Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 16, no. 2, pp. 199–224 (2012)

Centdian Computation in Cactus Graphs

Boaz Ben-Moshe[1] *Amit Dvir*[2] *Michael Segal*[3] *Arie Tamir*[4]

[1]Department of Computer Science
Ariel University Center of Samaria, Israel
[2]Laboratory of Cryptography and System Security (CrySyS)
Budapest University of Technology and Economics, Hungary
[3]Department of Communication Systems Engineering
Ben-Gurion University of the Negev, Israel
[4]School of Mathematical Sciences
Tel-Aviv University, Israel

Abstract

This paper focuses on the centdian problem in a cactus network where a *cactus* network is a connected undirected graph, and any two simple cycles in the graph have at most one node in common. The cactus network has important applications for wireless sensor networks when a tree topology might not be applicable and for extensions to the ring architecture. The centdian criterion represents a convex combination of two QoS requirements: transport and delay. To the best of our knowledge, no efficient algorithm has yet been developed for constructing a centdian node in a cactus graph, either sequential or distributed. We first investigate the properties of the centdian node in a cycle graph, and then explore the behavior of the centdian node in a cactus graph. Finally, we present new efficient sequential and distributed algorithms for finding all centdian nodes in a cycle graph and a cactus graph.

Submitted:	Reviewed:	Revised:	Reviewed:
July 2010	February 2011	June 2011	December 2011
Revised:	Accepted:	Final:	Published:
December 2011	December 2011	January 2012	January 2012
Article type:		Communicated by:	
Regular paper		G. J. Woeginger	

This work was done while the second author was graduate student in the Department of Communication System, Ben-Gurion University of the Negev, Israel. Amit has also been supported by the Marie Curie Mobility Grant, OTKA-HUMAN-MB08-B 81654. Work of Michael Segal has been partially Supported by EPSRC grant EP/H018816/1, US Air Force European Office of Aerospace Research and Development, grant FA8655-09-1-3016, Deutsche Telecom, France Telecom, European project FLAVIA and Israeli Ministry of Industry, Trade and Labor (consortium CORNET).

E-mail addresses: benmo@ariel.ac.il (Boaz Ben-Moshe) azdvir@gmail.com (Amit Dvir) segal@cse.bgu.ac.il (Michael Segal) atamir@post.tau.ac.il (Arie Tamir)

1 Introduction

A wireless sensor network contains a number of sensor nodes, limited in power and memory, distributed across an area using wireless communication links to deliver information between nodes. In some sensor network applications the nodes are barely changed [45]. In recent years, the wireless sensor network has attracted attention [38,40,47] since this type of network can be used in a variety of applications, such as health, military, and emergency response.

A wireless ring is a structure for applications running on wireless sensor networks that provide quality of service [13, 30–32, 34]. Wireless Token Ring Protocol (WTRP) is a medium-access-control (MAC) protocol with advantages such as robustness against single node failure, and support for flexible topologies, in which nodes may be partially connected [13–15, 30, 31]. Lee et al. [30, 31] presented a WTRP for ad hoc networks and an intelligent transportation system. Xianpu et al. [44] showed a dynamic token protocol for Mobile Ad Hoc Network (MANET) where all nodes in the network are clustered into several subnets, whose functions are the same as the logic token rings in WTRP. Ergen [13] showed a number of topologies extended to the WTRP protocol:

- Hierarchical hybrid schemes – combination of star/tree and ring topologies.
- Token chain – combination of several rings.
- Data forwarding – clustering stations into multiple rings.
- Sensor networks – hierarchical clustering by rings, where ring leaders are connected by tree topology.

Cactus graphs are motivated by models where a tree topology would be irrelevant in telecommunications [8, 28, 42, 46]. Moreover, the above extensions to the architectures form a cactus graph, which is why practical communication networks may have cactus graph topology (Figure 1). A cactus graph is also a planar graph, enabling us to transmit between nodes without having to consider cross-transmissions, and improving the delivery rate by using the two available paths in each cycle. Wang et al. [42] presented a robust and energy efficient routing scheme using a backup path; therefore, considering all the solutions that use a tree topology as intra-topology and merging them with [42], will lead to cactus topology.

1.1 Cactus Graph in Wireless Sensor Networks - Motivation

An overview of sensor network applications can be seen in Figure 2. One generic type of application for these networks is monitoring, where all the sensors produce relevant information by sensing the area and transmitting the information to a central node called a sink node. In some applications, namely data aggregation, performing in-network fusion of data packets is a useful paradigm.

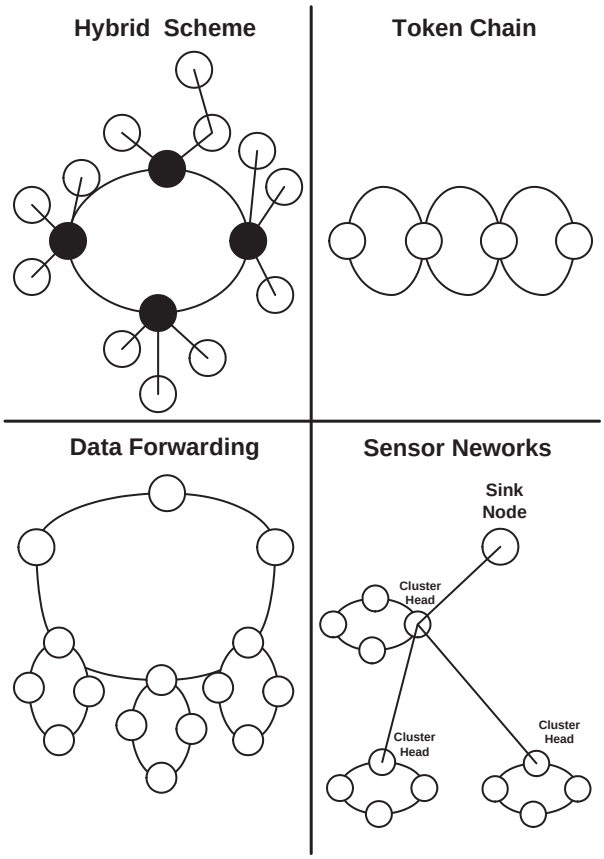

Figure 1: Wireless token ring extension to a cactus graph [13].

However, it is not applicable in all sensing environments. For military applications, such as receiving an image of a battlefield, the data being transmitted by the nodes provide an important point of view. In such situations, it might not be feasible to aggregate the data from different sensors into a single data packet. In those applications all the information is sent to the sink node.

We are assuming that the sink node has the ability to change its position [27,37,40] to improve the performance of the network and does not have energy limitations. For example, a group of soldiers (considered a sink node) collects information from other units in a battlefield. The soldiers may move around, but have to be able to continuously receive data reports [33].

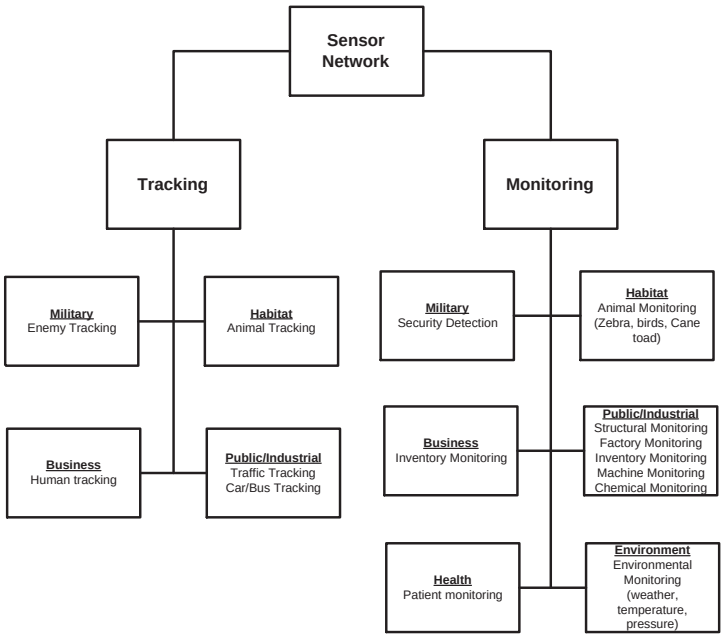

Figure 2: Overview of sensor applications [45].

1.2 Model and Definitions

In general, a network topology is defined by an undirected connected graph $G(V, E, W)$, where V is the node set, E the set of edges between neighboring nodes, and W is a function from E to $\mathbb{R}$, which takes on positive values only. Note that the present paper deals only with unweighted nodes. For each edge $e(u, v) \in E$, $0 < w_e < \infty$ represents edge weight or length, where a length/weight value represents the amount of energy required to transmit one packet from node v to node u along the edge $e(v, u)$. Note that the edges represent logical connectivity between nodes, i.e., there is an edge between the two nodes u and v if they can hear one another.

For a given pair of nodes u and v, $P(u, v)$ denotes a simple path in G connecting u and v, and its length, $d(P(u, v))$, is defined as the sum of weights (lengths) of the edges on $P(u, v)$. Define $d(u, v)$ as the length of a shortest path between u and v, the minimum of the lengths of all paths connecting u and v. For each $v \in V$, define $dist(v) = \max_{u \in V} d(u, v)$, and $sum(v) = \sum_{u \in V} d(v, u)$. A node c is a center of the graph if $dist(c) = \min_{v \in V} dist(v)$, and a node m is a median of the graph if $sum(m) = \min_{v \in V} sum(v)$.

Consider a node v' of the network G. Let T be a spanning tree of G rooted at some node v'. The *transport* of tree network T is defined as the total length of packet transmissions required to deliver packets from all nodes to node v' by a convergecast process on the tree. The maximum *delay* of tree network T is

the maximum length to be traversed by any packet when traveling from node v' to other nodes. The corresponding solution concepts for convergecast and delay constraints have been considered in the literature as median and center [4]. Since the goal is to minimize the transport and delay, we assume without loss of generality that packets follow shortest routes, i.e., T is a tree consisting of shortest paths from v' to all other nodes in G. Hence, we can ignore T and refer to the terms transport and delay of v' in G.

Using the median approach to select the best core node (sink) v', often provides a solution overlooking the nodes at the end of the network (the farthest leaf). The alternative center approach may therefore be applied; that is, choosing the core to be at the center of a spanning tree where the farthest length is the minimum among the nodes. However, locating a core at the center might entail a large increase in length. The problems with using only the center or median as a core have led to a search for a compromise solution concept called *centdian*, where a centdian function at a node is a convex combination of the corresponding median and center functions [18]. The centdian function for node v (given a fixed $\lambda \in [0, 1]$) in the network is defined by

$$D_v(\lambda) = \lambda \cdot dist(v) + (1 - \lambda) \cdot sum(v) \ ; \ 0 \leq \lambda \leq 1 \tag{1}$$

Another possible definition for the centdian function is:

$$D_v(\alpha) = \alpha \cdot dist(v) + sum(v), \alpha = \frac{\lambda}{1 - \lambda}, \ 0 \leq \lambda < 1 \tag{2}$$

Although the two functions differ by a factor of $(1 - \lambda)$, a point that minimizes $D_v(\lambda)$ also minimizes $D_v(\alpha)$. For each nonnegative α, the centdian value, $Cent(\alpha)$, is defined by

$$Cent(\alpha) = \min_{v \in V} D_v(\alpha) \tag{3}$$

Lemma 1 *Given a general network $G(V, E, W)$, the function $Cent(\alpha)$, defined over the range $\alpha \geq 0$, is a monotone, piecewise linear and concave function with at most $n = |V|$ breakpoints. Moreover, when all the terms $\{(dist(v), sum(v)) : v \in V\}$ are given, the sorted sequence of breakpoints of $Cent(\alpha)$ can be generated in $O(n \log n)$ (sequential) time.*

Proof: By definition $Cent(\alpha)$ is the lower envelope (minimum function) of a collection of n linear functions (see Figure 3 for example). To find the breakpoints of $Cent(\alpha)$ in $O(n \log n)$ time, use a standard divide and conquer algorithm, e.g., Sharir and Agarwal [39]. □

The case when the underlying network is a tree has been studied in the literature [18, 19]. First, we note that a tree network has at most two center and two median nodes. Wherever there are two median or center nodes, they are neighbors. For the sake of the next lemmata proved by [18], if there is more than one center or more than one median we select a center c and a median m such that there is no other center or median on the unique path connecting c and m.

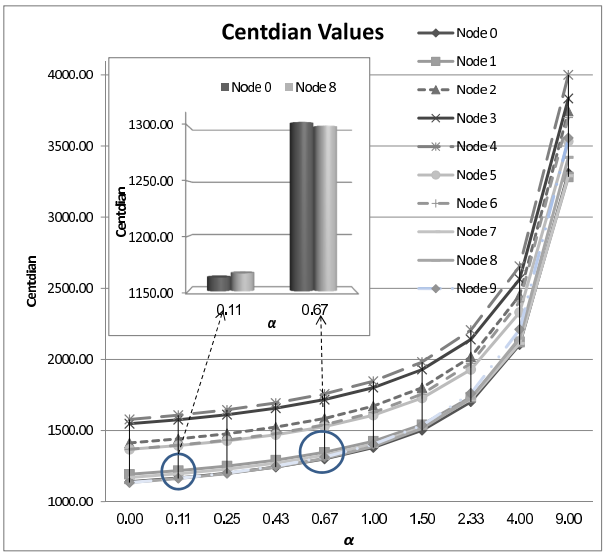

Figure 3: Example of a lower envelope function of the centdian functions. Note that in the small frame the centdian functions of node 0 and node 8 intersect.

Lemma 2 *[18] Suppose that the network G is a tree, given α, a centdian is located at one of the nodes along the unique path $P(m, c)$ connecting the median, m, and the center, c.*

Lemma 3 *[18] Suppose that the network G is a tree, given α, the centdian function, defined on the sequence of nodes along $P(m, c)$, is (discrete) convex, i.e., its sequence of slopes between adjacent nodes is increasing.*

The proof of the lemmata 2 and 3 can be found in [18].

Consider a general network $G(V, E, W)$ and assume that it is connected, have no parallel edges and self-loops. A node v of G is a *cut node* if after removing v and all edges incident to it the resulting graph is not connected. A graph without a *cut node* is called nonseparable. A *block* of G is a maximal nonseparable subgraph. A *cycle* is a connected graph (or subgraph) in which every node is of degree two [46]. G is called a cactus graph if every block with three or more nodes is a cycle. Equivalently, every pair of cycles has at most one common node.

In this paper we model the network topology as a connected cactus graph $CG = (V(CG), E(CG), W)$ (sometimes called a cactus tree). The node set $V(CG)$ of the cactus graph is partitioned into three subsets. The first is the subset of C-nodes, where a C-node is a node of degree two included in exactly one cycle. The second is the subset of NCG-nodes, where an NCG-node is a node not included in any cycle. The remaining nodes, if any, are referred to as H nodes, or hinges [5].

A cycle block (CB), i.e., a block that is a cycle consisting of say, p nodes, will be denoted by a clockwise sequence of its nodes, i.e., $R = (r_0, r_1, r_2, ..., r_{p-1})$. A *subtree* is a connected subgraph induced by a subset of NCG- and H-nodes only. A maximal subtree is a subtree for which the subset of NCG- and H nodes defining it cannot be extended. A graft is a maximal subtree with no two H nodes belonging to the same cycle [5]. For convenience we will view each graft of a cactus graph as a block. Hence, each block is either a cycle or a graft. A cycle/graft block containing only one H node is defined as a "leaf" of the cactus graph. A k-cactus graph is a cactus graph with each block containing at most k edges. It is not difficult to see that a cactus graph consists of blocks, where each block is either a cycle or a graft [5], and the blocks are glued by H nodes [5].

We will use the following lemma, which follows from the general results in Chen et al. [6, 7].

Lemma 4 *Let $CG = (V(CG), E(CG), W)$ be a cactus graph. There exist a pair of blocks of CG, B^c and B^m, such that if $v \in V(CG)$ is a center node, then $v \in B^c$ and if v is a median node, then $v \in B^m$.*

We define an H node radius as its *dist* value, and a cycle block radius as the maximum between the *dist* values (radii) of its H nodes and the longest *dist* value in the cycle block itself.

A common way to deal with a cactus graph is to construct a tree $T_{CG} = (V'_{CG}, E'_{CG})$, each block of G and each H node of G are uniquely represented by some node of T_{CG}. Two nodes of T_{CG} are connected by an edge if and only if one of them represents an H node and the other represents a block containing this H node [2, 5, 6]. Figure 4 shows an example for constructing a tree from a cactus graph.

For our cactus network we assume that all the nodes share the same frequency band, and time is divided into equal size slots that are grouped into frames. Thus, the study is conducted in the context of TDMA (Time Division Multiple Access). In TDMA wireless sensor networks, a transmission scenario is valid if and only if it satisfies the following three conditions: First, a node is not allowed to transmit and receive simultaneously. Second, a node cannot receive from more than one neighboring node at the same time, and a node receiving from a neighboring node should be spatially separated from any other transmitter by at least some distance D. However, if nodes use unique signature sequences (i.e., a joint TDMA and CDMA (Code Division Multiple Access) scheme), then the second and third conditions may be dropped, and the first condition only characterizes a valid transmission scenario. Thus, our MAC layer is based on TDMA scheduling [9, 12, 43], such that collisions and interferences do not occur. In the case where we are using Aloha, CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) or 802.11 as the MAC layer, we are assuming that after a finite number of tries (in case of collision, error, failure) the node will succeed in transmitting the message.

The entities in the network communicate with each other by message passing. Message passing is the paradigm of communication where messages are sent

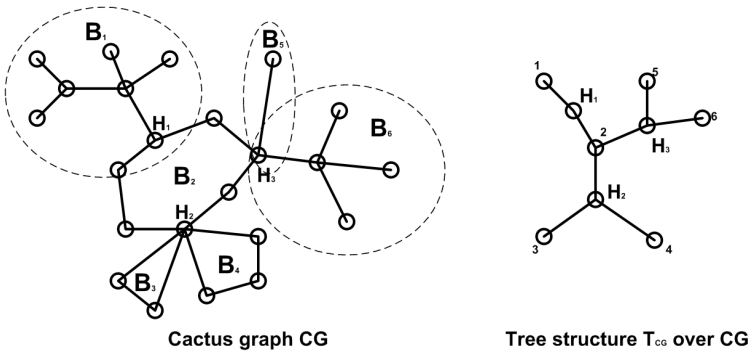

Figure 4: A cactus graph and its corresponding tree structure [2]. The cactus graph has three H nodes (H_1, H_2, H_3), six blocks $(B_1, ..., B_6)$, where B_1, B_5, B_6 are graft blocks and B_2, B_3 and B_4 are cycle blocks, nine C nodes (the nodes of B_2, B_3 and B_4, excluding the H nodes), and ten NCG nodes (the nodes of B_1, B_5, B_6 excluding the H nodes).

from a sender to one or more recipients. Forms of messages include (remote) method invocation, signals, and data packets. Synchronous message passing systems require the sender and receiver to wait for each other to transfer the message. That is, the sender will not continue until the receiver has received the message. Asynchronous message passing systems deliver a message from sender to receiver, without waiting for the receiver to be ready [25]. Therefore, we also assume the following: The network is asynchronous, where each node can start the algorithm at any time or upon receiving a message. Messages are guaranteed to be received within some predefined delay and processes have comparable computational speeds. In a time unit a node can receive messages, perform local computation, and send some messages (broadcast for example). Each node has a unique ID in range $[0, 1, \ldots n-1]$. The computation time of each node is negligible compared to the send/receive times [35]. The "free" calculation is bounded in time; the nodes cannot calculate the entire topology of the network free but can do some calculations such as calculate its sum and dist value. When a node is sending a message to its neighbors (using the wireless link), it will use broadcast transmission. Moreover, when a node is sending a message to one of its neighbors it will use the unicast transmission and the ID of the neighbor as the MAC address. Moreover, leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes). Before the task is begun, all network nodes are unaware of which node will serve as the "leader," or coordinator, of the task. After a leader election algorithm has been run, each node throughout the network recognizes a particular, unique node as the task leader [1, 16].

1.3 Related Work

As defined above, a center node in a cactus graph minimizes the function $dist(v)$. Lan et al. [29] studied the center problem on general cactus graphs and showed a linear time algorithm (linear time algorithms for 3-cactus graphs are reported in [22]). Recently, Ben-Moshe et al. [2] studied a more general model of general cactus graphs where nodes are associated with nonnegative weights. For this general model, they [2] presented an $O(n \log n)$ time algorithm to solve the node weighted 1-center problem, and an $O(n \log^3 n)$ time algorithm to solve the continuous node weighted 2-center problem. Zmazek and Zerovnik [46] presented a linear algorithm estimating the traffic on a cactus graph, computing the sum of all delays on cactus graphs. Das and Pal [8] found the maximum and minimum heights spanning trees on a cactus graph in linear time. Note, however, that all the algorithms above cannot be applied directly for wireless sensor networks since these algorithms are sequential and not based on local updates.

A median problem in a cactus graph seeks a node minimizing the function *sum*. Similarly to the case of the center, there are linear time algorithms for solving the median problem on 3-cactus graphs [23]. Lan and Wang [28] showed that the median problem in 4-cactus graphs can be solved as efficiently as on trees. Burkard and Krarup [5] presented a linear time algorithm for the median problem in a cactus graph with positive and negative node weights. Recently, Hatzl [21] presented a linear time algorithm for the median problem on wheel graphs and cactus graphs, where a wheel graph is a graph consisting of a cycle of order $p-1$ and an additional node that is connected by an edge to each of the cycle nodes. For more information on *median* and *center* problems in cactus graphs see [2, 21, 29].

The centdian problem is well known in the context of the facility location problem, see [17–19, 36]. Dvir et al. [3, 10, 11] were the first to deal with the centdian function as expressed by Eq. (1) in the context of wireless ad hoc networks and wireless sensor networks.

1.4 Our Contributions

As mentioned, all existing cactus graph algorithms may not be applied directly to sensor networks since they are not based on local updates. To the best of our knowledge, no sequential or distributed algorithm presented so far has been shown to construct a centdian node in a cactus graph. We investigate the properties of a centdian node in a cycle graph and present interesting observations on the behavior of a centdian using a lower envelope [24, 39] of the centdian functions of the nodes. Moreover, we show algorithms to determine centdian nodes for all λ values (α values) in a cycle graph (this algorithm is later used to solve the problem on cactus graphs) that work in $O(n \log n)$ time (sequential solution) and $O(n)$ time with $O(n \log n)$ messages (distributed solution). We then consider the behavior of a centdian node in a cactus graph, and present new efficient algorithms for constructing all centdian nodes in cactus graphs

that run in $O(n \log n)$ time (sequential solution) and $O(n)$ time with $O(n \log n)$ messages (distributed solution).

2 Centdian in Cycle Graph

The purpose of this section is to explore the centdian nodes' behavior in a cycle graph. We present algorithms to determine centdian nodes for all λ values (α values) in a cycle graph that works in $O(n \log n)$ time (sequential solution) and $O(n)$ time with $O(n \log n)$ messages (distributed solution). These algorithms are later used to solve the problem on cactus graphs.

2.1 Finding a Centdian Node in a Cycle Graph: Sequential Algorithm

We first compute the values of $sum(v)$ for all nodes v of the cycle graph in linear time based on the algorithm in [5]. Then, we compute the values of $dist(v)$ for all nodes of the cycle graph in linear time based on the algorithm in [29].

It is well known that in the case of a tree topology both the *dist* and *sum* functions are convex. In particular, every local minimum solution of the *dist* (*sum*) function is a global minimum. Also, the set of nodes minimizing the *dist* (*sum*) function consists of at most a pair of nodes. Moreover, these nodes are neighbors, i.e., they induce a connected subgraph (edge). In contrast, in the case of cycle topology, local solutions are not necessarily global and the set of minimizers of the *dist* (*sum*) function does not necessarily induce a connected subgraph, as illustrated in Figures 5 (a) and 5 (b).

Thus, contrary to the case of tree topology, where we can find the centdian node by starting from the median node and searching for the optimal node with the minimum centdian function on the path between the median and center nodes [11], we cannot apply the same technique to cactus graphs. In [18], Halpern proved that while $\lambda < 0.5$, the centdian node coincides with the median node in a tree network. As can be seen in Figure 5 (c), this no longer holds for a cycle graph, which leads us to the conclusion that in a cycle graph a node $t \neq m, c$ can be a centdian node in some range of λ, $0 < \lambda < 1$.

Figure 5 (d) is an example of a cycle graph with ten nodes (The centdian functions of the nodes can be seen in Figure 3). The edge length is depicted on the edge, and the nodes are indexed. The node is shown as black in the figure if for a certain value of λ it serves as the centdian of the cycle graph. Moreover, the largest such λ value is associated with a node. Nodes without a small circle (gray color) do not serve as centdian nodes for any range of λ. For example, node 7 is the centdian node when $0.84 < \lambda < 0.87$, but node 4 is not a centdian node for any range of λ. The center node is node 1 and the median node is node 9. Finally, given a cycle graph with n nodes, we give an $O(n \log n)$ algorithm to construct the centdian nodes for all values of α ($\alpha = \frac{\lambda}{(1-\lambda)}$).

Lemma 5 *For any range of α, all the centdian nodes can be found in $O(n \log n)$ time in a cycle graph.*

Proof: Using the algorithms by Burkard and Krarup [5], and Lan et al. [29], in $O(n)$ time we can compute $sum(v)$ and $dist(v)$ for all nodes. Thus, the result follows from Lemma 1 in the previous section. □

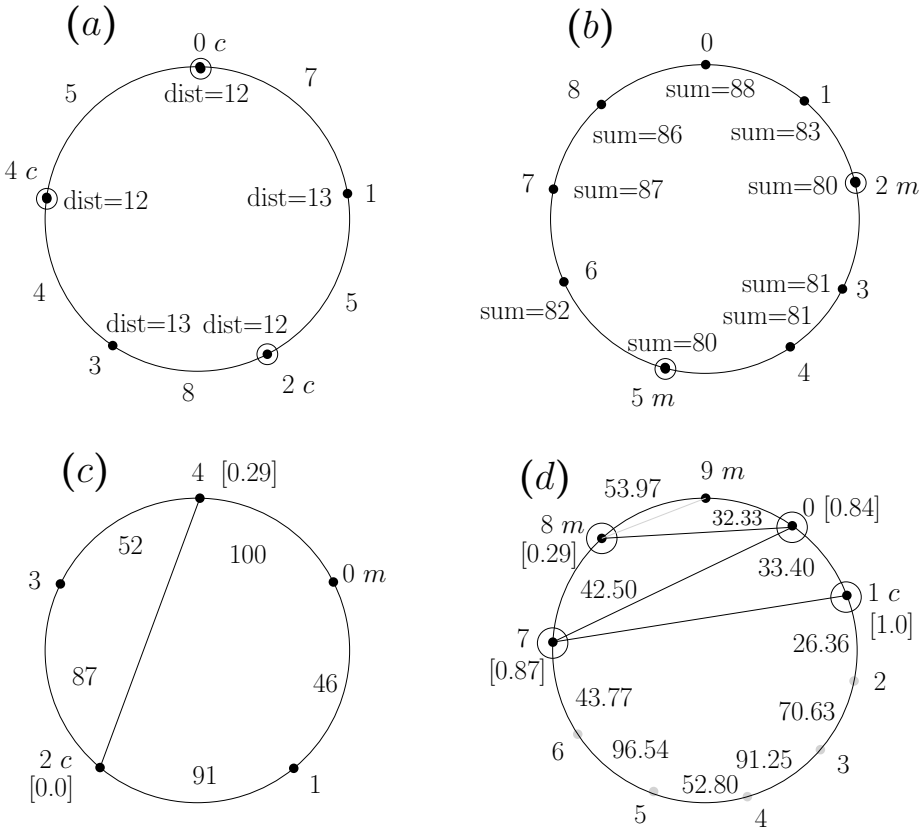

Figure 5: (a,b) An example presenting the non-convex property of the *dist* and *sum* functions and the non-connected center/median nodes in a cycle graph. (c) An example of $\lambda < 0.5$. (d) An example of a zigzag centdian behavior in a cycle graph.

Remark: A centdian node in a cycle graph can zigzag between upper and lower paths $P(m, c)$ for various values of λ, see Figure 5 (d). Meaning, for various λ values the centdian node is not in the same path between the median and the center ($\lambda = 0, \lambda = 1$). For example, in Figure 5 (d) we can see that the centdian node starts at node number 9, then moves to node 8, jumps to 0 and then to 7. The line between node numbers $9, 8, 0, 7$ creates a zigzag

line. However, in some cases, we found that the order behavior might be self intersecting, such example is shown in Figure 6. □

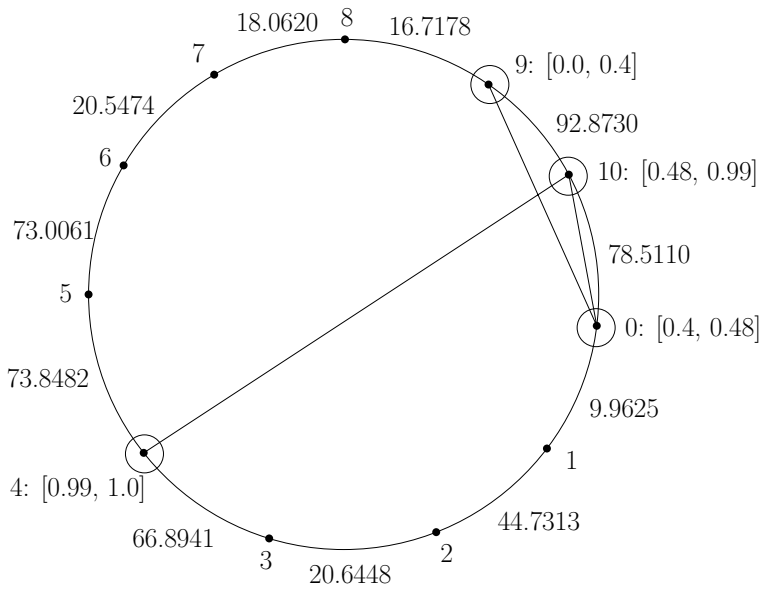

Figure 6: The centdian node is 9 while $0 \leq \lambda < 0.4$, node 0 is a centdian, while $0.4 \leq \lambda < 0.48$, node 10 is a centdian, while $0.48 \leq \lambda < 0.99$, node 4 is a centdian while $0.99 \leq \lambda < 1$. The centdian node does not creates a regular zigzag movement, as in Figure 5 (d), where the centdian creates regular zigzag movement.

2.2 Finding a Centdian Node in a Cycle Graph: Distributed Algorithm

In the following section we present a distributed algorithm to find centdian nodes in a cycle graph in $O(n)$ time with $O(n \log n)$ messages. In order to do this, we have to find a way to provide distributed solutions for each of the algorithms in Section 2.1. Algorithm 1 summarizes the distributed algorithms and presents details (lines 2–15) about one of the distributed algorithms that finds the *sum* and *dist* values for each of the nodes in the cycle graph.

```
   Input: Cycle Graph
   Output: All centdian nodes
 1 Find a leader in the cycle graph based on [1]
 2 foreach u ∈ V do
 3 |   if leader then
 4 |   |   send to both neighbors
   |   |   info(leader = id, tempDistVector = [e], nodes = 1)
 5 |   else
 6 |   |   if received info message then
 7 |   |   |   tempDistVector_u = [tempDistVector, e]
 8 |   |   |   forwards info(leader = id, tempDistVector_u, nodes++)
 9 |   |   end
10 |   end
11 end
12 foreach u ∈ V do
13 |   Calculate the sum and dist values
14 |   Calculate the centdian function
15 end
16 Find the medians and centers of the cycle
17 Find the lower envelope of the centdian function
18 Find all centdian nodes of the cycle graph
```

Algorithm 1: Cycle Graph Algorithm

As the basic step of our algorithm, we find a leader and the size of the cycle graph by using the distributed algorithm given by Awerbuch [1] in $O(n)$ time and $O(n \log n)$ messages.

To find the median and center nodes in a distributed manner [26], the leader sends $\underline{info(leader = id,\ tempDistVector = [e],\ nodes = 1)}$ messages to its neighbors, where *id* is the unique ID of the leader and e the length of the edge between the leader and its neighbor. When node u receives a $\underline{info}$ message from one of its neighbors, it saves the $tempDistVector$ and $nodes'$ values, and sends a $\underline{info(leader = id, tempDistVector = [tempDistVector,\ e],\ nodes++)}$ message to another neighbor. The $tempDistVector$ holds the edge lengths between the nodes receiving the $\underline{info}$ message. When node z receives $\underline{info}$ messages from both its neighbors it has complete knowledge of the cycle graph. Using the vectors from both messages, node z can calculate its *dist* and *sum* values in the cycle graph. After the calculation, node z sends the $\underline{info}$ messages to its neighbors. Thus, at the

end of this process each node knows its *dist* and *sum* values.

We then need to choose the nodes with the minimum *dist* values to be the centers of the cycle graph and the nodes with the minimum *sum* values to be the medians of the cycle graph. Therefore, the leader needs to send a message to the cycle graph to collect the *dist* and *sum* values and to recognize the median/center nodes. By receiving a *median/center* message from the leader of the cycle graph, each node knows whether it serves as a median/center node. From all the median nodes, the leader chooses the median with the minimum ID as the new leader (median leader) of the cycle graph.

Finally, the median leader starts the following operation of computing the lower envelope for finding the centdian nodes of the cycle graph, using the knowledge that each node has its own *dist* and *sum* values. The median leader starts a procedure to build the lower envelope centdian functions of the cycle graph by sending a $LowerEnv(LF)$ message to one of its neighbors, where the $LF = D_{leader}(\alpha)$. When node u receives a $LowerEnv(LF)$ message it merges its centdian function with the current lower envelope, calculates the new lower envelope function, and propagates a $LowerEnv(LF)$ message to another neighbor. At the end of this process the leader has obtained the lower envelope function and can find all the centdian nodes in the cycle graph. Note that, instead of involving the entire network in the operation of computing the lower envelope centdian functions, the leader can compute the entire lower envelope by himself and broadcast the information.

It is easy to see that our distributed algorithm, in fact, emulates the execution of the sequential solution. Thus, the correctness of this distributed solution follows immediately. The running time of the different parts of the algorithm that finds the centdian nodes in a cycle graph is $O(n)$ that uses $O(n \log n)$ messages. First, we start with finding a leader in the cycle graph and the cycle graph size using algorithm in [1] in $O(n)$ time and $O(n \log n)$ messages. In order to find the centdian nodes in a cycle graph, each node calculates its *dist* and *sum* values; this can be done in $O(n)$ time using $O(n)$ messages when the computed leader sends the required information to all nodes as explained above. As the result, a new, median leader is elected. Then, to calculate the lower envelope of the centdian functions of the nodes in the cycle graph, we need additional $O(n)$ time using $O(n)$ messages by sequential scanning all the nodes starting from median leader and merging their centdian functions. From the computed lower envelope function the centdian nodes for a given α range can be calculated.

3 Centdian in Cactus Graph

The purpose of this section is to explore the behavior of the centdian nodes in a cactus graph, and design an algorithm to find them. Specifically, we will extend the results on tree graphs in [18], and present new efficient algorithms for constructing all centdian nodes in cactus graphs that use the cycle graph algorithm as its last step, run in $O(n \log n)$ time (sequential solution), and in $O(n)$ time with $O(n \log n)$ messages (distributed solution).

3.1 Finding a Centdian Node in a Cactus Graph: Sequential Algorithm

In what follows we present a number of observations and then explain the sequential algorithm for finding the block containing the centdian nodes (centdian block) for a given α, in a cactus graph and finding all centdian nodes in the cactus graph in $O(n \log n)$ time.

As stated in Lemma 4 there exists a single block of the cactus graph containing all the median nodes (median block). Similarly, from Lemma 4 , there exists a single block of the cactus graph containing all the center nodes (center block). The effort to find the median and center block is linear [2, 6, 7].

Suppose, first, that the median block and the center block coincide. From the definition of centdian function it follows that for each α, all the respective centdian nodes are also in the common block. We can then apply the above results to construct the centdian function in $O(n \log n)$ time.

Suppose next that the median block and the center block do not coincide.

Definition 3.1 *Let P be the unique path connecting the pair of nodes of T_{CG} corresponding to the median block and center block. The set of blocks of the cactus graph corresponding to the nodes of P, including the median and the center blocks, is called the sausage (S) of the cactus graph (as depicted in Figure 7).*

Lemma 6 *For any α value, the set of centdian nodes of the cactus graph is located on the sausage subgraph.*

Proof: Consider a node v not in a block belonging to S, then there is a block $B' \in S$ and H node $v' \in B'$ such that v' is on every path connecting v to the median block or the center block. In particular, from [6, 29], $dist(v') < dist(v)$, $sum(v') < sum(v)$, and from the definition of the centdian function $\alpha \cdot dist(v) + sum(v) > \alpha \cdot dist(v') + sum(v')$. □

The previous result implies that all centdian nodes are in the sausage. Although not all nodes of the sausage are centdian nodes, the next theorem states that each H node of the sausage is a centdian for some nonempty range range of α.

Theorem 1 *Each H node on a shortest path in the sausage connecting the median block with the center block is a centdian for some value of α.*

Proof: We use the notation in Figure 7, where H_1 is the hinge node in the median block, say B_0, H_1 and H_2 are connected via block B_1, etc., and H_t is the hinge node contained in the center block, say B_t as presented in Figure 7. Let $P = \{(dist(v), sum(v)) : v \in G\}$, and let $LCH(P)$ be the lower convex hull of the set P in the $(dist, sum)$ plane.

Let V' be the set of all nodes in the sausage.

Observation 1: Let $v' \in V'$ be in B_i, the block of the sausage connecting H_i with H_{i+1}. If $d(v', H_{i+1}) \geq d(H_i, H_{i+1})$, then $dist(H_i) \leq dist(v')$ and

$sum(H_i) < sum(v')$. If $v' \in B_i$ contributes a point to $LCH(P)$, then $dist(H_i) \leq dist(v') \leq dist(H_{i+1})$, and $dist(v') = d(v', H_{i+1}) + dist(H_{i+1})$.

Let V_i' be the set of nodes in B_i, and let P_i be the respective points in the $(dist, sum)$ plane.

Suppose by contradiction that some H node does not contribute to $LCH(P)$. For convenience of notation, suppose without loss of generality that this node is H_2 (Figure 7). Then there are nodes $u' \in V_1'$ and $u" \in V_2'$, such that u' contributes to $LCH(P_1)$, $u"$ contributes to $LCH(P_2)$, and

$$\frac{(sum(u") - sum(H_2))}{(dist(H_2) - dist(u"))} < \frac{(sum(H_2) - sum(u'))}{(dist(u') - dist(H_2))}$$

Let n_1 be the number of nodes which are connected to H_2 (only) via H_1. Let B_1 be the block containing H_1 and H_2. Let V_1 be the set of all nodes in G, which are connected to H_1 and H_2 via some node of B_1, which is neither H_1 nor H_2. (Note that V_1 does not include H_1 and H_2.)

Let n_2 be the number of nodes which are connected to H_1 (only) via H_2. Let B_2 be the block containing H_2 and H_3. Let V_2 be the set of nodes in G, which are connected to H_2 and H_3 via some node of B_2, which is neither H_2 nor H_3.

With the above notation we have

$$sum(H_2) - sum(u') = n_1 d(H_1, H_2) - n_1 d(u', H_1) - n_2 d(u', H_2) + \sum_{v \in V_1} (d(v, H_2) - d(v, u')) \leq$$

$$n_1 d(u', H_2) - n_2 d(u', H_2) + \sum_{v \in V_1} d(u', H_2) =$$

$$(n_1 - n_2 + |V_1|) d(u', H_2) = (n_1 - n_2 + |V_1|)(dist(u') - dist(H_2)).$$

The inequality follows from the triangle inequality, and the last equality follows from Observation 1. Next,

$$sum(u") - sum(H_2) = (n_1 + |V_1| + 1) d(u", H_2) + (n_2 - 1 - |V_2|) d(u", H_3) -$$

$$(n_2 - 1 - |V_2|) d(H_2, H_3) + \sum_{v \in V_2} (d(v, u") - d(v, H_2)) \geq$$

$$(n_1 + |V_1| + 1) d(u", H_2) - (n_2 - 1 - |V_2|) d(u", H_2) +$$

$$\sum_{v \in V_2} (-d(u", H_2)) = [(n_1 + |V_1| + 1) - (n_2 - 1 - |V_2|)] d(u", H_2) +$$

$$|V_2|(-d(u", H_2)) = (n_1 - n_2 + |V_1| + 2) d(u", H_2) \geq$$

$$(n_1 - n_2 + |V_1| + 2)(d(H_2, H_3) - d(u", H_3)) =$$

$$(n_1 - n_2 + |V_1| + 2)(dist(H_2) - dist(u")).$$

All the inequalities follow from the triangle inequality, and the last equality follows from Observation 1. Note that since H_1 is on the path connecting H_2 to

the median block, $sum(H_2) \geq sum(H_1)$, which implies that $n_1 - n_2 + |V_1| \geq 0$. Finally, from the above, we have

$$\frac{(sum(u") - sum(H_2))}{(dist(H_2) - dist(u"))} \geq (n_1 - n_2 + |V_1| + 2) >$$

$$(n_1 - n_2 + |V_1|) \geq \frac{(sum(H_2) - sum(u'))}{(dist(u') - dist(H_2))}.$$

The proof shows that the slope sequence is actually increasing at each H node by an additive term which is at least 2. From the above proof we can actually compute the α range for each hinge node H_i. Specifically, it is sufficient to calculate $\frac{(sum(H_i)-sum(v))}{(dist(H_i)-dist(v))}$, for all nodes in $B_{i-1} \cup B_i, i \in 1, ...H_t$. □

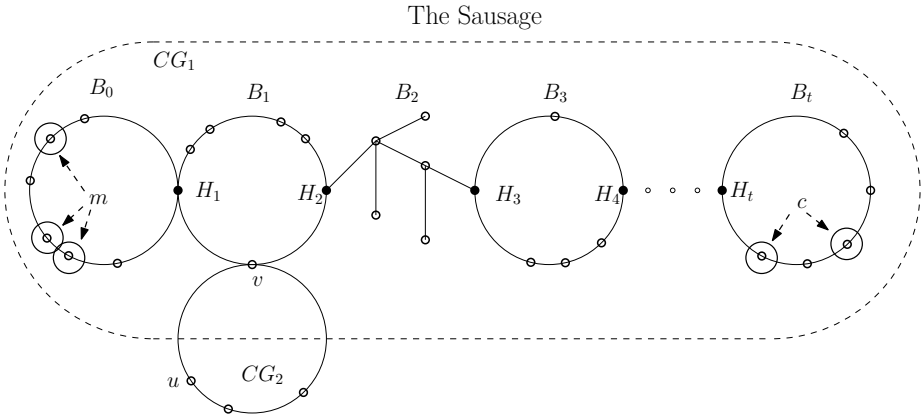

Figure 7: Example of a sausage in a cactus graph, B_0 is the median block, B_t is the center block, the blocks in dashed circle the are the sausage, H_1 is a H node between B_0 and B_1, H_2 is a H node between B_1 and B_2, H_3 is a H node between B_2 and B_3.

Remark: The above proof can also be used to conclude that if B_i is a graft block of the sausage, then each node of the graft which is on the unique simple path connecting the two adjacent hinges, H_i and H_{i+1} is also a centdian for some value of α. □

Let H_i be an hinge node on (any) simple path in the sausage connecting the median block with the center block. We let $[\alpha'_i, \alpha''_{i+1}]$ denote the range of α for which H_i is a centdian node of the cactus graph.

Lemma 7 *For each i, $1 \leq i < t$ and $\alpha'_i < \alpha < \alpha''_{i+1}$, the centdian block corresponding to α is the single block in S containing both H_i and H_{i+1}.*

Lemma 8 *For any range of α, all the centdian nodes can be found in $O(n \log n)$ time in a cactus graph.*

Proof: Using the $O(n)$ time algorithms in [5] and [29], compute $sum(v)$ and $dist(v)$ for all nodes of the cactus graph. Then we need to find the (lower) convex hull of the points ($dist(v)$,$sum(v)$). The extreme points of the lower envelope are the breakpoints of the $Cent(\alpha)$ function. If ($dist(v*)$,$sum(v*)$) is such an extreme point, then the α range connecting this extreme point to node $v*$ is the range between the two slopes connecting this breakpoint to its two neighbors on the hull. All the extreme points on the hull correspond to nodes in the sausage. The sequential time to find the above hull is clearly $O(n \log n)$. In case we have given α, finding a centdian can be done by performing a binary search in $O(\log n)$ time, over the extreme points of the hull (or the respective α ranges). □

3.2 Finding Centdian in a Cactus Graph: Distributed Algorithm

In the following section we present a new distributed algorithm for finding centdian nodes in a cactus graph in $O(n)$ time with $O(n \log n)$ messages of size $O(\log n)$ bits. For this purpose, we define node x as the cactus graph leader, which can be found by using the distributed algorithm in [1]. Algorithm 2 shows the distributed parts of the algorithm. The distributed algorithm follows the same steps as in our sequential solution.

Input: Cactus Graph
Output: All centdian nodes
1 Find a leader in the cactus graph based on [1]
2 Each node checks whether it is an H node based on a Depth First Search tour.
3 Have each H node identify its cycle blocks.
4 Recognize the cactus graph "leaves".
5 Locate the median block based on [21].
6 Locate the center block based on [2].
7 Identify the cactus graph sausage.
8 Calculate the optimal λ range for each node in the sausage, finding all the breakpoints on the convex hull.

Algorithm 2: Cactus Graph Algorithm

Each node in the cactus graph having more than two neighbors needs to determine whether it is an H or an NCG node. The following distributed algorithm recognizes the H nodes.

The main idea is to apply a Depth First Search (DFS) tour to the cactus graph. Node x (leader node) sends a *findH(id)* message to one of its neighbors, where *id* is the unique leader ID. When a node u, having only one neighbor, receives a *findH* message, it sends the message back to its neighbor. When a node v having more than one neighbor receives a *findH* message from one of

its neighbors, it randomly chooses one of its unmarked edges (other than the receiving edge), marks it, and sends the *findH* message from the chosen edge. If a node has no more unmarked edges, it sends a *findH* message back, through the DFS tour backtracking path.

Lemma 9 *After the DFS tour, every node correctly determines whether it is an H node.*

Proof: A node that sends a *findH* message via one of its edges and receives a message back can conclude that both edges are on the same cycle. Since DFS traverses all the edges the claim follows. □

Thus, by performing a DFS tour on the cactus graph a *findH* message travels on the edges no more than $2 * |E|$ times. At the end of the algorithm, each node determines whether it is an H node, and if so, calculates the number of cycles it belongs to and identifies the neighbors in each cycle.

After a node u identifies itself as an H node, we have to compute the number of nodes in the cycle blocks with u and the number of H nodes in each cycle block containing u. Each H node sends a *countCycle*$(id, i = 1, h = 1)$ message to one of its neighbors in each cycle block, where id is the unique node ID, i represents the number of nodes so far in this cycle block, and h is the number of H nodes detected so far in this cycle block. When a C node z receives a *countCycle* message, it sends a *countCycle(id, i++, h)* message to another neighbor. When an H node v receives a *countCycle* message it checks whether $id == id_v$. If so, it saves the i value as the number of nodes and the h value as the number of H nodes in this cycle block. If not, it sends a *countCycle(id, i++, h++)* message to another neighbor belonging to this cycle.

In order to find the median block in a cactus graph in a distributed manner, each H node needs to compute the number of nodes in each of its subgraphs. This can be computed by propagating information from the cactus graph leaves to a root node, and then from the root back to the leaves. At the end, each node will determine the number of nodes in each of its subgraphs. We distinguish between the cases of NCG nodes and H nodes. Our distributed algorithm is based on the assumptions and Lemmata of [6].

An NCG node with one neighbor sends a *subGraphNodes*$(id, i = 1)$ to its neighbor, where id is its unique node ID. Any other NCG node receiving a *subGraphNodes* message waits until it receives *subGraphNodes* messages from all but one of its neighbors. Then, it saves the number of nodes in each of its subgraphs and sends a *subGraphNodes*$(id, \sum i + 1)$ message on its last edge, with the sum of all i values it received plus one (itself).

Each H node has to wait until receiving the information from all the H nodes in each of its cycles and then propagate the information to the rest of the cactus graph. Note that the neighbors of an H node in a graft block are the closest neighbor nodes, while the neighbors of an H node in a cycle block are the H nodes in this cycle. An H node that has a leaf cycle sends a *subGraphNodes*$(id, numInCycle)$ message under the following conditions, where $numInCycle$ is the number of nodes in the subgraph of H:

- If the H node has only one additional graft or cycle (not leaf) block, it sends a $\underline{subGraphNodes(id, numInCycle = i)}$ message to its neighbor in the block, where i is the number of nodes in the leaf cycle.
- If the H node has more than one graft or cycle block, it waits until it receives $\underline{subGraphNodes}$ messages from all but one of its neighbors, saves the information about the number of nodes in each subgraph, and sends a $\underline{subGraphNodes(id, numInCycle = \sum i + 1)}$ message, with the sum of all the i values it receives plus one (itself) to its last neighbor.

When an H node u (without a leaf cycle) receives a $\underline{subGraphNodes}$ message with $id \neq id_u$, u decreases its h value of this cycle by one and checks whether h equals one. If not, u saves the information about its subgraph. If yes, u sends a $\underline{subGraphNodes}$ message under the condition of an H node with leaf cycle as explained above. An H node z receiving $\underline{subGraphNodes}$ messages from all its neighbors concludes that it is the root, saves the information about the number of nodes in each subgraph, and sends its ID as the root ID and the information about the number of nodes in each subgraph to its neighbors. Each node (H or NCG) receiving $\underline{subGraphNodes}$ messages from all its neighbors with their root ID, saves the information about the number of nodes in each subgraph and sends the information to its neighbors. Therefore, at the end of this process, each H node knows the number of nodes in each of its subgraphs. Then, using the algorithm from [21], we can search in a distributed manner for either a cycle block in which all its H nodes have $cut(H) \geq 0$, or for an NCG node v in which $\Delta(v, w) \geq 0$ for all w neighbors of v are the median blocks.

The distributed algorithm for finding the center block in a cactus graph is similar to the above algorithm and based on [2, 7](please refer to the paper for more explanation). The main difference is that each H node propagates its current $dist$ value instead of the number of nodes in each block. At the end, each H node calculates its radius. Then, in each cycle block one of the H nodes is selected as the leader of the cycle block [1], and sends a $\underline{radius}$ message in its cycle block to collect the radii of the H nodes. With this information, each cycle block leader can calculate the radius of its cycle block and check whether its block has more than one subgraph with the maximum radius [2]. Finally, the H leader of the cactus graph can find, in a distributed manner, the cycle block that have more than one subgraph with the maximum radius to be the center block.

The next stage is to find the sausage of a cactus graph. Node u, which is the H leader of the median block, sends $\underline{findSausage(id,\ TTL{=}n)}$ messages to its neighbors. A C or NCG node receiving a $\underline{findSausage}$ message, sends it to all its neighbors. An H node receiving the $\underline{findSausage}$ message, marks the edge receiving the $\underline{findSausage}$ message, decreases the TLL by one, and propagates the $\underline{findSausage}$ message to its neighbors. When the $\underline{findSausage}$ message is received by node v, which is the H leader of the center block, it sends $\underline{findSausage(id,\ TTL{=}n)}$ messages backward with its own ID. Each H node that receives both messages from different blocks (median and center) concludes that it belongs to the sausage.

Lemma 10 *The algorithm correctly determines the sausage.*

Proof: According to the sausage definition, we need to determine the nodes in the blocks between the median and center blocks. Clearly, if a node receives message from headers of both (center and median) blocks, it can conclude that it belongs to the sausage, otherwise it does not belong. □

Finally, we need to find the optimal α range for each node v in the sausage, which is equivalent to find the breakpoint of the convex hull (see Lemma 8). Therefore, after all the nodes belonging to the sausage have identified themselves, they need to compute a convex hull of points ($dist(v)$,$sum(v)$) and not the convex hull of themselves. Thus, each node in the sausage sends to the leader, the H node connecting the median block (H_1 in Figure 7) to the sausage, these values, $dist(v)$ and $sum(v)$ (overall n messages). The leader will compute the convex hull having these values and find the optimal α range, if any, for each node v in the sausage. Then the leader will broadcast the entire solution to all of the nodes in the sausage. This can be done in $O(n)$ time with $O(n \log n)$ messages.

The running time of the different parts of the algorithm that finds the centdian nodes in the cactus graph is $O(n)$ time and $O(n \log n)$ messages. First, each of the algorithms starts with finding a leader in the cycle block/cactus graph and the cycle block/cactus graph size using [1] in $O(n)$ time and $O(n \log n)$ messages. To find the centdian nodes in a cactus graph, each node determines whether it is an H node, and this can be done in $O(n)$ time using $O(n)$ messages. Then, each H node evaluates the number of nodes in each of its subgraphs in $O(n)$ time using $O(n)$ messages, and in additional $O(n)$ time with $O(n)$ messages we can find the sausage S. Finally, to calculate the α range for every node in the sausage we need $O(n)$ time using $O(n \log n)$ messages.

4 Conclusions

In this paper, we have investigated the properties of the behavior of centdian nodes in a cycle graph and presented efficient sequential and distributed algorithms to find centdian nodes in cactus graphs in $O(n \log n)$ time for sequential solution and $O(n)$ time using $O(n \log n)$ messages for distributed solution. It is still unknown whether the $O(n \log n)$ bound for sequentially finding all the centdian nodes, even in a cycle graph, is optimal.

The exact complexity of finding centdian nodes in a cactus graph with weighted nodes is still open. In the following we present a short discussion to shed light on a possible solution. Suppose that each node v is associated with 2 weights, $a(v), b(v) > 0$. Define, $sum(v) = \sum_{u \in V} a(u)d(u, v)$ and $dist(v) = \max_{u \in V} b(u)d(u, v)$ [36]. The definitions (1), (2), (3) depending only on the $dist(v)$ and $sum(v)$ remain the same. If $b(v)$ is constant for all v and equal to 1, our results extend directly to this model, without affecting the complexity. For general nonnegative coefficients, some of the properties still hold. There is a unique median block and a unique center block. $Cent(\alpha)$ is still concave and

piecewise linear with at most n breakpoints. We can apply our approach to generate $Cent(\alpha)$, but the complexity is higher. The effort to compute $sum(v)$ for all $v \in V$, is still $O(n)$ by the algorithm in [5]. However, computing $dist(v)$ for all $v \in V$ seems to be more expensive. For a cycle there is an $O(n \log n)$ algorithm in [2] for finding $dist(v)$ for all v, in the weighted case. For a tree there is an $O(n \log n)$ algorithm in [41]. We are not aware of any published algorithm for a general cactus. However, we suspect that this can be done in $O(n \log^3 n)$ time, by using the above algorithms and the 2-centroid decomposition scheme for series-parallel graphs in [20].

References

[1] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Annual ACM Symposium on Theory of Computing*, pages 230–240, New York, USA, July 1987.

[2] B. Ben-Moshe, B. K. Bhattacharya, Q. Shi, and A. Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237–252, June 2007.

[3] B. Ben-Moshe, A. Dvir, M. Segal, and A. Tamir. Centdian computation for sensor networks. In *TAMC*, pages 187–198, Prague, Czech Republic, June 2010.

[4] S. C. Bruell, S. Ghosh, M. H. Karaata, and S. V. Pemmaraju. Self-stabilizing algorithms for finding centers and medians of trees. *SIAM Journal on Computing*, 29(2):600–614, Oct. 1999.

[5] R. E. Burkard and J. Krarup. A linear algorithm for the pos/neg-weighted 1-median problem on cactus. *Computing*, 60(3):193–215, May 1998.

[6] M. L. Chen, R. L. Francis, J. F. Lawrence, T. J. Lowe, and S. Tufekci. Block-vertex duality and the 1-median problem. *Networks*, 15(4):395–412, Feb. 1986.

[7] M. L. Chen, R. L. Francis, and T. J. Lowe. The 1-center problem: Exploiting block structure. *Transport Science*, 22:259–269, 1988.

[8] K. Das and M. Pal. An optimal algorithm to find maximum and minimum height spanning trees on cactus graphs. *Advanced Modeling and Optimization*, 10(1):121–134, 2008.

[9] B. Deb and B. Nath. On the node-scheduling approach to topology control in ad hoc networks. In *ACM MOBIHOC*, pages 14–26, 2005.

[10] A. Dvir and M. Segal. The (k, l)-coredian tree for ad hoc networks. *Journal of Ad Hoc and Sensor Wireless Networks*, 6(1-2):123–144, 2008.

[11] A. Dvir and M. Segal. Placing and maintaining a core node in wireless ad hoc sensor networks. *Wireless Communications and Mobile Computing*, 10(6):826–842, 2010.

[12] T. A. Elbatt and A. Ephremides. Joint scheduling and power control for wireless ad-hoc networks. In *IEEE INFOCOM*, pages 976–984, 2002.

[13] M. Ergen. WTRP - wireless token ring protocol. M.sc. thesis, Electrical Engineering and Computer Science, Berkeley, California, USA, 2002.

[14] M. Ergen, D. Lee, R. Sengupta, and P. Varaiya. Wireless token ring protocol-performance comparison with IEEE 802.11. In *International Symposium on Computers and Communication*, pages 710–715, Kiris-Kemer, Turkey, July 2003.

[15] M. Ergen, D. Lee, R. Sengupta, and P. Varaiya. WTRP - wireless token ring protocol. *IEEE Transactions on Vehicular Technology*, 53(6):1863–1881, Nov. 2004.

[16] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, Jan. 1983.

[17] J. Halpern. The location of a centdian convex combination on an undirected tree. *Regional Science*, 16:237–245, 1976.

[18] J. Halpern. Finding minimal center-median convex combination (cent-dian) of a graph. *Management Science*, 24:534–544, 1978.

[19] G. Y. Handler. Medi-centers of a tree. *Transportation Science*, 19(3):246–260, 1985.

[20] R. Hassin and A. Tamir. Efficient algorithms for optimization and selection on series-parallel graphs. *SIAM Journal On Algebraic and Discrete Methods*, 7:379–389, 1986.

[21] H. Hatzl. Median problems on wheels and cactus graphs. *Computing*, 80(4):377–393, Sept. 2007.

[22] R. K. Kincaid and T. J. Lowe. Locating an absolute center on graphs that are almost trees. *European Journal of Operational Research*, 44(3):357–372, April 1990.

[23] R. K. Kincaid and O. Z. Maimon. A note on locating a central vertex of a 3-cactus graph. *Computers and Operations Research*, 17(3):315–320, Feb. 1990.

[24] R. Klein. *Algorithmische Geometrie*. Addison Wesley, 1996.

[25] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems*, 12(1):84–101, Jan. 1990.

[26] 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, July 1984.

[27] H. Kwang-il, I. Jeongsik, Y. Yeo-hong, and E. Doo-Seop. Dynamic sink oriented tree algorithm for efficient target tracking of multiple mobile sink users in wide sensor field. In *IEEE Vehicular Technology Conference*, pages 4607–4610, CA, USA, Sept. 2004.

[28] Y. F. Lan and Y. L. Wang. An optimal algorithm for solving the 1-median problem on weighted 4-cactus graphs. *European Journal of Operational Research*, 122(3):602–610, May 2000.

[29] Y. F. Lan, Y. L. Wang, and H. Suzuki. A linear-time algorithm for solving the center problem on weighted cactus graphs. *Information Processing Letters*, 71(5):205–212, Sept. 1999.

[30] D. Lee, R. Attias, A. Puri, R. Sengupta, S. Tripakis, and P. Varaiya. A wireless token ring protocol for intelligent transportation systems. In *IEEE Intelligent Transportation Systems*, pages 25–29, California, USA, 2001.

[31] D. Lee, A. Puri, P. Varaiya, R. Attias, R. Sengupta, and S. Tripakis. A wireless token ring protocol for ad-hoc networks. In *IEEE Aerospace Conference*, Montana, USA, 2002.

[32] X. Lu, G. Fun, and R. Hao. A dynamic token passing MAC protocol for mobile ad hoc networks. In *International conference on Wireless communications and mobile computing*, pages 743–748, British Columbia, Canada, 2006.

[33] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. TTDD: Two-tier data dissemination in large-scale wireless sensor networks. *Wireless Networks*, 11(1):161–175, Jan. 2005.

[34] D. Maniezzo, G. Pau, M. Gerla, G. Mazzini, and K. Yao. T-mah: A token passing mac protocol for ad hoc networks. In *MedHocNet*, Sardegna, Italy, 2002.

[35] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. SPINS: security protocols for sensor networks. In *ACM MobiCom*, Rome, Italy, July 2001.

[36] J. Puerto, A. Tamir, D. Perez-Brito, and A. M. Rodriguez-Chia. The bicriteria doubly weighted center-median path on a tree. *Networks*, 47:237–247, 2006.

[37] K. Sang-Sik, J. Kwang-Ryul, and P. Ki-Il. A model to support mobile users in wireless sensor networks. In *International Conference on Multimedia and Ubiquitous Engineering*, pages 207–213, Seoul, Korea, April 2007.

[38] D. Saxena. Security in wireless sensor networks - a layer based classification. CERIAS Tech Report 2007-04, April 2007.

[39] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and their Geometric Applications.* Cambridge University Press, 1995.

[40] P. Soochang, K. Bongsoo, L. Euisin, L. Donghun, C. Younghwan, and K. Sang-Ha. A novel communication architecture to support mobile users in wireless sensor fields. In *IEEE Vehicular Technology Conference*, pages 66–70, Dublin, Ireland, April 2007.

[41] A. Tamir. Sorting weighted distances with applications to objective function evaluations in single facility location problems. *Operations Research Letters*, 32:249–257, 2004.

[42] H. Wang, P.-Y. Kong, and W. S. K. Guan. A robust and energy efficient routing scheme for wireless sensor networks. In *International Conference Workshops on Distributed Computing Systems*, pages 83–89, DC, USA, July 2006.

[43] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. Algorithms for energy-effcient multicasting in static ad hoc wireless networks. *ACM MONET*, 6(3):251–263, 2001.

[44] S. Xianpu, Z. Yanling, and L. Jiandong. Wireless dynamic token protocol for MANET. In *Parallel Processing Workshops*, pages 1–5, Xian, China, Sept. 2007.

[45] J. Yicka, B. Mukherjeea, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, Aug. 2008.

[46] B. Zmazek and J. I. Zerovnik. Estimating the traffic on weighted cactus networks in linear time. In *International Conference on Information Visualisation*, pages 536–541, London, UK, July 2005.

[47] M. Zuniga, K. Seada, B. Krishnamachari, and A. Helmy. Efficient geographic routing over lossy links in wireless sensor networks. *ACM Transactions on Sensor Networks*, 4(3):1–33, May 2008.