4, v_5) is feasible, while the crossing partition induced by (v_1, v_2, v_4), (v_1, v_3, v_5) is infeasible or illegitimate. We point out that tree partitioning problems requiring that the subtrees are non-crossing (with respect to the given embedding), can be solved in polynomial time. These problems can be formulated as edge-partitioning problems on cactus graphs (or cycle trees, as they are called in [15]).

An undirected graph is a *cactus* if each edge is contained in at most one cycle. Consider an embedded tree network $T = (V, E)$. For each node v_i, let $(v_{i(1)}, v_{i(2)}, ..., v_{i(t)}, v_{i(1)})$, $t = deg(v_i)$, be the cyclic ordering of all the neighbors of v_i, induced by the embedding. Define the following cactus graph $G = (V', E')$. Each node v_i of T is replaced by a cycle C_i, with $t = deg(v_i)$ nodes $\{u_{i,i(1)}, u_{i,i(2)}, ..., u_{i,i(t)}\}$, and $t = deg(v_i)$ edges $\{(u_{i,i(1)}, u_{i,i(2)}), ..., (u_{i,i(t)}, u_{i,i(1)})\}$. The length of each edge along the cycle is zero. Next, each edge (v_i, v_j) of T is replaced by an edge (of the same length) connecting node $u_{i,i(s)}$ of C_i with node $u_{j,j(q)}$ of C_j, where for some indices $s \leq deg(v_i)$ and $q \leq deg(v_j)$, $i(s) = j$ and $j(q) = i$.

A partition of the tree defines, for each node v_i, a partition of its neighbors into connected components. Such a partition is called *non-crossing* if it induces a partition of $(v_{i(1)}, v_{i(2)}, ..., v_{i(t)})$, $t = deg(v_i)$, into consecutive segments along the cycle. It is now easy to see that there is a one-to-one correspondence between non-crossing partitions of the tree T and edge-partitions of the cactus G. Finally, we note that the algorithms in Sections 3-4 can be extended to yield polynomial algorithms of the same complexity for edge-partitioning problems on cactus graphs. Combined with the above transformation, this will imply the polynomial solvability of bottleneck tree length, non-crossing partitioning problems on tree network spaces.

References

[1] A. Agnetis, P.B. Mirchandani, and A. Pacifici, "Partitioning of bi-weighted trees," *Naval Research Logistics* **49**, 2002, 143-158.

[2] R. Becker, Y. Perl and S. Schach, "A shifting algorithm for min-max tree partitioning," *J. ACM* **29**, (1982), 58-67.

[3] R. Becker and Y. Perl " Shifting algorithms for tree partitioning with general weighting functions," *J. of Algorithms* **4**, (1983), 101-120.

[4] R. Becker, B. Simeone and Y-I. Chiang, "How to make a polynomial movie from pseudopolynomially many frames: A continuous shifting algorithm for tree partitioning," in Fun with Algorithms: Proceedings of the international conference, E. Lodi, L. Pagli, and N. Santoro eds., Island of Elba, Italy, 1999, pp. 20-31.

[5] R. Becker, B. Simeone and Y-I. Chiang, "Continuous max-min tree partitioning-Part 1: From continuous to discrete, and Part 2: From discrete to continuous," Technical Report, Dipartimento di Statistica, Probabilita e Statistiche Applicate, Universita degli Studi di Roma "La Sapienza", no. 12, 1998.

[6] R. Becker, B. Simeone and Y-I. Chiang, "A shifting algorithm for continuous tree partitioning," *Theoretical Computer Science* **282**, 2002, 353-380.

[7] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, "Time bounds for selection," *J. Comput. System Science* **7**, (1972), 448-461.

[8] R. Chandrasekaran and A. Daughety, "Location on tree networks: p-center and N-dispersion problems," *Mathematics of Operations Research* **6**, (1981), 50-57.

[9] R. Chandrasekaran and A. Tamir, " An $O((n \log p)^2)$ algorithm for the continuous p-center problem on a tree," *SIAM J. Algebraic and Discrete Methods* **1**, (1980), 370-375.

[10] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms," *J. ACM* **34**, (1987), 200-208.

[11] G.N. Frederickson, "Optimal algorithms for tree partitioning," Proceedings 2-nd ACM-SIAM Symp. on Discrete Algorithms, San Francisco, 1991, 168-177.

[12] G.N. Frederickson, "Parametric search and locating supply centers in trees," Proceedings 2-nd Workshop on Algorithms and Data Structures, Ottawa, Canada, 1991, 299-319. Springer Verlag. LNCS 519.

[13] G.N. Frederickson, "Optimal parametric search algorithms in trees I: Tree partitioning," Technical Report, CS Purdue University, 1992.

[14] G.N. Frederickson, "Optimal parametric search algorithms in trees II: p-center problems and checkpointing," Technical Report, CS Purdue University, 1992.

[15] G.N. Frederickson and D.B. Johnson, "Finding k-th paths and p-centers by generating and searching good data structures," *J. of Algorithms* **4**, (1983), 61-80.

[16] M.R. Garey and D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-completeness, Freeman, San Francisco, 1979.

[17] M. Lucertini, Y. Perl and B. Simeone, "Most uniform path partitioning and its use in image processing," *Discrete Applied Math.* **42**, (1993), 227-256.

[18] N. Megiddo, "Combinatorial optimization with rational objective functions," *Mathematics of Operations Research* **4** , (1979), 414-424.

[19] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. ACM* **30**, (1983), 852-865.

[20] N. Megiddo and A. Tamir, "New results on the complexity of p-center problems," *SIAM J. Computing* **12**, (1983), 751-758.

[21] C.H. Papadimitriou, "Efficient search for rationals," *Information Processing Letters* **8**, (1979), 1-4.

[22] Y. Perl and S. Schach, "Max-min tree partitioning," *J. ACM* **28**, (1981), 5-15.

[23] Y. Perl and U. Vishkin, "Efficient implementation of a shifting algorithm," *Discrete Applied Math.* **12**, (1985), 71-80.

[24] S.P. Reiss, "Rational search," *Infromation Processing Letters* **8**, (1979), 89-90.