DIMACS Technical Report 2004-07
April 2004

Finding Central Sets of Tree Structures in Synchronous Distributed Systems

by

Jonathan W. Berry[1,2]
Dept. of Computer Science
Lafayette College
berryjw@cs.lafayette.edu

Daniel Hrozencik
Dept. of Math and Computer Science
Chicago State University
dhro@att.net

Shrisha Rao
Dept of Math and Computer Science
Mount Mercy College
srao@mtmercy.edu

Zhizhang Shen
Dept of Computer Science
Plymouth State University
zshen@plymouth.edu

[1]Permanent Member
[2]This work is the result of DIMACS Reconnect '03, Illinois Institute of Technology, attended by all four authors.

DIMACS is a collaborative project of Rutgers University, Princeton University, AT&T Labs–Research, Bell Labs, NEC Laboratories America and Telcordia Technologies, as well as affiliate members Avaya Labs, HP Labs, IBM Research and Microsoft Research. DIMACS was founded as an NSF Science and Technology Center.

Abstract

Finding the central sets, such as center and median sets, of a network topology is a fundamental step in the design and analysis of complex distributed systems. This paper presents distributed synchronous algorithms for finding central sets in general tree structures.

Our algorithms are distinguished from previous work in that they take only qualitative information, thus reducing the constants hidden in the asymptotic notation, and all vertices of the topology know the central sets upon their termination.

1 Introduction

The study of the center set and the median (centroid) set, collectively referred to as the central sets, as graph related concepts can be traced back to Jordan [8], where it was shown that any tree contains one or two adjacent center vertices, and one or two (possibly different) median vertices. Intuitively, the center set of a graph consists of vertices that have the minimum distance to the furthest vertex among all the vertices in the structure, while the median set of a graph consists of vertices that have the minimum aggregate distance to all the other vertices. Since the static part of communication delay depends on the number of hops between the two involved vertices of the interconnection network[1], when using a graph to represent the underlying topology of a computer network structure it is immediate that vertices contained in the center set of such a network lead to the minimum communication latency among all the nodes, while those contained in the median set lead to the minimum average latency.

Therefore, central vertices naturally play a critical role in unicast and multicast (broadcast) communication within the related network structure, respectively. For example, it is pointed out in [6] that the best possible strategy for sorting in a tree based network structure involves routing through a median vertex. The concept of the central nodes is also applied to model the Internet topology [5], where it is suggested that when a new node is added to the Internet, its distances to the central nodes should be minimized. Although the Internet does not possess a tree topology, the construction of the central vertices in a graph generally starts with the construction of the central vertices of a spanning tree. This computation will then be generalized into the that for finding central vertices of the original graph, as pointed out in[9]. However, even we limit our attention to trees, there are still important applications. For example, computational biologists are interested in structures as simple as rooted binary trees [11], and parallel center-finding on huge trees is of definite interest [18].

These examples and others demonstrate the importance of having simple and efficient distributed algorithms to find such central sets for various network structures. In this paper, we discuss distributed synchronous algorithms for finding central sets of a general tree structure, with the assumption that the only topological knowledge available to each node of the tree is its set of neighbors.

Korach *et al.* [9] present distributed algorithms for locating both the center and the

[1] This is even true in a wormhole routing model [12] when the network is congested.

median set in a general tree structures, following a different approach. In particular, their algorithms

- are naturally asynchronous, and therefore are not prepared to take advantage of a shared clock,
- terminate in a state in which each vertex knows whether or not it is a central vertex, but does not know the central vertices, and
- are quantitative in the sense that the program for each processor computes, transmits, and receives quantities.

In contrast, our center and median algorithms are naturally synchronous, do not terminate until every vertex has stored the central sets, and are based on respective sequential algorithms [2, 7], both of which are of a qualitative nature.

The rest of this paper proceeds as follows: Section 2 reviews some necessary graph theoretic concepts, and discusses the computational model for which our algorithms are designed. We then present our center algorithm and discuss its properties in Sections 3 and Section 4, respectively, and do the same for our centroid algorithm in Sections 5 and Section 6. Finally, Section 7 contains our concluding remarks.

2 Background and assumptions

The two main graph theoretic concepts related to the central sets we consider are the *center* and the *(branch weight) centroid.* In Slater's words [17], "Distinguishing the concepts of center and median is easy: One compares a worst-case distance criterion with a total distance criterion." Zelinka proved in [19] that for any tree, its (branch weight) centroid coincides with its median. It has also been proved that, for a tree, the notion of centroid coincides with numerous notions intended to capture the idea of a central set of a tree—such as the telephone center [10], the 1-processing center [3], security center [14, 17], accretion center [16], the $0-$ball branch weight [13], the $(n-1)$-centrum (where n is the number of vertices in the tree) [15], the latency center, the pairing center, etc.. This impressive collection of equivalency results indeed shows that the notion of the central vertices is a natural one.

We assume familiarity with basic graph theory, but include some discussion below to clarify our use of notation. Let E_v denote the set of edges incident to vertex v, and let $d(v,w)$ denote the length of the shortest path from vertex v to vertex w. The *eccentricity* of a vertex x, denoted $e(x)$, is given by:

$$e(x) \ = \ \max\{d(x,y) \mid x,y \in V(T)\}.$$

Throughout the paper, we will use T to denote a tree. The *center* of a tree, denoted $C(T)$, is the set of nodes having minimum eccentricity:

$$C(T) \ = \ \{x \in V(T) | \forall y \in V(T) \ e(x) \leq e(y)\}.$$

The *status function* of a vertex is given by:

$$s(x) \;=\; \sum_{v \in V(T)} d(x,v).$$

The *median*, is the set of all vertices of T whose status function gives the smallest value:

$$M(T) \;=\; \{x \in V(T) | \forall y \in V(T) \; s(x) \leq s(y)\}.$$

A *branch* $T_{v,e}$ is the subtree induced by the set of all vertices w sch that there exists a path that begins with edge e incident on v.

For $v \in (V(T) - C(T)$, let e_{vc} be the first edge on the (unique) path from v to the center, the *subtree induced by* v, denoted by T_v, is $\{v\} \cup T_{v,e}, e \in E_v - \{e_{vc}\}$. Intuitively, T_v denotes the subtree of T, rooted in v.

For any $x \in V(T)$, the *branch weight* of x, denoted $bw(x)$, is given by the number of vertices in the largest component of the subtree induced by $V(T) - \{x\}$. It is well known that the *branch weight centroid*, i.e., the set of vertices with minimum branch weight, is equivalent to the median.

The *radius* of a tree T, denoted as $r(T)$, refers to the minimum eccentricity among its vertices, and the *diameter* of T, denoted as $dia(T)$, refers to the maximum eccentricity of any vertex in T.

Finally, we assume a synchronous computational model, with all processors sharing a clock and a program. It is further assumed that a node representing a vertex $v \in V(T)$ has only the knowledge of E_v, the edges incident to v; it does not even know the number of vertices in the tree. Furthermore, communication links are duplex, supporting simultaneous communication in both directions, and all messages sent during clock tick t arrive at their destination by the beginning of the clock tick $t + 1$.

3 Finding the center of a tree

We address the center set related issues first.

3.1 The classical center algorithm

The following result [8] is one of the first regarding the central sets of graphs.

Theorem 3.1 *The center of a tree consists of either one or two adjacent vertices.*

We first establish the following result.

Lemma 3.1 *Let $T(V,E)$ be a tree, $|V| > 2$, v a leaf of $T, (v,w) \in E$. Then $e(v) > e(w)$.*

proof: In fact, let u be an eccentric node of w, then $e(v) = d(v,u) + d(u,w) = 1 + e(w) > e(w)$. □

Proof of the Theorem [2]: Let T be a tree, if T contains either one vertex, or two adjacent vertices, then, by definition, that vertex, or the pair of adjacent vertices, is (are) indeed the center of the respective tree. On the other hand, if T contains more than two vertices, then, by Lemma 3.1 and the fact that a center vertex has the minimum eccentricity among all vertices, no leaf belongs to the center. Since the simultaneous removal of all such leaves will cut the eccentricity of all the remaining vertices by 1, such removal will not affect the center set. Thus, we can remove the collection of leaves from the partially pruned tree in phases, until we have either one vertex, or two adjacent vertices, which must be the center. □

The essence of this constructive proof is that if we keep on removing leaves from the tree, until only one or two vertices are left, we will end up with the center of the tree.

Hence, the constructive proof of Theorem 3.1 also establishes the correctness of the following algorithm[2], which we call *Algorithm J.*

1. Set $T' = T$.
2. If $|V(T')| = 1$, or $|V(T')| = 2$, , then set $C(T) = V(T')$ and stop. Else go to Step 3.
3. Remove all the leaves along with its incident edges from T'. Call the resulting graph T'', reset $T' = T''$. Go back to Step 2.

Since Step 3 of the above algorithm needs global information, namely, the size of a graph, it is not a distributed algorithm. But, it certainly provides some inspiration for us to come up with an algorithm for a distributed structure.

3.2 Algorithm C

We assume that there is a 1-1 mapping between the computation nodes and the vertices of a tree, and we will use the term "vertex" to refer to both objects because of this 1-1 mapping in between. We also assume that the 1-1 mapping of vertices to computation nodes can be accomplished such that the assumption of unit message passing times stated in Section 2 is preserved.

We present a distributed algorithm, Algorithm C, that can be used to find the set of center vertices in a tree, $T(V,E)$. This algorithm mimics Algorithm J by pruning leaves from a partially pruned tree step by step until only one or two vertices left, which must constitute the center vertex set of the original tree.

The algorithm is expressed using the following notation:

- $(v,0)$: a *report message,* sent by a leaf v to its (only) adjacent vertex, in a partially pruned tree, when v together with T_v are being pruned;
- $(v,w), w \in V$: a *result message,* sent by a vertex v to notify its neighbors of a center vertex;

- $C(v)$: the set of center vertices known to v so far, when it is evaluated;
- $N(v)$: the set of neighbors of v in T;
- $M_r(v)$: the set of messages, both report and result messages, received by v so far when it is evaluated, initially set to be $\emptyset$;
- $M_r(v,t)$: the set of messages, both report and result messages, received by v at time t, initially set to be $\emptyset$;
- $S(v)$: a boolean variable indicating whether v has sent out a report message, and pruned from the partially pruned tree, initially set to be $false$;
- t: time step, initially set to be 0;
- $U(v)$: the set of the neighbors of v that have sent v a report message, initially set to be $\emptyset$;

We now specify Algorithm C, which is uniform for every vertex $v \in V$. With $N(v)$, as defined in T, as its input, this algorithm will notify all such v about $C(T)$, the center vertex set of T, at the end of its execution.

1. $C(v) \leftarrow \emptyset$; $M_r(v) \leftarrow \emptyset$; $U(v) \leftarrow \emptyset$; $S(v) \leftarrow false$; and $t \leftarrow 1$;
2. $M_r(v,t) \leftarrow \{$msgs received by vertex v at time $t\}$;
3. $M_r(v) \leftarrow M_r(v) \cup M_r(v,t)$;
4. if $((|M_r(v)| \geq |N(v)|) \wedge (|M_r(v,t)| = 0))$ $\quad STOP.$
5. for $(u,w) \in M_r(v,t)$
   (a) if (w = 0) $\quad U(v) \leftarrow U(v) \cup \{u\}$;
6. (a)
   - If $(|U(v)| = |N(v)| - 1) \wedge \neg S(v)$:
     i. send $(v, 0)$ to the single $x \in N(v) - U(v)$;
     ii. $S(v) \leftarrow true$;
   - Else If $|M_r(v)| > |U(v)|$:
     i. $C(v) \leftarrow C(v) \cup \{w\}$: $\quad (u,w) \in M_r(v,t)$;
     ii. send (v,w) to every $x \in N_v - \{u\}$;
   - Else If $|U(v)| = |N(v)|$:
     i. $C(v) \leftarrow C(v) \cup \{v\}$;
     ii. send (v,v) to every $x \in N_v$;

   (b) Continue
7. $t \leftarrow t + 1$;
8. return to Step 2;

4 Properties of Algorithm C

Before proceeding with the correctness proof of Algorithm C, we introduce some additional notation. First, we define a predicate $\mathcal{R}(\tau)$ that will be true iff there exists a vertex that sends a *report* message in time period τ. We define $\mathcal{B}(\tau)$ similarly for *result* messages.

Let $\mathcal{T}_r = \{\tau | \mathcal{R}(\tau) \text{ is true}\}$, $\mathcal{T}_b = \{\tau | \mathcal{B}(\tau) \text{ is true}\}$, and $\tau_r = \max\{\tau | \tau \in \mathcal{T}_r\}$, we partition Algorithm C into three phases as follows: Time periods $\mathcal{T}_r - \{\tau_r\}$ comprise the *reporting phase*, τ_r is the *discovery phase*, and $\mathcal{T}_b$ is the *broadcast phase*.

The discovery phase follows the reporting phase by definition, and Lemma 4.5 will show that the broadcast phase is disjoint from the discovery phase.

4.1 The reporting phase

Recall that T' is the partially pruned tree as mentioned in Algorithm J, Algorithm C is based on the following fundamental design proposition that relates the reporting phase to executions of Step 3 of Algorithm J (the pruning step), since sending out such a report message is coupled with setting $S(v)$ to *true*, indicating that it is pruned from T' by definition.

Proposition 4.1 *The act of sending a* report *message from a vertex in Algorithm C's reporting phase pruns the corresponding vertex from T'.*

This proposition leads directly to the following lemma.

Lemma 4.1 *During the reporting phase of Algorithm C, a vertex v has a neighbor w that has not sent a* report *message iff w is a neighbor of v in T'.*

Proof: Suppose that v has a neighbor w in T', yet all neighbors of v have sent *report* messages. By Proposition 4.1, v should have no neighbors left in T', which is a contradiction. Conversely, suppose that when applying Algorithm C there is a neighbor w of v that has not sent a *report* message. By Proposition 4.1, w has not been pruned from T', and thus remains a neighbor of v in T'. □

Thus, a *report* message sent from w to v in Algorithm C's reporting phase provides vertex v with the following information:

- The vertex corresponding to w in Algorithm J has been pruned, and therefore the center is not in T_w.
- If w was the second-to-last neighbor of v to be pruned, i.e., if $|U(v)| = |N(v)| - 1$, then the center is in the direction of the sole remaining neighbor. A *report* must be sent to the latter.

At each step of Algorithm J except the last, all leaf vertices are pruned. Similarly, at each iteration of the loop of Lines 2-8 of Algorithm C, all vertices representing leaves in T' send *report* messages toward the center of the tree.

The reporting phase of Algorithm C excludes τ_r because the natural correspondence between Algorithm C and Algorithm J is guaranteed only during time periods before τ_r. By definition, τ_r is the last period in which a report message is sent. Regardless of whether the input tree has one or two center vertices, therefore, all center vertices will have received reports from all of their neighbors by the beginning of $\tau_r + 1$. In the case of two center vertices, they are adjacent by Theorem 3.1, and the report messages they send each other in τ_r do not correspond to pruning steps since Algorithm J terminates without pruning center vertices. Formalizing this idea, we have the following.

Lemma 4.2 *By the end of τ_r, all vertices v, $v \notin C(T)$, have sent exactly one* report *message.*

Proof: Every non-center vertex in T is pruned by Algorithm J, and by Lemma 4.1, every such sertex has to send out a *report* message in Algorithm C in some time period τ, where $\tau < \tau_r$. If $|C(T)| = 2$, all non-center vertices have been pruned by time $\tau_r - 1$, as this is the case in which Algorithm J terminates and Algorithm C executes one more round of *report* messages (exchanged between center vertices). If $|C(T)| = 1$, then the last pruning step of Algorithm J corresponds to the discovery phase of Algorithm C. In this case, Lemma 4.1 continues to hold during τ_r, and any remaining non-center vertices will send report messages. The $S(v)$ flag in Line 6a of Algorithm C ensures that no vertex reports more than once, and the result follows. □

4.2 The discovery phase

In Algorithm J, only the center vertices remain unpruned in T' after the final pruning step. The correspondence we have established between Algorithm J and the reporting phase of Algorithm C can be used to show that Algorithm C finds the center vertices correctly.

Lemma 4.3 *At the beginning of the discovery phase of Algorithm C (τ_r), every vertex $v \in T, v \notin C(T)$, has received exactly $N(v) - 1$* report *messages.*

Proof: Suppose that there exists a vertex $v \notin C(T)$ that has received fewer than $N(v) - 1$ *report* messages at the beginning of τ_r. Then $U(v) < N(v) - 1$, and v, which must report by Lemma 4.2, must wait to report in some subsequent time period. This is a contradiction since τ_r is defined to be the last period in which a report is sent. On the other hand, suppose that v has received more than $N(v) - 1$ *report* messages. By Lemma 4.1, all of v's neighbors have been pruned during the reporting phase. In other words, v remains unpruned in Algorithm J and is a center vertex, contradicting the assumption of this result. □

Lemma 4.4 *A vertex v is a center vertex iff v receives a* report *message in $\tau_r + 1$.*

Proof: Suppose that $v \in C(T)$, but v does not receive a *report* message in $\tau_r + 1$, then the final *report* message sent to v must have been received at $\tau_1 \leq \tau_r$ and sent at $\tau_2 < \tau_r$. Regardless of $|C(T)|$, all center vertices receive their final reports in the same time period, so no further report will be sent in τ_r, which is a contradiction.

Suppose a vertex v that receives a report message in $\tau_r + 1$, but is not a center vertex. It must be the case that $|U(v)| < |N(v)|$, or the third test of Line 6a of Algorithm C would have succeeded in identifying v as a center. Thus v has a neighbor w that has not sent v a report message. By Lemma 4.3, $|U(v)| = |N(v)| - 1$, so $|U(v)|$ was less than $|N(v)| - 1$ in period τ_r. In other words, v has not yet sent a report message ($S(v)$ is false). However, by Lemma 4.2, since v is assumed to be a non-center vertex, it should have sent out a *report* message by the end of τ_r, which is a contradiction. Hence, v must be a center vertex. □

4.3 The broadcast phase

An immediate consequence of Lemma 4.4 is the following:

Lemma 4.5

$$\mathcal{T}_r \cap \mathcal{T}_b = \emptyset$$

Proof: By Lemma 4.4, the first result message cannot be sent until $\tau_r + 1$, one time period after the last report message was sent. □

The broadcast phase relays one and possibly two *result* messages originating at center vertices to all other vertices in the tree.

Lemma 4.6 *After a vertex v receives a* result *message in time period $\tau > \tau_r$, $|M_r(v)| > |U(v)|$. Furthermore, if v is to receive another* result *message, it will arrive in time period $\tau + 1$.*

Proof: If $v \notin C(T)$, Lemma 4.3 requires that $|M_r(v)| = |U(v)| = |N(v)| - 1$ just before τ. Receiving a result message ensures that $|M_r(v)| > |U(v)|$. On the other hand, if $v \in C(T)$, then $|U(v)| = |N(v)|$ held at $\tau_r + 1$. Once again, the new result message ensures that the required condition will hold.

Since there are at most two center vertices, a center vertex can receive a result message from at most one other center vertex. Therefore, if v receives two result messages, it is not a center vertex. Since the *result* messages originate during the same time period ($\tau_r + 1$) from adjacent center vertices and propagate along the unique path to v at the same rate, they will arrive in consecutive time periods. □

To conclude our discussion of the broadcast phase, we show that it covers all vertices.

Lemma 4.7 *Every vertex v, $v \notin C(T)$, receives a result message during the broadcast phase.*

Proof: Refer to Figure 4.3. Suppose there is a vertex $v \notin C(T)$ that does not receive a result message. By Lemma 4.3, v has received *report* messages from all $u \in N(V)$ except one. Call this neighbor w. By Lemma 4.2, v has sent a *report* message to w, and all other vertices on the unique $v \rightsquigarrow c$ path have sent their *report* messages. Since v does not receive a result message, there must be a vertex on this path that does not send a result message to its neighbor on the unique path to v. Let x be the first such vertex, by Lemma 4.6, $|M_r(x)| > |U(x)|$, so x must send result messages to all neighbors not on the $x \rightsquigarrow c$ path. In other words, it must send one to its neighbor in the $x \rightsquigarrow c$ path, a contradiction. □

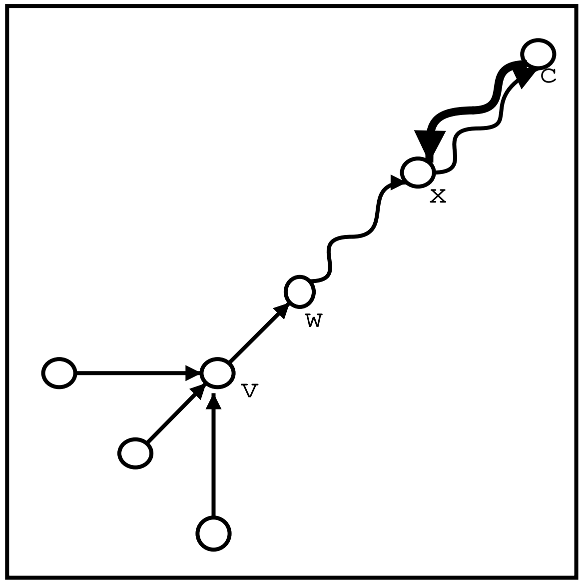

Figure 1: Lemma 4.7. Thin edges represent *report* messages, while thick edges represent result messages.

4.4 Correctness

The lemmas of this section yield the following theorem.

Theorem 4.1 *At the end of the execution of Algorithm C on an input tree T, the center set $C(T)$ has been identified and is known to all vertices of T.*

4.5 Termination and time complexity

We show that, when applied to a tree T, Algorithm C terminates after $2r(T)$ steps if T contains one center; or $2r(T) + 1$ steps if T contains two steps.

Lemma 4.8 *Let $T(V, E)$ be a tree. When applying Algorithm C to T, for all $v \in V, w \in N(v), v$ receives at least one message from w.*

Proof: The proof is an immediate consequence of Lemma 4.3, Lemma 4.7, and the proof of Lemma 4.4. □

Theorem 4.2 *Let $T(V, E)$ be a tree. When applying Algorithm C to T, for all $v \in V$, there exists a τ, at which the termination condition in Step 4, namely, $(|M_r(v)| \geq |N(v)|) \wedge (|M_r(v, t)| = 0)$ holds for v.*

Proof: By Lemma 4.8, for all $v \in V$, there exists a τ_v, at which, $M_r(v) \geq |N(v)|$. Take $\tau_1 = \max_{v \in V}\{\tau_v\}$.

By definition, no report messages are sent after τ_r, and by Lemma 4.6, a result message received by a vertex v at time τ can be followed by at most one more result message, to be received at $\tau + 1$. By Lemma 4.7, all non-center vertices receive the center broadcast(s), so such a τ exists for each $v \notin C(T)$. Center vertices can receive at most one result message, then will receive no further message. □

Theorem 4.3 *Let T be a tree. When applied to T, Algorithm C terminates in $2r(T)$ steps if T contains one center, or in $2r(T)+1$ steps if T contains two centers.*

Proof: Starting at $t = 1$, Algorithm C takes one time step for a leaf to send up a $(v, 0)$ message to the next node in the path to the center. By definition of being a center vertex, a center vertex collects all the report messages at $r(T) + 1$, and, by the definition of the algorithm, it starts to send out result message during the same time step. This message will be received by all of the vertices, particularly, the furthest one, at $2r(T)+1$, a total of $2r(T)$ steps.

In the case of two centers, both centers will have complete knowledge at time step $r(T)+2$, and the process will complete at time step $2r(T) + 2$, a total of $2r(T) + 1$ steps. □

5 Finding the centroid of a tree

We first present a well-known sequential algorithm that finds the centroid of a tree, then give our distributed version.

5.1 Goldman's Algorithm

Given a tree, $T(V, E)$, initially, we set $w(v)$, the weight associated with $v \in V$, to 1. In general, let $w(S) = \sum_{v \in S} w(v)$, where $S \subseteq V$.

Alan Goldman's algorithm [7, pp.215] is essentially the following:

1. If T contains a single node, stop; that node is the median;

2. Search for a leaf t and an edge connecting t and its adjacent node s. If $w(t) \geq \frac{|w(T)|}{2}$, stop; t is a median;

3. Modify T by deleting t together with the edge (t, s), and increment $w(s)$ by $w(t)$;

4. Go back to Step 1.

Given a tree, $T(V, E)$. Let $(s_1, s_2) \in E$ be an edge connecting two vertices $s_1, s_2 \in V$. Then $T - \{(s_1, s_2)\}$ consists of exactly two components. Let S_1, S_2 be the vertex set of those two components such that $s_1 \in S_1, s_2 \in S_2$, the two connected components will be named T_{S_1} and T_{S_2}. The correctness of Goldman's algorithm, as stated in Theorem 5.1, is based on the following two lemmas [7, Lemma 1 and Lemma 2], by setting $S_1 = V - \{t\}$, and $S_2 = \{t\}$, where t is the leaf mentioned in Step 2 of the above algorithm.

Lemma 5.1 *If $w(S_2) \geq w(S_1)$, then S_2 contains at least one median.*

Lemma 5.2 *If $w(S_2) \leq w(S_1)$, then the median of T is also a median of a subnetwork T_{S_1}, with node-weights w' as in S_1 except that $w'(s) = w(s) + w(S_2)$.*

Theorem 5.1 *Let $T(V, E)$ be a tree, a vertex $v \in V$ is a median of T iff, during certain time step τ, it is found out that $w(v) \geq \frac{|w(T)|}{2}$ when Goldman's algorithm is applied to T.*

Incidentally, the following result [8] is fundamental to the notion of centroid (median vertex).

Theorem 5.2 *The (branch weight) centroid of a tree consists of either one or two adjacent vertices.*

5.2 Algorithm M

We now present a distributed algorithm, Algorithm M, that can be used to find the set of median vertices in a tree. This algorithm mimics Goldman's sequential algorithm by continuously pruning the leaves from the tree, all at once, until the weight of a node becomes "critical", when it is confirmed to be a median vertex.

Below are some of the notations used in the algorithm and their explanation:

- $d(v)$: the degree of v in the partially pruned tree T.
- (v, m): sent by a leaf v of its weight, m, to its (only) adjacent vertex;
- $(v, \{m\})$: initialized by m, a median vertex, and broadcast to all the vertices via v;
- $M(v)$: the set of median vertices known to v so far, initially set to be $\emptyset$;
- $N(v)$: the set of neighbors of v in T;
- $S(v)$: a boolean variable indicating whether v should further forward the weight information collected in v;
- t: time step, initially set to be 0;
- $U(v)$: the subset of v's neighbors that have sent v a (v, m) type message, initially set to be $\emptyset$;
- w: the weight of T, which is always equal to $|V|$;
- $w(v)$: the current weight of v, initially set to 1;

We note that, with the aforementioned initial values set as is, the sum of $w(v), v \in V(T)$ equals to $|V|$ throughout the execution of the algorithm. This quantity can be obtained by applying a distributed depth-first search algorithm, e.g., the one reported in [1], which takes $O(V(T))$ to complete. Thus, we assume that each node has this information when the following Algorithm M starts.

We now specify Algorithm M, which is uniform for every vertex, $v \in V$. With $N(v)$ as its input, this algorithm will notify v about $M(T)$, the median vertex set of T, upon its completion.

1. If $w \leq 2$, declare v as a median vertex, then stop.
2. $d(v) \leftarrow |N(v)|$; $M(v) \leftarrow \emptyset$; $S(v) \leftarrow false$; $U(v) \leftarrow \emptyset$; $w(v) \leftarrow 1$;
3. $t \leftarrow 1$;
4. Repeat the following until nothing is applicable:
   - (if $d(v) = 0$) Immediately going to Step 5
   - (else if $d(v) = 1$)
     (a) if($S(v) = false$) let $x \in N(v) - U(v)$.
       i. $d(v) \leftarrow 0$;
       ii. send $(v, w(v))$ to x;
       iii. $S(v) \leftarrow true$.
   - (else if $d(v) > 1$)
     (a) for each message (u, m) received
       i. $d(v) \leftarrow d(v) - 1$;
       ii. $U(v) \leftarrow U(v) \cup \{u\}$;
       iii. $w(v) \leftarrow w(v) + m$;
     (b) if $w(v) \geq \lceil \frac{w}{2} \rceil$
       i. $M(v) \leftarrow \{v\}$;
       ii. send $(v, \{v\})$ to all $x \in N(v)$;
       iii. $S(v) \leftarrow true$;
5. for each message $(u, \{m\})$ received
   (a) $M(v) \leftarrow M(v) \cup \{m\}$;
   (b) send $(v, \{m\})$ to every $x \in N(v) - \{u\}$;
6. $t \leftarrow t + 1$;

5.3 Explanation of the algorithm

If the size of the tree is no more than 2, T either contains one vertex or two adjacent vertices. The task then becomes trivial in either case.

In general, after initializing in Steps 2, and 3, the algorithm checks to see, for a specific vertex, v, which of the following three cases applies, *for the first time,* at time $\tau \geq 1$.

- If $d(v) = 0$, we need to discuss two cases, depending on *when* $d(v)$ is set to 0:

  – $d(v) \leftarrow 0$ at $t_0 \in (1, \tau)$, when the value of $d(v)$ was 1. By definition, $S(v) = false$ at the beginning of t_0, thus, v has heard from all but one of its vertices, becoming a leaf in the partially pruned tree, during $t_0 - 1$. Also, it is not a median vertex, otherwise, $S(v)$ would be set to ture during $t_0 - 1$. Hence, by Lemma 5.2, v should be deleted, as shown in Step 3 of Goldman's algorithm. This is done during t_0 by setting $d(v)$ to 0.

    Eventually, a broadcast message containing median information is sent back to v, and received at τ, which is further sent to all vertices in $T_v - \{v\}$, as described by the action taken at τ, in Step 5.

  – It is also possible that $d(v)$ is decremented to 0 in case of $d(v) > 1$ at $\tau - 1$, when it hears from each and every of its neighbors[2]. As a result, its $w(v)$ value has to be the sum of the weights of all its neighbors, i.e., $w(v) = w > \lceil \frac{w}{2} \rceil$. Thus, v has to be the only median vertex of the tree[3]. In this case, no other vertex sends v a $(v, \{m\})$ type message, particularly, prior to τ, and no action needs to be taken.

- If $d(v) = 1$ at τ, then $d(v) \leftarrow 1$ at $\tau - 1$, since the algorithm always checks $d(v)$ in the next round. There could also be two cases:

  – Satisfaction of this condition, including the special case that v is a leaf in T, indicates that v has collected all the information in $T_v - \{v\}$ at $\tau - 1$. If the condition of $w(v) \geq \lceil \frac{w}{2} \rceil$ was not satisfied at $\tau - 1$, i.e., v is not a median, then $S(v)$ remains to be false at τ. Thus, by Lemma 5.2, such a v becomes a leaf of the partially pruned tree, and should be pruned itself, with its weight being transferred to its only neighbor. Obviously, such transfer should be done only once, thus, $S(v)$ is flipped after the transfer.

    For the special case of v being a leaf, since for any non-trivial tree, $|V| \geq 3$, no leaf could be a median vertex. Thus such initial weight transfer will not cause any problem.

    We also notice that it is possible, even before a vertex sends over its weight, it could receive a message containing median vertex information. The "bloomer"

[2] The star graph provides an example here. Notice that this case can't be proved with Goldman's lemmas, but it certainly can be simulated with Goldman's algorithm.

[3] The very reason that all of its neighbors send over their weight information indicates that none of them is a median, as its $S(v)$ value is still false.

graph provides such an example. That is why Step 5 is also necessary for this case. Particularly, after sending over the weight information, it still could relay any such notification message to the other neighbors.

– On the other hand, if the condition of $w(v) \geq \lceil \frac{w}{2} \rceil$ was satisfied at $\tau - 1$, then it finds itself to be a median due to Lemma 5.1 at $\tau - 1$, and notifies all its neighbors. Since in this case, the reduction process as described in Lemma 5.2 is no longer necessary, we set $S(v)$ to be true, which will suspend the reduction process in the $d(v) = 1$ case in the next round at τ.

  But, when there exist two median vertices, according to Theorem 5.2, this just found median vertex is adjacent to another median vertex, which could also send over a notification at τ, thus the need for sending out a $(v, \{m\})$ type message for this subcase, as well.

- If $d(v) > 1$ at τ, we are in a pruning and data collecting stage. For each weight message sent over by u, a neighbor of v, based on an inductive argument, we have that T_u is being removed, thus $d(v)$ has to be decremented by 1. Also, since we need to know the only neighbor from which we have not heard thus report to it in the aforementioned case of $d(v) = 1$, we have to add u to $U(v)$.

  Once we have collected all such information, we check to see whether the condition of $w(v) \geq \lceil \frac{w}{2} \rceil$ is satisfied. If that is the case, by Lemma 5.1, v is a median vertex, and should be so declared. We also start to notify the others. Finally, we have to stop the reduction process in this case, thus we set $S(v)$ to be true.

  It is also possible that v receives a $(v, \{m\})$ type message in this case.

Since all three cases need a step for checking, and possibly, sending out, a message containing information about the median vertex, we place such a step in the Step 5 of the algorithm.

6 Verification and analysis of Algorithm M

To establish the correctness of Algorithm M, we firstly consider some trivial cases. Let $T(V, E)$ be a tree, and let $|V| \leq 2$, then T contains either one vertex v, or two adjacent vertices u, v such that $(u, v) \in E$. By the very definition of a vertex being a median vertex, either v is the median vertex of $T(\{v\}, \emptyset)$ or both u and v are median vertices of $T(\{u, v\}, \{(u, v)\})$. We notice that Goldman's algorithm takes one step to find out the median in the first case, and non-deterministically finds out, in two steps, that u, v are both median vertices in the second case.

Now, we argue the general correctness of Algorithm M, when it is applied to $T(V, E), |V| \geq 3$, via its connection to Goldman's algorithm.

Theorem 6.1 *Let $T(V, E)$ be a tree, $|V| \geq 3$, $v \in V$, and let w denote $|V|$. Then, $\exists \tau_1 \geq 2$, Algorithm M discovers the fact that $w(v) \geq \lceil \frac{w}{2} \rceil$ iff $\exists \tau_2 \geq \tau_1$, Goldman's algorithm discovers $w(v) \geq \lceil \frac{w}{2} \rceil$.*

Proof: Assume that at $\tau_1 = 2$, Algorithm M discovers in the case of $d(v) > 1$ of Step 4 that $w(v) \geq \lceil \frac{w}{2} \rceil$, which is certainly consistent with the assumption that $|V| \geq 3$. Thus, $w(v) \geq 2$. Hence, v must have received at least one message in the form of $(u, w(u))$ sent over at $t = 1$. By the definition of the algorithm, only vertices with their degree being 1 at $t = 1$ could send out such messages. Since the degree of a vertex is tested first before sending out such a message, these vertices, arbitrarily denoted as $u_1, u_2, \ldots, u_m, m \geq 1$, must be leaves of the original T. Now, we repeatedly apply Goldman's algorithm, taking $u_i, i \in [1, m]$ as the leaf as mentioned in Step 2, it takes at least two steps to exhaust all these leaves. When it does complete, we must have that $w(v) \geq \lceil \frac{w}{2} \rceil$.

On the other hand, Assume that at $\tau_2 = 1$ or $\tau_2 = 2$, Goldman's algorithm discovers the median vertices, then Algorithm M takes one step, at $t = 0$, to directly conclude for both cases, although in each case, the condition of $w(v) \geq \lceil \frac{w}{2} \rceil$ does hold.

In general, assume that at $\tau_1 > 2$, in case $(d(v) > 1)$ of Step 4, Algorithm M discovers that $w(v) \geq \lceil \frac{w}{2} \rceil$ holds true for v for the first time. Then, similar to the argument we made for the previous case, v must have received at least one message in the form of $(u, w(u)$ from a set U_1 of its neighbors in a previous step during τ_1; and might have received more messages in the same form from a set U_2 of its neighbors during a previous time unit $t < \tau_1$; and it has not heard from the rest of its neighbors $N(v) - (U_1 \cup U_2)$. By the definition of the algorithm, the weights of all the vertices in $V(T_u), u \in U_1 \cup U_2$, have been transferred to v prior and during τ_1. Moreover, the assumption for this case indicates that the sum of their weights is at least as large as half of the total weight of the original tree.

To proceed, we order all the trees, $T_u, u \in U_1 \cup U_2$, in an arbitrarily but fixed manner; and, for each such tree, we also arrange all its vertices according to a postorder tree walk [4, pp.245]. We then apply Goldman's algorithm to T, taking the vertices of $T_u, u \in U_1 \cup U_2$, arranged in the aforementioned order, as the leaf mentioned in Step: 2. It will take $\tau_2 \leq \sum_{u \in U_1 \cup U_2} |V(T_u)|$ time steps for Goldman's algorithm to complete, at which it will also find out that $w(v) \geq \lceil \frac{w}{2} \rceil$, as the weights of all those vertices will transfer to v, and by the assumption of this case, their sum is at least as large as half of the total. Finally, since it takes Goldman's algorithm one time step to transmit the weight of a leaf vertex to its parent, but all the leaf vertices in the case of Algorithm M, it will take Goldman's algorithm at least the time it takes Algorithm M to transmit the weights from $u \in U_1 \cup U_2$. Hence, $\tau_1 \leq \tau_2$.

We now assume that Goldman's algorithm discovers, for some $v \in V$, that $w(v) \geq \lceil \frac{w}{2} \rceil$ at τ_2. Let $L(\tau, v)$ be $\{u | u$ is a leaf removed by Goldman's algorithm in Step 3 during time step τ. when discovering $w(v) \geq \lceil \frac{w}{2} \rceil \}$; and let $L_v(i, \tau)$ be $\{u | u \in L(\tau, v)$ and $d(u, v) = i\}$. Then, we have that $U_{i \leq \tau} L_v(i, \tau)$ is a partition of $L(\tau, v)$.

We now apply Algorithm M to the same tree, and during time step $i, i \in [1, \tau)$, by definition of the algorithm, Algorithm M remove all the vertices in $L_v(\tau - i, v)$ and transfer their weights to their parent in T. When all the vertices in $L(\tau, v)$ are removed during step τ_1, all their weights will be transferred to v by definition of $L(\tau, v)$. By the assumption for this case, we will have that $w(v) \geq \lceil \frac{w}{2} \rceil$. It is also easy to see, based on the sequential nature of weight transmission of Goldman's algorithm, that $\tau_1 \leq \tau_2$, and they are equal iff for all $i \in [1, \tau_2], |L_v(i, \tau_2)| = 1$. □

Finally, we analyze the time complexity of Algorithm M.

Theorem 6.2 *Let $T(V, E)$ be a tree. When applied to T, Algorithm M completes in $\Theta(dia(T))$, where $dia(T)$ is the diameter of T.*

Proof: Let the longest path in T be p, and u and v be its end points. Thus, both u and v are leaves. At the end of $t = 1$ for both of them, their d_v becomes 0. It is likely that $w(v)$ for some vertex, v, reaches the thresh hood at $t = 1$, for example, in the case of the "bloomer" graph example. It will then take $\Theta(dia(T))$ time to send the centroid set information all the way to the other end of p.

In general, assume that at $t = t_0$ the median vertex information is complete, then it takes t_0 time to obtain this information for a median vertex, and $l(p) - t_0$ time to distribute throughout the tree. Thus, it takes $\max\{t_0, l(p) - t_0\}$ time in total, considering the parallel nature of the distributed system. The lowest upper bound of this latest time complexity is indeed $l(p) = \Theta(dia(T))$. □

Since the preprocessing step takes $O(|V|)$, and $dia(T) = O(|V|)$, we have the following final result.

Corollary 6.1 *Let $T(V, E)$ be a tree. It takes $O(|V(T)|)$ to find out $M(T)$.*

7 Conclusions

The primary contribution of this paper is to describe modern, synchronous adaptations of classical, qualitative algorithms for finding central sets of trees. Both our computational model and the topology we address are simple and static, so avenues for future research include allowing for irregularities along both dimensions.

We are also working on asynchronous versions of the algorithms we presented in this paper.

References

[1] B. Awerbuch. A new distributed depth-first-search algorithm. *Information Processing Letters*, 20:147–150, 1985.

[2] Fred Buckley. Facility location problems. *Journal of College Mathematics*, 18:24–32, 1987.

[3] D. Cikanek and Peter J. Slater. The $k-$processing center of a graph. *Congressus Numerantium*, 60:199–210, 1987.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, second edition, 2001.

[5] A. Fabrikant, E. Koutsoupias, and C. H. Papadimitriou, C. Heuristically optimized trade-offs: A new paradigm for power laws in the internet. *Proc. ICALP*, pages 110–122, 2002.

[6] O. Gerstel and S. Zaks. A new characterization of tree medians with application to distributed sorting. *Networks*, 24:23–29, 1994.

[7] Alan J. Goldman. Optimal center location in simple networks. *Transportation Science*, 5:212–221, 1971.

[8] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew Math*, 70:185–190, 1869.

[9] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Transactions on Programming Languages and Systems*, 6(3):380–401, 1984.

[10] S. Mitchell. Another characterization of the centroid of a tree. *Discrete Mathematics*, 24:277–280, 1978.

[11] B.M.E. Moret, S. Wyman, D.A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symp. on Biocomputing (PSB 2001), Hawaii*. World Scientific Pub., 2001.

[12] L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb. 1993.

[13] K. Brooks Reid. Centroids to centers in trees. *Networks*, 21:11–17, 1991.

[14] Peter J. Slater. Maximin facility location. *J. Research of the National Bureau of Standards*, B, 79B:107–115, 1975.

[15] Peter J. Slater. Centers to centroids in graphs. *Journal of Graph Theory*, 2:209–222, 1978.

[16] Peter J. Slater. Accretion centers: a generalization of branch weight centroids. *Discrete Appl. Math.*, 3:187–192, 1981.

[17] Peter J. Slater. A survey of sequences of central subgraphs. *Networks*, 34:244–249, 1999.

[18] T. Warnow. personal communication.

[19] B. Zelinka. Medians and peripherians of trees. *Arch. Math. (Brno)*, 4:87–95, 1968.