Neighbor Oblivious and Finite-State Algorithms for Circumventing Local Minima in Geographic Forwarding[☆]

Chandramani Singh[a,*], Santosh Ramachandran[b], S. V. R. Anand[c], Malati Hegde[c], Anurag Kumar[c], Rajesh Sundaresan[c]

[a] *Department of ESE, Indian Institute of Science, Bangalore, India*
[b] *Qualcomm Corp. R&D La Jolla, California, USA*
[c] *Department of ECE, Indian Institute of Science, Bangalore, India*

Abstract

We propose distributed link reversal algorithms to circumvent communication voids in geographic routing. We also solve the attendant problem of integer overflow in these algorithms. These are achieved in two steps. First, we derive partial and full link reversal algorithms that do not require one-hop neighbor information, and convert a destination-disoriented directed acyclic graph (DAG) to a destination-oriented DAG. We embed these algorithms in the framework of Gafni and Bertsekas [1] in order to establish their termination properties. We also analyze certain key properties exhibited by our neighbor oblivious link reversal algorithms, e.g., for any two neighbors, their t-states are always consecutive integers, and for any node, its t-state size is upper bounded by $log(N)$. In the second step, we resolve the integer overflow problem by analytically deriving one-bit full link reversal and two-bit partial link reversal versions of our neighbor oblivious link reversal algorithms. We also discuss the work and time complexities of the proposed algorithms.

Key words: Wireless sensor networks, Routing protocols, Link reversal, Distributed algorithms, Finite bit width

[☆] This work was presented in part at the National Conference on Communications (NCC) 2010, IIT Madras, Chennai, India.

[*] Corresponding author
Email addresses: chandra@dese.iisc.ernet.in (Chandramani Singh)

1. Introduction

1.1. Motivation

Consider a wireless sensor network (WSN) with *a single designated sink node*. We shall focus particularly on an example application where the objective of the WSN is to raise an alarm upon detecting an event (e.g., an intruder or an incipient fire). An alarm packet originating at a node near the location of the alarm event has to be routed to the sink node. For such purposes, *geographic routing* [2] is a popular protocol for packet delivery. It is scalable, stateless, and reactive, and does not need prior route discovery. In this protocol, a node forwards a packet to another node within its communication range (hence, called a *neighbor* node) and closer to the destination. Ties can be broken arbitrarily, for example, by using node indices. Such a protocol requires a node with a packet to be aware of its own geographical location, and that of the sink and of its neighbors. To each node, the next hop nodes that are closer to the sink are defined as *greedy* neighbors, and wireless "links" are oriented from the nodes to their greedy neighbors. The resulting routing graph is a *directed acyclic graph* (DAG).

A DAG is said to be *destination-oriented* when there is a directed path in the DAG from any node to the sink. A DAG is *destination-disoriented* if and only if there exists a node other than the sink that has no outgoing link [1]. The disadvantaged node with no outgoing links is said to be *stuck* (as it is unable to forward towards the sink a packet that it receives). A destination-oriented network under geographic routing may be rendered destination-disoriented due to various reasons such as node failures, node removal or radio jamming.. The failure of geographic routing in the presence of stuck nodes is commonly referred to as *the local minimum condition* [3]. Numerous solutions have been proposed in the literature to pull the network out of a local minimum condition (See Section 1.2 for details). *However, all these solutions require knowledge of one-hop neighbors (i.e., adjacent nodes) and their locations.* Maintenance of one-hop

neighbor information, in general, requires periodic transmissions of *keep alive* packets.

We associate each node in the network with a unique numerical value, henceforth referred to as *state*. A link between a pair of neighboring nodes is oriented from the node with the higher state to the node with the lower state. Thus states (of all the nodes) determine the routing graph. The routing graph is clearly acyclic.

Let us consider two natural protocols to determine link directions and if nodes are stuck, in a design based on nodes' states. Nodes could occasionally broadcast *hello* packets in order to determine whether they are stuck or not. Let us tag such a querying node. The node's hello packet contains its state. All its *alive* neighbors with lower states acknowledge the hello packet. If the tagged node (querying node) does not receive any acknowledgment until a fixed timeout period, it concludes that its state is the least among its alive neighbors, i.e., it is stuck. Then, the node updates its state appropriately to reverse some (or all) of the incident links. It also broadcasts the new state to facilitate its neighbors to update the corresponding link directions. Thus all the nodes always have an updated view of the directions of all their links.[1] The nodes may store the updated states of their neighbors. It enables them to make an update if they determine that they are stuck. In an alternative scheme that does not require each node to hold full state information of neighbors, whenever a node broadcasts a hello packet, all its neighbors respond with their full state information. Then the tagged node not only can determine if it is stuck, but can also use states of its neighbors to make an update and get out of its stuck state. As before, it broadcasts the new state to its neighbors. While this scheme saves some memory, it incurs substantial communication overhead for each query (hello packet) broadcast. In either scheme, the link reversal processes

[1] For correct data forwarding to the sink, any two neighbors must have a consistent view of the direction of the link between them. Thus broadcast of the updated state is an intrinsic part of all routing algorithms.

associated with all the nodes reach an equilibrium when each node has at least one directed path to the sink (i.e., none of them is stuck).

An update protocol is called *neighbor oblivious* if the updating node does not need to know the *exact* values of states of its neighbors. Neighbor oblivious protocols do not incur the overhead of discovery of full state information of neighbors, and thus save precious communication time and energy. An update protocol is called *finite state* if node states always take values in finite set. The space size can potentially be a function of the network size.

Gafni and Bertsekas [1] proposed a general class of distributed *link reversal algorithms* for converting a destination-disoriented DAG into a destination-oriented DAG. They also described two representative algorithms, *full link reversal* and *partial link reversal*, of their general class. Henceforth, we refer to their algorithms as *GB algorithms*. In the GB algorithms, a stuck node's update depends on the exact values of states of its neighbors. Secondly, the correctness of the GB algorithms relies on the fact that states are a priori not bounded, and nodes' state grow without bound as the algorithm proceeds. Thus the GB algorithms are *neither neighbor oblivious nor finite state*.

Our work is motivated by the question: Are there *distributed, finite-state, neighbor oblivious* protocols that can pull a network out of its local minimum condition and render it destination-oriented?[2]

1.2. Related Literature

Kranakis et al. [4] introduced geographic routing protocols for planar mobile ad hoc networks, called *compass routing* or *face routing*. This technique guarantees delivery in a connected network, but requires a priori knowledge of

[2] One simple neighbor oblivious algorithm is to always make a stuck node increment its state (taking integer values) by unity. This algorithm renders the network destination-oriented but requires a huge number of updates. In particular, it is neither full link reversal nor partial link reversal. Recall that each updating node broadcasts its state to determine if it stuck, and then waits for a timeout period for acknowledgments. Consequently, this simple algorithm results in significant energy expenditure and delay, and hence, is not desirable.

full neighborhood. Karp and Kung [2] presented *greedy perimeter stateless routing* (GPSR) which also ensures successful routing over planar networks. Kalosha et al. [5] addressed a beaconless recovery problem where the local planar subgraph is constructed on the fly. Chang et al. [6] presented *route guiding protocol* (RGP), a shortest path routing protocol to bypass voids, but that also requires communication of current states among neighbors. Yu et al. [7] discussed a void bypassing scheme when both source and sink nodes are mobile. Leong et al. [8] presented a new geographic routing protocol called *greedy distributed spanning tree routing* (GDSTR). GDSTR employs convex hulls which require maintaining topology information. Casari et al. [9] proposed *adaptive load-balanced algorithm* (ALBA), another greedy forwarding protocol for WSNs. Some other algorithms developed for mobile adhoc networks include *destination sequenced distance vector* (DSDV) routing [10], *wireless routing protocol* (WRP) [10], *dynamic source routing* (DSR) [11] and *node elevation ad hoc routing* (NEAR) [12]. All the above algorithms require neighbor information at a stuck node, and some even require more extensive topology information (e.g., [8]).

Gafni and Bertsekas [1] introduced a general class of link reversal algorithms to maintain routes to the destination. They also presented two particular algorithms, the full link reversal algorithm and the partial link reversal algorithm. The GB algorithms were designed for connected networks. In a partitioned network, GB algorithms lead to infinite number of state updates without ever converging. Corson and Ephremides [13] presented *lightweight mobile routing* (LMR), a variant of GB link reversal algorithms. Park and Corson [14] proposed *temporally-ordered routing algorithm (TORA)* for detecting and dealing with partitions in the networks. TORA is also an adaptation of GB partial link reversal algorithm and employs extended states that include *current time* and *originator id*. GB link reversal algorithms have also motivated several *leader election* algorithms which are an important building block for distributed computing, e.g., mutual exclusion algorithms or group communication protocols. Malpani et al. [15] built a leader election algorithm on the top of TORA for mobile networks. Ingram et al. [16] proposed a modification

of the algorithm in [15] that works in an asynchronous system with arbitrary topology changes. All these link reversal algorithms employ state variables that either require infinitesimal precision (e.g., current time) or grow unbounded, thus imposing enormous memory requirements. Further, state updates in these algorithms require frequent information exchanges among neighboring nodes, and also network wide clock synchronization, thus imposing signification communication overhead. These drawbacks render the above algorithms unsuitable for large mobile networks with lightweight mobile nodes. We focus on connected ad hoc networks with single destination and develop neighbor-oblivious and memory-savvy link reversal algorithms.

Busch and Tithapura [17] analyzed GB algorithms (full and partial link reversal), and provided asymptotic upper and lower bounds on the work (number of node reversals) and the time needed until these algorithms converge to a destination oriented DAG.

Charron-Bost et al. [18] proposed a new framework for link reversal based on binary *link labels* as opposed to GB algorithms whose link reversals are based on *node labels* (i.e., states). Recall that in GB algorithm dynamic node states are used to establish link directions, and also to selectively reverse them. On the other hand, Charron-Bost et al. [18] assumed the existence of some mechanism that informed nodes of initial directions of all links incident on them, and employed link-labeling only to decide which of these links should be reversed. After each reversal, the nodes are somehow aware of the link directions of all links incident on them. See next subsection on the advantage of the node-based operation. Notwithstanding this difference, from an operational point of view, the GB full and partial link reversal schemes are special cases of the class of schemes proposed in [18]. The authors in [18] analyzed work complexities of any arbitrary node in a routing graph under their class of schemes (including the GB full and partial reversal algorithms). In subsequent works, Charron-Bost et al. presented the exact expressions for the time complexities of any arbitrary node under the GB full and partial reversal algorithms in [19] and [20] respectively.

1.3. Our Contributions

We focus on connected ad hoc networks (e.g., WSNs) with a single destination.[3] We propose neighbor oblivious full and partial link reversal (NOLR) algorithms in which a stuck node does not need full information on states of one-hop neighbors to execute its state update. However, as discussed earlier, a node still has to communicate with its neighbors in order to determine if it is stuck. But this communication only involves a *hello* packet and its acknowledgments, and thus is "lightweight". We then embed our NOLR algorithms into the framework of the GB algorithms. The embedding provides a method to assert that our proposed algorithms render the network destination-oriented.

In GB and NOLR algorithms, the state spaces are (countably) infinite. The reason is that in both the algorithms each node's state grows without bound with the number of link reversals. The algorithms therefore cannot be realized in a real operating environment with only a finite number of bits to represent states, when repeated link reversals may be encountered. We show that simple modifications of our NOLR algorithms result in finite-state link reversal algorithms. At each node, in addition to the initial state, the full link reversal algorithm requires only a one-bit dynamic state and the partial link reversal algorithm requires only a two-bit dynamic state.

We now compare our work with that of Charron-Bost et al. [18]. These authors assume that nodes can explicitly determine the directions of the incident links and and can explicitly change these directions (see Assumptions (a) and (b) in [18, Page 147]). Their schemes are link-centric. On the other hand, our link reversal algorithms are based on node states, and the link directions are directly inferred from these states. The link-centric schemes require one bit per link to implement link reversal. In a real network, the incident nodes must however hold these link-labels, and after each reversal, must communicate the new link

[3] If routing to multiple destinations is required, for each destination, a logically separate copy of our algorithm should be run. This limitation is inherent to the class of GB link reversal algorithms (see [1, 13, 14, 15, 16]).

labels and link directions to the corresponding adjacent nodes (neighbors). This requires $O(N)$ storage and significant communication overheads at each node in the worst case. As we will see below, our node-centric algorithms require each node to store only $\log N$ bits for the node index and an additional two bits to execute our proposed link reversal schemes. Moreover, in our schemes, the nodes do not need to communicate individually to their neighbors after a reversal; they just broadcast the new state.

It must however be noted that the evolutions of the routing graph under our full and partial reversal algorithms are identical to the evolutions under the corresponding schemes of Charron-Bost et al. [18]. So, our convergence results may also follow from [18, Corollary 3.8], if one could formally establish an analogy between our framework and protocols and those in [18]. This however is not obvious. We instead embed our algorithms within the GB framework in order to establish the algorithms' correctness and convergence. Due to operational equivalence of the corresponding algorithms, the asymptotic work complexity bounds of [17] and the exact work and time complexity results of [18, 19, 20] apply to our algorithms as well.

1.3.1. Assumptions

We assume that new nodes or links are not added to the existing network. Our framework does not apply to mobile settings where the connection topology keeps changing. These assumptions are identical to the assumption in [18] that the underlying undirected graph (called *support*) does not change with time. The last section contains a discussion of how addition of new nodes or links affects our algorithms. Our algorithms also rely on the assumptions that the nodes are equipped with distinct indices belonging to an ordered space and that each node knows $h_{\max}$, the maximum of the initial node heights (see Section 3.1). Finally, we also assume that broadcasts and acknowledgements are received without error.

1.4. Organization of the Paper

The rest of the paper is organized as follows. In Section 2 we provide an overview of the GB algorithms. In Section 3 we discuss full link reversal. We begin with the NOLR proposal, but with a countably infinite state space. Then, we make an observation that renders the NOLR algorithm into a finite-state algorithm without loss of correctness. In Section 4, we address partial link reversal, and pass through the same trajectory as for full-link reversal – an NOLR algorithm with infinite states followed by a finite-state version. In Section 5, we discuss the work and time complexities of the proposed algorithms. We end the paper with some concluding remarks in Section 6.

2. Overview of GB Algorithms

Consider a WSN with a designated destination node and nondestination nodes $\{1, 2, \ldots, N\}$. The nodes are assumed to have static locations. Two nodes are neighbors if they can directly communicate, and then we say that there is a *link* between them. Link reversal schemes can be used in geographic forwarding by assigning unique states, $a_1, a_2, \ldots, a_N$, to the nodes. The states are totally ordered by a relation $<$ in the sense that for any two nodes i and j, either $a_i < a_j$ or $a_j < a_i$, but not both. These states are used in assigning routing directions to links. The link between a pair of neighbors is always oriented from the node with the higher state to the node with the lower state.

In GB algorithms, the state associated with a node i is a pair of numbers (h_i, i) for full reversal and a triplet of numbers (p_i, h_i, i) for partial reversal, where h_i (called i's *height*) and p_i (called i's p-state) are integers.[4] The ordering $<$ on the tuples in each case is the lexicographical ordering.[5] For a node i, let C_i

[4]The heights (h_is) are initialized to either hop counts or distances from the destination (evaluated from either actual or virtual locations [21]), with the destination's height being zero. All p-states are initialized to 0.

[5]For tuples a, b of the same dimension, $a > b$ iff $a_i > b_i$ where i is the smallest index such that $a_i \neq b_i$. Thus even if the heights and p-states of two adjacent nodes are identical (e.g.,

denote the set of i's *neighbors*. Also, let $h = (h_1, \ldots, h_N)$ and $p = (p_1, \ldots, p_N)$. Then, the forwarding set of node i can be written as

$$F_i(h) = \{j \in C_i |\ (h_j, j) < (h_i, i)\}$$

for full reversal, and

$$F_i(p, h) = \{j \in C_i |\ (p_j, h_j, j) < (p_i, h_i, i)\}$$

for partial reversal. Clearly, node i is *stuck* if $F_i(h) = \emptyset$ (for full reversal), or $F_i(p, h) = \emptyset$ (for partial reversal). Node i, to determine if it is stuck, broadcasts its state. All its alive neighbors with lower states acknowledge. (Recall that a few of the neighbors might not be awake due to battery outage). If node i does not receive any acknowledgment until an a priori fixed timeout, it concludes that its state is the least among its neighbors, i.e., it is stuck.

The GB algorithms distributively update the states of stuck nodes so that a destination-oriented DAG is obtained. The algorithms are as follows.

Full link reversal. In this algorithm, a stuck node reverses the direction of all the incoming links. Node i updates its state as follows.

Algorithm 1 GB Full link reversal

1: **if** $F_i(h) = \emptyset$ **then**
2: $\quad h_i \leftarrow \max\{h_j |\ j \in C_i\} + 1$
3: **end if**

Remarks 2.1. *Evidently, a node i, if stuck, leapfrogs the heights of all its neighbors after an iteration of the above algorithm. All neighbors thereby enter the forwarding set of node i.*

Partial link reversal. In this algorithm, every node keeps a list of its neighbors that have already reversed their links to it. If a node is stuck, it reverses the

if both have equal hop counts from the destination), their distinct indices can be used to set the direction of the connecting link.

directions of links to all those neighbors that are not in the list, and empties the list. If all its neighbors are in the list, then it reverses the directions of all the incoming links, and empties the list. Node i updates its state as follows.

Algorithm 2 GB Partial link reversal

1: **if** $F_i(p,h) = \emptyset$ **then**
2: $\quad p_i \leftarrow \min\{p_j |\ j \in C_i\} + 1$
3: $\quad$ **if** there exists a $j \in C_i$ with $p_i = p_j$ **then**
4: $\quad\quad h_i \leftarrow \min\{h_j |\ j \in C_i$ with $p_i = p_j\} - 1$
5: $\quad$ **end if**
6: **end if**

Remarks 2.2. *The update rule (Line 2) ensures that for neighboring nodes p_is are always adjacent integers. For a stuck node i, the h_i update (Lines 3-4) ensures that, i does not reverse the links to the neighbors that have updated states since i's last update.*

Note that all the nodes run Algorithm 1 (or Algorithm 2 in case of partial link reversal) asynchronously, i.e., their reversals can follow any arbitrary timing and order. Gafni and Bertsekas [1] show the following properties.

Proposition 2.1. *(a) Starting from any state h (or (p,h) in case of Algorithm 2), Algorithms 1 and 2 terminate in a finite number of iterations yielding destination oriented DAGs.*

(b) Algorithm 1 results in the same destination-oriented DAG regardless of the timing and order of reversals. The same holds for Algorithm 2.

(c) Algorithms 1 and 2 are such that only those nodes that do not initially have a greedy path to the destination update their states at any stage.

Remarks 2.3. *The updates at a stuck node, in both Algorithms 1 and 2, depend on knowledge of states of neighbors (see Line 2 in Algorithm 1 and Lines 2, 3, 4 in Algorithm 2). After each link reversal, the updating node needs to broadcast*

its new state, and its neighbors need to gather this information in a reliable fashion (e.g., using an error detection scheme). In the following sections, we see how to avoid these exchanges, a desired level of ignorance that we call neighbor obliviousness.

3. Full Link Reversal

3.1. Neighbor Oblivious Full Link Reversal

The main idea may be summarized as follows. Suppose that the algorithm is such that a node, at any stage, knows the entire *range* of heights of its neighbors. Then it may execute a full reversal by raising its height to a value higher than the maximum in the range. Note that the updating node does not need to know the exact states of its neighbors, so valuable communication time and energy are saved.

Notation. The notation used is listed below for ease of reference.

- $[N] = \{1, 2, \ldots, N\}$ is the set of nodes (or node indices).
- $t_i \in \mathbb{Z}_+$ is the number of height updates made by node i; this is initialized to 0 for all i.
- $h_i(t_i) \in \mathbb{Z}_{++} = \mathbb{Z}_+ \backslash \{0\}$ is the height of node i after t_i updates; $h_i(0)$ refers to the initial height. The destination's height is 0.
- $a_i = (t_i, h_i(t_i), i)$ is the state of node i; t_i is referred to as its t-state.
- C_i is the set of neighbors of i, i.e., those with which i can directly communicate.
- $F_i(h) = \{j \in C_i |\ (h_j(t_j), j) < (h_i(t_i), i)\}$ is the forwarding set of node i, given the heights $h = (h_1(t_1), h_2(t_2), \ldots, h_N(t_N))$.
- $h_{\max} = \max\{h_1(0), \ldots, h_N(0)\}$.

The algorithm is simple. Node i updates its state a_i as follows.

Algorithm 3 Neighbor oblivious full link reversal

1: **if** $F_i(h) = \emptyset$ **then**
2: $\quad t_i \leftarrow t_i + 1$
3: $\quad h_i(t_i) \leftarrow h_i(t_i - 1) + h_{\max}$
4: **end if**

Remarks 3.1. *Node i, if stuck, updates its state such that the new height surpasses the heights of all its neighbors (see Line 2). This reverses all the incoming links, a fact that we will prove in Proposition 3.2.*

Node i broadcasts a *hello* packet to determine if it is stuck. The lack of feedback (silence) following a broadcast suffices to determine if $F_i(h)$ is empty or not. However, node i does not need to know the states of its neighbors to perform updates (see Lines 2, 3 in Algorithm 3). Other nodes also independently and asynchronously execute similar algorithms. All the nodes broadcast their new states whenever they update. Timing and order of state updates can be arbitrary.

We emphasise that the updating node broadcasts its new state merely to facilitate its neighbors to update the corresponding link directions. It could instead broadcast any message that serves this purpose. For instance a node could broadcast a bit string with as many bits as the number of its neighbours, with each bit indicating the link direction to an associated neighbour. The length of this bit string would not increase with time. Moreover, the neighbouring nodes do not need to store this broadcast message for use at a later stage. We now proceed to state and prove some of the properties of Algorithm 3).

Proposition 3.1. *(a) The height of a node i in t-state t_i is explicitly given by*

$$h_i(t_i) = h_i(0) + t_i h_{\max}.$$

(b) For any node i, and $t_i \in \mathbb{Z}_+$, we have $t_i h_{\max} < h_i(t_i) \leq (t_i + 1)h_{\max}$.

(c) For any two neighbors i and j, and $t_i, t_j \in \mathbb{Z}_+$ we have the following implication

$$t_i > t_j \Rightarrow h_i(t_i) > h_j(t_j).$$

(d) For any two neighbors i and j, at any stage of the algorithm, we have $0 \leq \mid t_i - t_j \mid \leq 1$.[6]

(e) For any node i, $t_i \leq N$ at any stage of the algorithm.

Proof 1. *(a) This follows immediately from the height update rule (Line 3 in Algorithm 3).*

(b) This follows from (a) *and* $0 < h_i(0) \leq h_{\max}$.

(c) The implication holds because $h_i(t_i) > t_i h_{\max}$ *and* $h_j(t_j) \leq (t_j+1)h_{\max}$ *(see* (b)*).*

(d) Without loss of generality, assume $t_i \geq t_j$. We claim that $t_i \leq t_j + 1$. We prove the claim via contradiction. Suppose $t_i > t_j + 1$. Node i must have reached this state through $t_j + 1$ because t_i is initialized to zero and is incremented by one each time node i updates its state. When node i's t-state was $t_j + 1$, from (c) *$h_i(t_j + 1) > h_j(t_j)$, and therefore it had an outgoing link to node j. Thus, i would not have updated its t-state to $t_j + 2$ or higher. This contradicts our supposition, and proves the claim.*

(e) Observe that any one hop neighbor of the destination never updates its heights; it always has an outgoing link to the destination. Consequently, for any such node, say node i, $t_i = 0$ at any stage of the algorithm. Now, assume that for a node j, $t_j > N$ at some stage. Then, there is pair of neighbors k and l such that $\mid t_k - t_l \mid > 2$. But this contradicts part (d)*. Thus, we have the bound $t_i \leq N$ for any node i.*

[6] The analogous property for the link-centric full reversal algorithm of Charron-Bost et al. [18] is asserted in [19, Propositions 2 and 3]. Our partial link reversal protocol also exhibits this property (see Proposition d). This is expected because of the fact that every execution of the partial link reversal on any directed routing graph corresponds to an execution of full link reversal on a transformed graph (see [20])

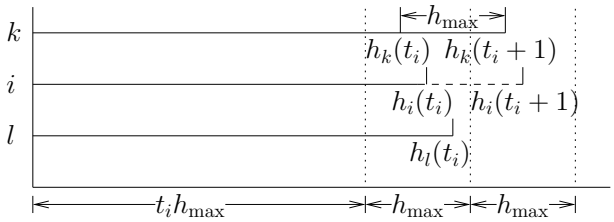

Figure 1: An illustration of Algorithm 3 at a stuck node i. Note that $t_l = t_i$ while $t_k = t_i + 1$. When node i updates its state, it reverse the links to both l and k.

Remarks 3.2. *1. For any node, the size of the state (i.e., the number of bits required to represent the state) grows with the number of state updates. However, Proposition 3.1*(e) *implies that, for any node, the number of updates is upper bounded by N, and hence the $t-$state size is upper bounded by* $\log(N)$*. Notice that heights are functions of t-states (Proposition 3.1*(a)*), and hence need not be stored separately.*

2. Proposition 3.1(c) *implies that the forwarding set of node i can be alternatively defined as*

$$F_i(a) = \{j \in C_i |\ a_j < a_i\},$$

where $a = (a_1, \ldots, a_N)$ are the nodes' states.

Proposition 3.2. *In Algorithm 3, a stuck node reverses the directions of all the incoming links.*

Proof 2. *Consider a stuck node i. For any node $j \in C_i$, $h_j(t_j) \geq h_i(t_i)$. So, by virtue of Propositions 3.1*(c)-(d)*, we have either $t_j = t_i$ or $t_j = t_i + 1$. See Figure 1 for an illustration.*

(i) Consider $t_j = t_i$. This is the case of node l in Figure 1. In this case, when node i makes an update, it moves to t-state $t_j + 1$. Hence the link is from i to j after the update.

(ii) Consider $t_j = t_i + 1$. This is the case of node k in Figure 1. In this case, observe that when node j updated its t-state from t_i to $t_j = t_i + 1$, node i's t-state must have been t_i. Further, it must have been the case that either $h_j(t_i) < h_i(t_i)$,

or $h_j(t_i) = h_i(t_i)$ *and* $j < i$. *Thus we have either* $h_j(t_j) < h_i(t_i) + h_{\max}$, *or* $h_j(t_j) = h_i(t_i) + h_{\max}$ *and* $j < i$. *Hence when node* i *makes an update, since* $h_i(t_i + 1) = h_i(t_i) + h_{\max}$, *the link is now from* i *to* j. *This concludes the proof.*

340 **Proposition 3.3.** *Algorithm 3 can be embedded within the GB algorithms framework. Thus it inherits the properties in Proposition 2.1.*

Proof 3. *For all* $i \in [N]$, *let* A_i *be the set of feasible states of node* i. *Notice that* $a_i = (t_i, h_i(t_i), i)$ *in our case. Define* $v = (a_1, a_2, \ldots, a_N)$. *Let* V *be the set of all such* N*-tuples. For each* $v \in V$, *let* $S(v) \subset [N]$ *denote the set of stuck nodes.*

$$S(v) = \{i \in [N] |\ a_j > a_i \text{ for all } j \in C_i\}.$$

We consider iterative algorithms of the form

$$v \leftarrow \overline{v} \in M(v),$$

where $M(\cdot)$ *is a point-to-set mapping;* $M(v) \subset V$ *for all* $v \in V$. *In the following we show that the proposed neighbor oblivious link reversal algorithms satisfy the assumptions of GB algorithms.*

(A.1). Define $g_i : V \to A_i$, $i = 1, \ldots, N$ *as*

$$g_i(v) = \begin{cases} (t_i + 1, h_i(t_i) + h_{\max}, i) & \text{if } i \in S(v), \\ (t_i, h_i(t_i), i) & \text{if } i \notin S(v). \end{cases}$$

345 *The set* $M(v)$ *is then given by*

$$M(v) = \begin{cases} \{v\} & \text{if } S(v) = \emptyset, \\ \{\overline{v} = (\overline{a}_1, \ldots, \overline{a}_N) |\ \overline{v} \neq v \text{ and either } \overline{a}_i = a_i \\ \quad \text{or } \overline{a}_i = g_i(v) \text{ for all } i \in [N]\} & \text{if } S(v) \neq \emptyset. \end{cases}$$

(A.2). From (A.1), it is clear that for each $v = (a_1, \ldots, a_N)$ *and* $i = 1, \ldots, N$, *the functions* $g_i(\cdot)$ *satisfy*

$$g_i(v) > a_i \text{ if } i \in S(v),$$
$$\text{and } g_i(v) = a_i \text{ if } i \notin S(v).$$

Furthermore, for each $i = 1, \ldots, N$, $g_i(v)$ *depends only on* a_i *and* $\{a_j |\ j \in C_i\}$; *the latter states determine if* $i \in S(v)$ *or otherwise.*

(A.3). Consider a node i and a sequence $\{v^k\} \subset V$ for which $i \in S(v^k)$ for an infinite number of indices k. If r is one of these indices, $g_i(v^r) - a_i^r \geq (1, h_{\max}, 0)$, otherwise $g_i(v^r) - a_i^r = 0$. Hence the sequence

$$\left\{ a_i^0 + \sum_{r=0}^{k} [g_i(v^r) - a_i^r] \right\}$$

is unbounded in A_i.

Gafni and Bertsekas [1] show that if the communication graph is connected and an algorithm satisfies Assumptions (A.1)-(A.3), then Proposition 2.1 holds for the algorithm. This concludes the proof of the proposition.

3.2. Two Bits Full Link Reversal

In practice, states are stored using finite bit-width representations. While the size of the states can depend on the number of nodes in the network, it should not grow with the number of iterations of the algorithm. The t-states which are the counts of the number of reversals, though bounded (see Proposition 3.1(e)), grow as the algorithm runs. There could be 1000s of nodes in the network, and in resource limited nodes in wireless sensor networks, memory is also at a premium. Therefore, GB and NOLR algorithms need to be modified for implementation in practical systems.

We now give a modification of Algorithm 3 that uses only two bits for the t-state and does not update heights. To do this we exploit the fact that, for any two neighbors i and j, the link direction is entirely governed by $t_i, t_j, h_i(0)$ and $h_j(0)$. More precisely, the link is directed from i to j if and only if either $t_i > t_j$, or $t_i = t_j$ and $(h_i(0), i) > (h_j(0), j)$. Thus t-states along with the initial heights suffice to determine link orientations. Moreover, since at any stage t_i and t_j are either the same or adjacent integers (Proposition 3.1(d)), we need only two bits to describe their order. Specifically, if we define, for all i,

$$\tau_i = t_i \mod 4,$$

and a cyclic ordering

$$00 < 01 < 10 < 11 < 00$$

on candidate values of τ_i, we obtain

$$t_i > t_j \Longleftrightarrow \tau_i > \tau_j.$$

For node i, τ_i is referred to as its τ-state. Following the above discussion, we can redefine the forwarding set of node i as

$$F_i(\tau) = \{j \in C_i |\ \tau_j < \tau_i \text{ or } (\tau_j = \tau_i \text{ and } (h_j(0), j) < (h_i(0), i))\},$$

where $\tau = (\tau_1, \ldots, \tau_N)$. In the two bit full link reversal algorithm node i updates its state as follows.

Algorithm 4 Two bit full link reversal

1: **if** $F_i(\tau) = \emptyset$ **then**
2: $\quad \tau_i \leftarrow (\tau_i + 1) \mod 4$
3: **end if**

In Figure 2, we show the progression of Algorithm 4 in a sample network. Following are the key properties of this algorithm.

Proposition 3.4. *(a) In Algorithm 4, a stuck node reverses the directions of all the incoming links.*

(b) Algorithm 4 exhibits the properties in Proposition 2.1.

Proof 4. *(a) Consider a stuck node i. Following Proposition 3.1*(d) *and the definition of τ-states, for any node $j \in C_i$, we have either $\tau_j = \tau_i$ or $\tau_j = (\tau_i + 1)$* mod 4*.*

(i) Consider $\tau_j = \tau_i$. In this case, when node i makes an update, it moves to τ-state $(\tau_j + 1) \mod 4$ which is greater than τ_j. Hence the link is from i to j after the update.

(ii) Consider $\tau_j = (\tau_i + 1) \mod 4$. In this case it must be that $(h_j(0), j) < (h_i(0), i)$; were it not the case, node j at τ-state τ_i would not have done an update. Thus when node i updates its τ-state to $(\tau_i + 1) \mod 4 = \tau_j$, the link is now from i to j. This concludes the proof of part (a).

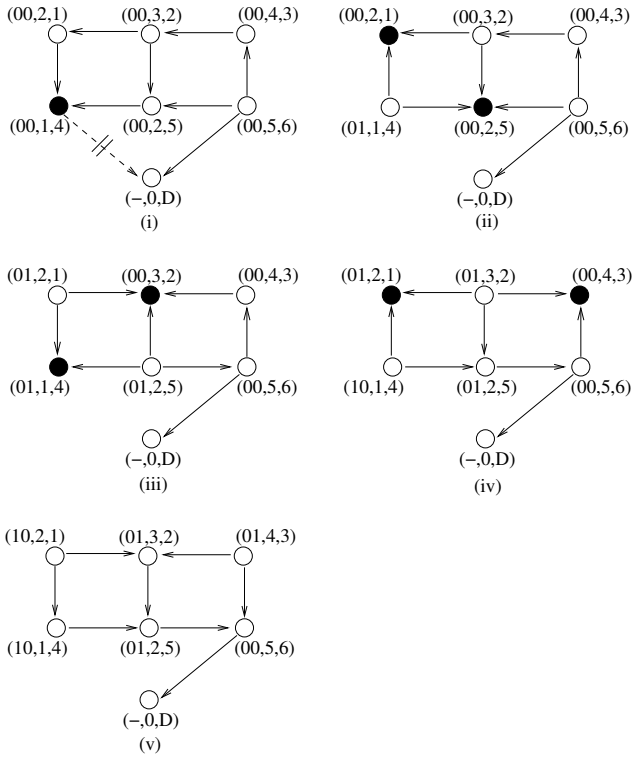

Figure 2: A sample execution of the two-bit full link reversal algorithm. We show the tuples $(\tau_i, h_i(0), i)$ for each node i at each stage; the node D with height 0 is the destination. At each stage, the solid circles depict the stuck nodes. The cut link in (i) results in an initial destination disoriented network with node 4 starting out from a stuck state.

(b) Let all the nodes in the network run Algorithm 4. Also consider another copy of the network (with the same initial link orientations) where all the nodes execute Algorithm 3 as follows. The same node as in the original network does the first update. Then we are left with the same set of stuck nodes as in the original network because updates lead to full link reversals in both. The next update is also by the same node as in the original network, thus again yielding the same set of stuck nodes. Likewise, subsequent updates also follow the same timing and order as in the original network. Since the nodes' updates in the latter network satisfy the properties in Proposition 2.1, so do the updates in the original network.

3.3. One Bit Full Link Reversal

Recall that in full reversal, a stuck node reverses the directions of all its incoming links. Algorithm 4 executes this using initial heights and a two bit state. We now describe a simpler way to achieve this using initial heights and a single flag bit at each node. More precisely, with each node i, we associate a binary state δ_i that is initialized to zero. For any two neighbors i and j with $(h_i(0), i) > (h_j(0), j)$, the corresponding link is directed from i to j if $\delta_i = \delta_j$, and from j to i if $\delta_i \neq \delta_j$. In other words, at any stage, the forwarding set of node i is

$$F_i(\delta) = \{j \in C_i |\ ((h_j(0), j) < (h_i(0), i) \text{ and } \delta_j = \delta_i) \text{ or } ((h_j(0), j) > (h_i(0), i) \text{ and } \delta_j \neq \delta_i)\},$$

where $\delta = (\delta_1, \ldots, \delta_N)$.

We propose the following one bit full link reversal algorithm. Node i updates its states as follows.

Algorithm 5 One bit full link reversal

1: **if** $F_i(\delta) = \emptyset$ **then**
2: $\quad \delta_i \leftarrow (\delta_i + 1) \mod 2$
3: **end if**

Remarks 3.3. *For stuck node i, the updated δ-state is same as the δ-states of neighbors with higher heights but complements the δ-states of neighbors with lower heights. Thus, all its links become outgoing.*

Algorithm 5 has similar properties as Algorithm 4.

Proposition 3.5. *(a) In Algorithm 5, a stuck node reverses the directions of all the incoming links.*

(b) Algorithm 5 exhibits the properties in Proposition 2.1.

Proof 5. *(a) Consider a stuck node i and an arbitrary node $j \in C_i$. Then, either $(h_i(0), i) < (h_j(0), j)$ and $\delta_i = \delta_j$, or $(h_i(0), i) > (h_j(0), j)$ and $\delta_i \neq \delta_j$. In either case, when node i flips δ_i, the link between i and j is reversed.*

(b) The proof is identical to that of Proposition 3.4(b).

4. Partial Link Reversal

Recall that the link reversals are intended to yield a destination oriented DAG. However, link reversals are accompanied by state updates and information exchanges, and can potentially lead to more nodes being stuck. Thus, a stuck node could execute a partial link reversal (i.e., need not reverse all its incoming links) so that the link graph converges quickly to a destination oriented graph. We focus on the partial link reversal scheme proposed by Gafni and Bertsekas [1] (see Algorithm 2).

4.1. Neighbor Oblivious Partial Link Reversal

As in neighbor oblivious full link reversal, the algorithm is such that a node, at any stage, knows the entire *range* of all neighbors' heights but not the exact values. Then, the node raises its height to an appropriate value to effect only a partial link reversal. Again, as in Section 3, the updating node does not need to know the exact states of its neighbors, so valuable communication time and energy are saved.

Notation. The new notation is collected below.

- $a_i = (t_i, h_i(t_i), (-1)^{t_i} i)$ is the state of node i; t_i is referred to as its t-state.
- $F_i(h) = \{j \in C_i |\ (h_j(t_j), (-1)^{t_j} j) < (h_i(t_i), (-1)^{t_i} i)\}$ is the forwarding set of node i for heights $h = (h_1(t_1), \ldots, h_N(t_N))$.
- $\{z(0), z(1), \ldots\}$ is a sequence satisfying

$$z(t) = \begin{cases} 0 & \text{if } t = 0, \\ 2^{t-1}(2h_{\max} + 1) & \text{if } t \geq 1. \end{cases}$$

In the neighbor oblivious partial link reversal algorithm node i updates its state as follows.

Algorithm 6 Neighbor oblivious partial link reversal

1: **if** $F_i(h) = \emptyset$ **then**
2: $\quad t_i \leftarrow t_i + 1$
3: $\quad h_i(t_i) \leftarrow z(t_i) - h_i(t_i - 1)$
4: **end if**

Remarks 4.1. *Assume that node i is stuck. The height update (Line 3) along with the definition of sequence $\{z(0), z(1), \dots\}$ ensure that i's updated height surpasses the heights of those neighbors that have not updated states since i's last update, but still falls short of the heights of other neighbors. A similar behavior is ensured by the third components of the states (e.g., $(-1)^{t_i} i$ in a_i) when two neighbors have identical initial heights.*

As discussed before, node i broadcasts a *hello* packet to determine if it is stuck. However, it does not need to know the states of its neighbors to perform updates (see Lines 2, 3 in Algorithm 6). Also, whenever it updates its state, it broadcasts its new state to facilitate its neighbors updating the corresponding link directions. Other nodes also independently and asynchronously execute similar algorithms. In particular, multiple nodes can update at the same time. The following properties of this algorithm are similar to those of Algorithm 3.

Proposition 4.1. *(a) The height of a node i is explicitly given by*

$$h_i(t_i) = \begin{cases} \displaystyle\sum_{l=1}^{t_i/2} z(2l-1) + h_i(0) & \text{if } t_i \text{ is even,} \\ \displaystyle z(1) + \sum_{l=1}^{(t_i-1)/2} z(2l) - h_i(0) & \text{if } t_i \text{ is odd.} \end{cases}$$

(b) For any node i, and $t_i \in \mathbb{Z}_{++}$, we have $z(t_i - 1) < h_i(t_i) < z(t_i)$.

(c) For any two neighbors i and j, and $t_i, t_j \in \mathbb{Z}_+$ we have the following implication

$$t_i > t_j \Rightarrow h_i(t_i) > h_j(t_j).$$

(d) For any two neighbors i and j, at any stage of the algorithm, we have $0 \leq \mid t_i - t_j \mid \leq 1$.

(e) For any node i, $t_i \leq N$ at any stage of the algorithm.

Proof 6. *(a) We first obtain a recursion on $h_i(t_i)$ using the height update rule (Line 3 in Algorithm 6). For any $t_i \geq 2$,*

$$\begin{aligned} h_i(t_i) &= z(t_i) - h_i(t_i - 1) \\ &= 2z(t_i - 1) - (z(t_i - 1) - h_i(t_i - 2)) \\ &= z(t_i - 1) + h_i(t_i - 2). \end{aligned}$$

Successive applications of this recursion leads to expression for the case when t_i is even. If we also use that $h_i(1) = z(1) - h_i(0)$, we get the expression for the case when t_i is odd.

(b) We prove the inequalities by induction on t_i. For $t_i = 1$,

$$0 < h_i(1) < z(1).$$

Now, assume that $0 < h_i(t_i) < z(t_i)$ for some $t_i \in \mathbb{Z}_{++}$. From the height update rule (Line 3 in Algorithm 6),

$$\begin{aligned} h_i(t_i + 1) &= z(t_i + 1) - h_i(t_i) \\ &= 2z(t_i) - h_i(t_i) \\ &> z(t_i), \end{aligned}$$

where the inequality holds because $h_i(t_i) < z(t_i)$. Also, $0 < h_i(t_i)$ implies that $h_i(t_i + 1) < z(t_i + 1)$. This completes the induction, and shows that the inequalities hold for all $t_i \in \mathbb{Z}_{++}$.

(c) The implication holds because $h_j(t_j) < z(t_j)$, $h_i(t_i) > z(t_i - 1)$ and $z(t)$ is increasing in t.

(d) The proof is identical to that of Proposition 3.1(d).

(e) The proof is identical to that of Proposition 3.1(e).

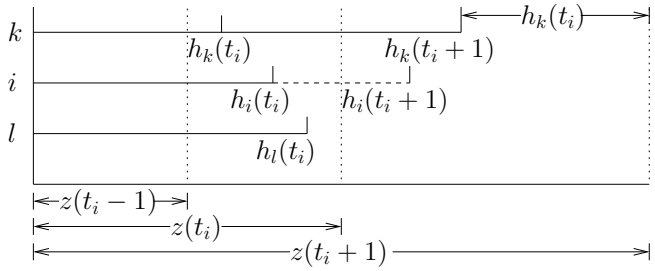

Figure 3: An illustration of Algorithm 6 at a stuck node i. Note that $t_l = t_i$ while $t_k = t_i + 1$. Node k has reversed its link to i after i's last update but node l has not. When node i updates its state, it reverse the link to l but not the one to k.

Remarks 4.2. *1. As in the case of Algorithm 3, for any node, the number of state updates is upper bounded by N, and hence the state size is upper bounded by* $\log(N)$.

2. Propositions 4.1(c) *implies that the forwarding set of node i can be alternatively defined as*

$$F_i(a) = \{j \in C_i |\ a_j < a_i\},$$

where $a = (a_1, \ldots, a_N)$ are the nodes' states.

Proposition 4.2. *In Algorithm 6, a stuck node i reverses the directions of only those of its links that have not been reversed since i's last update. If every link to node i has been reversed after i's last update, it performs two successive updates to reverse the directions of all its links.*

Proof 7. *Since node i is stuck, for any node $j \in C_i$,*

$$(h_j(t_j), (-1)^{t_j} j) > (h_i(t_i), (-1)^{t_i} i).$$

By virtue of Propositions 4.1(c)-(d)*, we also have either $t_j = t_i$ or $t_j = t_i + 1$. See Figure 3 for an illustration.*

(i) Consider $t_j = t_i$. This is the case of node l in Figure 3. We claim that node j has not reversed its link to i since i's last update. If $t_i = 0$, this claim is trivially valid. If $t_i \geq 1$, we will show that the progression of updates when

both nodes' t-states were $t_i - 1$ was: node j updated, then node i updated. As a consequence, again, our claim will be valid. To see the progression of updates, observe that if $h_j(t_j) = h_i(t_i)$, then $(-1)^{t_j} j > (-1)^{t_i} i$. Thus, by sign flipping, at t-states $t_i - 1 = t_j - 1$, $(-1)^{t_j - 1} j < (-1)^{t_i - 1} i$. Also, by the form of the updates at $t_i - 1$, $h_j(t_j - 1) = h_i(t_i - 1)$. So the link was from node i to node j and it must be j that updated first. On the other hand, if $h_j(t_j) > h_i(t_i)$, then

$$\begin{aligned} h_j(t_j - 1) &= z(t_j) - h_j(t_j) \\ &< z(t_i) - h_i(t_i) \\ &= h_i(t_i - 1). \end{aligned}$$

Again we conclude that the link was from i to j, and it must be j that updated first. This establishes the claimed progression of states.

Continuing with the case, when node i now makes an update, it moves to t-state $t_j + 1$. Hence the link is from i to j after the update.

(ii) Consider $t_j = t_i + 1$. This is the case of node k in Figure 3. We claim that node j has reversed its link to i after i's last update. Were it not the case, node i's t-state immediately prior to its last update would have been $t_i - 1 = t_j - 2$ which contradicts Proposition 4.1(d).

Moreover, when node j's t-state was $t_j - 1 = t_i$, it must have been the case that

$$(h_j(t_j - 1), (-1)^{t_j - 1} j) < (h_i(t_i), (-1)^{t_i} i).$$

If $h_j(t_j - 1) = h_i(t_i)$, then $(-1)^{t_j - 1} j < (-1)^{t_i} i$. Thus, by sign flipping, at t-states $t_i + 1 = t_j$, $(-1)^{t_j} j > (-1)^{t_i + 1} i$. Also, $h_j(t_j) = h_i(t_i + 1)$. So, even after node i makes an updates and moves to t-state $t_i + 1$, the link continues to be from j to i. If $h_j(t_j - 1) < h_i(t_i)$, then

$$\begin{aligned} h_j(t_j) &= z(t_j) - h_j(t_j - 1) \\ &> z(t_i + 1) - h_i(t_i) \\ &= h_i(t_i + 1). \end{aligned}$$

Again, even after node i makes an updates and moves to t-state $t_i + 1$, the link continues to be from j to i. This proves the first part of the proposition.

Finally, suppose that every neighbor of node i has reversed its link to i after i's last update. Then, as shown above, $t_j = t_i + 1$ for all $j \in C_i$. Again as argued above, if node i updates its state, it does not reverse any of its links. Thus it performs one more update. After this update its t-state is $t_i + 2$ which exceeds t_j for all $j \in C_i$. So all its links are reversed.

Remarks 4.3. *For a stuck node, if all its neighbors have reversed the corresponding links after its last update, it takes two iteration to reverse all the incoming links. This is unlike Algorithm 2 which needs only one iteration.*

Proposition 4.3. *Algorithm 6 can be embedded within the GB algorithms framework. Thus it inherits the properties in Proposition 2.1.*

Proof 8. *We use the same proof technique as used for Proposition 3.3. Notice that $a_i = (t_i, h_i(t_i), (-1)^{t_i} i)$ in this case. We define $g_i : V \to A_i$ as*

$$g_i(v) = \begin{cases} (t_i + 1, z(t_i + 1) - h_i(t_i), (-1)^{t_i+1} i) & \\ & \text{if } i \in S(v), \\ (t_i, h_i(t_i), (-1)^{t_i} i) & \text{if } i \notin S(v). \end{cases}$$

Again, it is easy to check that Assumptions (A.1)-(A.3) as in the proof of Proposition 3.3 hold. Thus, following the same argument as in the proof of Proposition 3.3, this proposition also holds.

4.2. Two-Bit Partial Link Reversal

In Algorithm 6, nodes' t-states grow as they update. We now give a modification of Algorithm 6 that uses only two bits for t-state and does not update heights. To do this we exploit the fact that for any two neighbors i and j, the link direction is entirely governed by $t_i, t_j, h_i(0)$ and $h_j(0)$. More precisely, the link is directed from i to j if and only if either $t_i > t_j$, or $t_i = t_j$ and $(-1)^{t_i}(h_i(0), i) > (-1)^{t_j}(h_j(0), j)$. Thus t-states along with the initial heights suffice to determine link orientations. Moreover, since at any stage t_i and t_j are either same or adjacent integers (Proposition 4.1(d)), we need only two bits

to describe their order. Specifically, if we define τ-states for all the nodes as in Section 3.2, we obtain

$$t_i > t_j \Longleftrightarrow \tau_i > \tau_j.$$

As before, for node i, τ_i is referred to as its τ-state. Following the above discussion, we can redefine the forwarding set of node i as

$$F_i(\tau) = \{j \in C_i | \tau_j < \tau_i \text{ or } (\tau_j = \tau_i \text{ and } (-1)^{\tau_i}(h_i(0), i) > (-1)^{\tau_j}(h_j(0), j))\},$$

where $\tau = (\tau_1, \ldots, \tau_N)$. We are thus led to the following two bit version of the partial link reversal algorithm. Node i updates its states as follows.

Algorithm 7 Two bit partial link reversal

1: **if** $F_i(\tau) = \emptyset$ **then**
2: $\quad \tau_i \leftarrow (\tau_i + 1) \mod 4$
3: **end if**

In Figure 4, we illustrate the progression of Algorithm 7 in the same sample network as in Figure 2.

Following are the key properties of this algorithm.

Proposition 4.4. *(a) In Algorithm 7, a stuck node i reverses the directions of only those of its links that have not been reversed since i's last update. If every link to node i has been reversed after i's last update, it performs two successive updates to reverse the directions of all its links.*

(b) Algorithm 7 exhibits the properties in Proposition 2.1.

Proof 9. *(a) Following Proposition 4.1*(d) *and the definition of τ-states, for any node $j \in C_i$, we have either $\tau_j = \tau_i$ or $\tau_j = (\tau_i + 1) \mod 4$.*
(i) Consider $\tau_j = \tau_i$. We claim that node j has not reversed its link to i since i's last update. If neither i nor j has ever made an update, this claim is trivially valid. If both of them have made updates, by Proposition 4.1(d)*, it cannot be that one of them made two updates without the other updating. So both must have*

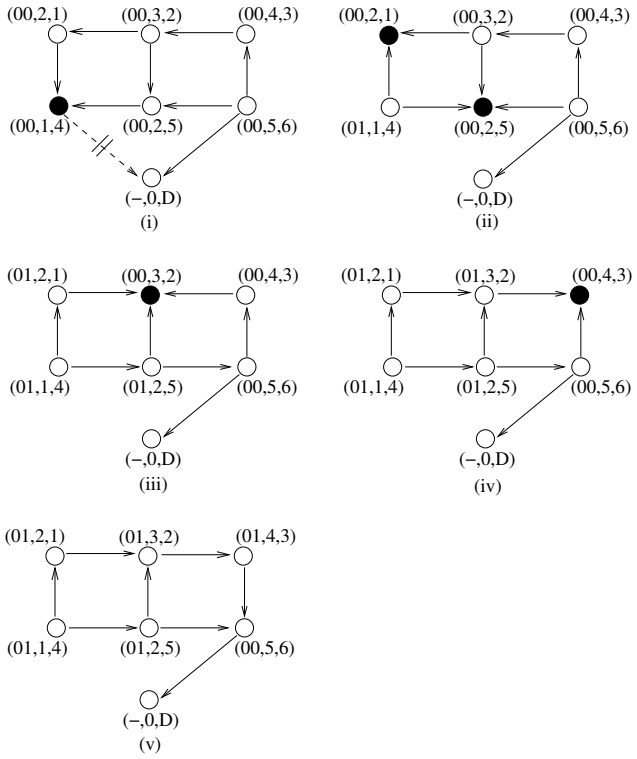

Figure 4: A sample execution of the two-bit partial link reversal algorithm. We show the tuples $(\tau_i, h_i(0), i)$ for each node i at each stage; the node D with height 0 is the destination. At each stage, the solid circles depict the stuck nodes. The cut link in (i) results in an initial destination disoriented network with node 4 starting out from a stuck state.

been at $(\tau_i - 1) \mod 4$ *at some point of time. We will show that the progression of updates when both nodes'* τ*-states were* $\tau_i - 1 \mod 4$ *was: node* j *updated, then node* i *updated. As a consequence, again, our claim is valid. To see the progression of updates, observe that*

$$(-1)^{\tau_j}(h_j(0), j) > (-1)^{\tau_i}(h_i(0), i).$$

Thus, by sign flipping, at the nodes' immediately prior τ*-states, the inequality was in reverse direction. So the link was from node* i *to node* j *and it must be* j *that updated first.*

Continuing with the case, when node i *makes an update, it moves to* τ*-state*

$(\tau_j + 1) \mod 4$. *Hence the link is from* i *to* j *after the update.*

(ii) Consider $\tau_j = (\tau_i + 1) \mod 4$. *We claim that node* j *has reversed its link to* i *after* i*'s last update. Were it not the case, node* i*'s* τ*-state immediately prior to its last update would have been* $(\tau_i - 1) \mod 4 = (\tau_j - 2) \mod 4$ *which contradicts the fact that at any stage* τ_i *and* τ_j *assume either same or adjacent values.*

Moreover, when node j*'s* τ*-state was* $(\tau_j - 1) \mod 4 = \tau_i$, *it must have been the case that*

$$(-1)^{\tau_i}(h_j(0), j) < (-1)^{\tau_i}(h_i(0), i).$$

Thus, by sign flipping, at τ*-states* $(\tau_i + 1) \mod 4 = \tau_j$,

$$(-1)^{\tau_j}(h_j(0), j) > (-1)^{\tau_j}(h_i(0), i).$$

So, even after node i *makes an update and moves to* τ*-state* $(\tau_i + 1) \mod 4$, *the link continues to be from* j *to* i.

Finally, suppose that every neighbor of node i *has reversed its link to* i *after* i*'s last update. Then, by the arguments above,* $\tau_j = (\tau_i + 1) \mod 4$ *for all* $j \in C_i$. *Also, if node* i *updates its state once, it does not reverse any of its links, i.e., it is still stuck. Thus it performs one more update. After this update its* τ*-state is* $(\tau_i + 2) \mod 4$ *which exceeds* τ_j *for all* $j \in C_i$. *So all its links are reversed.*

(b) The proof is identical to that of Proposition 3.4(b).

5. Complexity of the Proposed Algorithms

As indicated earlier, the evolutions of the routing graph under our full and partial reversal algorithms are identical to the evolutions under GB algorithms [1] (or those under the schemes of Charron-Bost et al. [18]). Thus the complexity results of [17, 18, 19, 20] apply to our algorithms as well. In this section, we review a few of these results. These results also enable us to estimate the savings in computations, communication, and storage overheads due to neighbor obliviousness and finite-state properties of our algorithms. We start by defining the notions of work and time complexities.

Work Complexity. Recall that a link reversal is the action of a stuck node reversing some or all of its adjacent links. Given a routing graph, the *work complexity* of an algorithm is defined to be the total number of reversals performed by all the nodes until a destination oriented DAG is obtained [17]. This is referred to as the *global work complexity* in [18]) This is a measure of the communication and computation resources consumed by the algorithm before reaching an equilibrium.

Time Complexity. Let us consider a slotted network in which link reversals take place only at slot boundaries, and all the nodes that are stuck perform link reversals at the next boundary. Given a routing graph, the *time complexity* of an algorithm is defined to be the number of slots needed until a destination oriented DAG is obtained [17, 19]. This is a measure of the speed of the algorithm.

For instance, in the sample execution of two bits full link reversal algorithm in Figure 2, seven link reversals are needed until a destination oriented DAG is obtained. If we assume that time is slotted and link reversals take place only at the slot boundaries, one would require five slots until the equilibrium is reached.

Busch and Tirthapura [17] and Charron-Bost et al. [18, 19, 20] proved that work and time complexities of full and partial link reversal algorithms are $O(N^2)$, and demonstrated networks in which both these algorithms exhibit $\Theta(N^2)$ work and time complexities.[7]

Since the evolutions of the routing graph under our full and partial reversal algorithms are identical to the evolutions under the GB

[7] To be precise, Busch and Tirthapura [17] established that the work and time complexities of GB partial link reversal algorithm are $O(PN + N^2)$ where P is the difference between the maximum and minimum p-states of the nodes in the initial routing graph. They also demonstrated networks in which GB partial link reversal algorithm exhibits $\Theta(PN + N^2)$ work and time complexities. They further illustrated that P can be arbitrarily larger than N in mobile ad hoc networks where the underlying connection topology keeps changing (in particular, Proposition 4.1(d) no longer holds; see also the discussion in Section 6). However, neither Charron-Bost et al. [18] nor we have considered mobile networks.

algorithms, the time complexities of our algorithms are identical to those of the GB algorithms. In fact, the $O(N^2)$ bound can be directly inferred from our Propositions 3.1(e) and 4.1(e) which assert that all N nodes need at most N link reversals.

For work complexities, we recall the two potential implementations of the GB protocols (see Section 1.1). In the first scheme, nodes hold their neighbors' full state information which are of size $O(\log N)$ bits. Thus, in the worst case, each node requires $O(N \log N)$ bits of storage. In the second scheme, nodes do not maintain the states of their neighbors. But when a hello packet is broadcast by a node, all its neighbors respond with their full states. This scheme requires at least $\Omega(N^2)$ transmissions (of full states) until an equilibrium is reached (at least one "state information" transmission per link reversal). In our neighbor oblivious and finite state algorithms, since a stuck node's update is not a function of the full states of its neighbors, we economize on these storage or communication overheads.

Also, in GB algorithms, each stuck node uses its neighbors' state information to come up with its new state. The simpler update rules in our finite-state algorithms ensure that we can further save $\Omega(N^2)$ computations. The overall complexity may still be $\Omega(N^2)$, but with a smaller constant.

The link-centric schemes of Charron-Bost et al. [18] are also lightweight in the sense that the link reversal rules are simple. But in those schemes, the nodes must hold the labels of all incident links, and after each reversal, must communicate the new link labels and link directions to the corresponding adjacent nodes (neighbors). This requires $O(N)$ bits of storage and significant communication overhead at each node in the worst case. On the other hand, we require each node to store only $\log N$ bits for the node index and an additional two bits to execute our proposed finite-state link reversals. See Algorithms 4 and 7. Observe that links' directions should be set to ensure acyclicity of the routing graph in the beginning and after every execution of link reversal. Associating nodes with distinct and ordered indices (of $\log N$ size) provides a convenient method to accomplish this. Charron-Bost et al. [18], on the other hand, bypass

this issue by explicitly assuming that the initial routing graph is acyclic (see [18, Corollary 3.8]). Moreover, the authors in [18] rely on an oracle for initial link direction identification and actual link reversal.

6. Conclusion

We proposed neighbor oblivious link reversal (NOLR) schemes to get a destination oriented network out of the local minimum condition in geographic routing. Our algorithms fall within the general class of GB algorithms [1]. We then argued that both the algorithms, GB and NOLR, may suffer the problem of state storage overflow. This led us to modify the NOLR algorithms to obtain one bit full link reversal and two bit partial link reversal algorithms. The finite state algorithms inherit all the properties of NOLR algorithms which in turn inherit the properties of GB algorithms, and are pragmatic link reversal solutions to convert a destination-disoriented DAG to a destination-oriented DAG. The communication is lightweight since only broadcasts (hello packets and new state advertisements) contain state information (acknowledgements need not), and further, acknowledgements are sent only by the neighbors that have lower states than the querying node. We have given order estimates of the resulting savings in computations, communication and storage overheads.

The property $|t_i - t_j| \leq 1$ at every stage for all pairs of neighboring nodes is crucial for getting the finite state version of our NOLR algorithms. If addition of new nodes or links to the existing graph is allowed, this property could be violated. If full t-states (instead of only τ-states) are maintained, then since Algorithms 3 and 6 belong to the class of GB algorithms, they continue to exhibit the properties in Proposition 2.1. However, Algorithm 3 does not execute a full link reversal, and similarly, Algorithm 6 does not execute a partial link reversal. Furthermore, the finite state algorithms are not robust to addition of new nodes or links because the newly added nodes may not be able to take up a state consistent with the above property, or the DAG may be burdened by cycles.

Acknowledgments

This work was supported in part by a research grant on Wireless Sensor Networks for Intrusion Detection from DRDO, Government of India, and in part by the Indo-French Centre for Promotion of Advanced Research (IFCPAR), Project No. 4000-IT-A.

References

[1] E. M. Gafni, D. P. Bertsekas, Distributed algorithms for generating loop-free routes in networks with frequently changing topology, IEEE Transactions on Communications 29 (1) (1981) 11–18.

[2] B. Karp, H. T. Kung, GPSR: Greedy perimeter stateless routing for wireless networks, in: Proceedings of the 6th annual international conference on Mobile Computing and Networking (MobiCom), Boston, MA, USA, 2000, pp. 243–254.

[3] Q. Fang, J. Gao, L. J. Guibas, Locating and bypassing holes in sensor networks, Mobile Networks and Applications 11 (2) (2006) 187–200.

[4] E. Kranakis, H. Singh, J. Urrutia, Compass routing on geometric networks, in: Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG), Charlottetown, Canada, 1999, pp. 51–54.

[5] H. Kalosha, A. Nayak, S. Ruhrup, I. Stojmenovic, Select-and-protest-based beaconless georouting with guaranteed delivery in wireless sensor networks, in: Proceedings of the 27th IEEE Conference on Computer Communications (INFOCOM), Phoenix, AZ, 2008, pp. 346–350.

[6] C.-Y. Chang, K.-P. Shih, S.-C. Lee, S.-W. Chang, RGP: Active route guiding protocol for wireless sensor networks with obstacles, in: Proceedings of the IEEE International Conference on mobile adhoc and sensor Systems (MASS), Vancouver, BC, 2006, pp. 367–376.

[7] F. Yu, S. Park, Y. Tian, M. Jin, S.-H. Kim, Efficient hole detour scheme for geographic routing in wireless sensor networks, in: Proceedings of the IEEE Vehicular Technology Conference (VTC Spring), Marina Bay, Singapore, 2008, pp. 153–157.

[8] B. Leong, B. Liskov, R. Morris, Geographic routing without planarization, in: Proceedings of the 3rd conference on Networked Systems Design & Implementation (NSDI), San Jose, CA, 2006, pp. 25–25.

[9] P. Casari, M. Nati, C. Petrioli, M. Zorzi, ALBA: An adaptive load - balanced algorithm for geographic forwarding in wireless sensor networks, in: Proceedings of the IEEE Military Communications Conference (MILCOM), Washington, DC, USA, 2006, pp. 1–9.

[10] C. Perkins, P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, ACM SIGCOMM Computer Communication Review 24 (4) (1994) 234–244.

[11] D. Johnson, D. Maltz, J. Broch, et al., DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks, Ad hoc networking 5 (2001) 139–172.

[12] N. Arad, Y. Shavitt, Minimizing recovery state in geographic ad hoc routing, IEEE Transactions on Mobile Computing, 8 (2) (2009) 203–217.

[13] M. Corson, A. Ephremides, A distributed routing algorithm for mobile wireless networks, Wireless Networks 1 (1) (1995) 61–81.

[14] V. Park, M. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, in: Proceedings of IEEE Conference on Computer Communications (INFOCOM), Kobe, Japan, 1997, pp. 1405–1413.

[15] N. Malpani, J. Welch, N. Vaidya, Leader election algorithms for mobile ad hoc networks, in: Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications (DIALM), Boston, MA, USA, 2000, pp. 96–103.

[16] R. Ingram, P. Shields, J. Walter, J. Welch, An asynchronous leader election algorithm for dynamic networks, in: Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS), Rome, Italy, 2009, pp. 1–12.

[17] C. Busch, S. Tirthapura, Analysis of link reversal routing algorithms, SIAM Journal on Computing 35 (2) (2005) 305–326.

[18] B. Charron-Bost, A. Gaillard, J. L. Welch, J. Widder, Routing without ordering, in: Proceedings of the 21st ACM symposium on Parallel algorithms and architectures (SPAA), Calgary, Canada, 2009, pp. 145–153.

[19] B. Charron-Bost, M. Fugger, J. L. Welch, J. Widder, Full reversal routing as a linear dynamical system, in: Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Gdansk, Poland, 2011, pp. 101–112.

[20] B. Charron-Bost, M. Fugger, J. L. Welch, J. Widder, Partial is full, in: Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Gdansk, Poland, 2011, pp. 113–124.

[21] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, I. Stoica, Geographic routing without location information, in: Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom), San Diego, CA, USA, 2003, pp. 96–108.