Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks

Costas Busch[*] Srikanth Surapaneni[†] Srikanta Tirthapura[‡]

July 23, 2003

Abstract

Link reversal algorithms provide a simple mechanism for routing in mobile ad hoc networks. These algorithms maintain routes to any particular destination in the network, even when the network topology changes frequently. In link reversal, a node reverses its incident links whenever it loses routes to the destination. Link reversal algorithms have been studied experimentally and have been used in practical routing algorithms, including TORA [8].

This paper presents the first formal performance analysis of link reversal algorithms. We study these algorithms in terms of *work* (number of node reversals) and the *time* needed until the network stabilizes to a state in which all the routes are reestablished. We focus on the full reversal algorithm and the partial reversal algorithm, both due to Gafni and Berstekas [5]; the first algorithm is simpler, while the latter has been found to be more efficient for typical cases. Our results are as follows:

(1) The full reversal algorithm requires $O(n^2)$ work and time, where n is the number of nodes which have lost the routes to the destination.

(2) The partial reversal algorithm requires $O(n \cdot a^* + n^2)$ work and time, where a^* is a non-negative integer which depends on the state of the network. This bound is tight in the worst case, for any a^*.

(3) There are networks such that for *every* deterministic link reversal algorithm, there are initial states which require requires $\Omega(n^2)$ work and time to stabilize. Therefore, surprisingly, the full reversal algorithm is asymptotically optimal in the worst case, while the partial reversal algorithm is not, since a^* can grow arbitrarily large.

1 Introduction

A mobile ad hoc network is a temporary interconnection network of mobile wireless nodes without a fixed infrastructure. The attractive feature of such a network is the ease with which one can construct it: there is no physical set up needed at all. If mobile nodes come within the wireless range of each other, then they will be able to communicate. More significantly, even if two mobile nodes aren't within the wireless range of each other, they might still be able to communicate through a multi-hop path. The lack of a fixed infrastructure makes routing between nodes a hard problem. Since nodes are moving, the underlying communication graph is changing, and the nodes have to adapt quickly to such changes and reestablish their routes.

[*]Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180; buschc@cs.rpi.edu

[†]Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180; suraps@cs.rpi.edu

[‡]Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010; snt@iastate.edu

Link reversal routing algorithms [9, Chapter 8] are adaptive, self-stabilizing, distributed algorithms used for routing in mobile ad hoc networks. The first link reversal algorithms are due to Gafni and Bertsekas [5]. Link reversal is the basis of the TORA [8] routing algorithm, and has also been used in the design of leader election algorithms for mobile ad hoc networks [6]. Link reversal routing is best suited for networks where the rate of topological changes is high enough to rule out algorithms based on shortest paths, but not so high as to make flooding the only alternative.

In the graph representing the network, each node has a link with each other node in its transmission radius. For any given *destination* node, the link reversal algorithms are applied on top of this underlying graph, which they convert to a *destination oriented graph* (see Figure 1). The links (edges) of the network are assigned directions, such that the resulting directed graph is acyclic and every directed path in the graph leads to the destination. Routing on a destination oriented network is easy: when a node receives a packet, it forwards the packet on *any* outgoing link, and the packet will eventually reach the destination.[*]

The task of the link reversal algorithm is to create and maintain the routes to the destination. When two nodes move out of range from one another, the link between them gets destroyed, and some nodes might lose their routes. The routing algorithm reacts by performing *link reversals* (i.e. re-orienting some of the edges) so that the resulting directed graph is again destination oriented. In particular, when a node finds that it has become a *sink* (has lost all of its outgoing links), then the node reacts by reversing the directions of some or all of its incoming links. The link reversals due to one node may cause adjacent nodes to perform reversals, and in this way, the reversals propagate in the network until the routes to the destination are reestablished.

Gafni and Bertsekas [5] describe a general family of link reversal algorithms, and present two particular algorithms: the *full reversal* algorithm and the *partial reversal* algorithm (referred to as the GB algorithms in the rest of this paper). In the full reversal algorithm, when a node becomes a sink it reverses the directions of all of its incident links. In the partial reversal algorithm, the sink reverses the directions only of those incident links that have not been reversed by adjacent nodes. The full reversal algorithm is simpler to implement, but the partial reversal algorithm may need fewer link reversals in the typical case. Gafni and Bertsekas show that when link failures occur, these algorithms *eventually* converge to a destination oriented graph. However, it was not known how many reversals the nodes performed, or how much time it would take till convergence.

1.1 Our Results

We present the first formal performance analysis of link reversal routing algorithms. We give tight upper and lower bounds on the performance of the full and partial reversal algorithms. We also show a lower bound on the performance of *any* deterministic link reversal algorithm. Surprisingly, from the perspective of worst-case performance, the full reversal algorithm is asymptotically optimal while the partial reversal algorithm is not.

Our setting for analysis is as follows. Suppose topological changes occur in the network, driving the system to a state where some nodes have lost their paths to the destination. This is called the *initial state* of the network. If there are no further topological changes, the network is said to have *stabilized* when it again becomes destination oriented (i.e. reaches a *final state*). We analyze two metrics:

[*]If there are multiple destinations in the network, then there is a separate directed graph for each destination; here, we will assume for simplicity that there is only one destination.

Work: The number of node reversals till stabilization. This is a measure of the power and computational resources consumed by the algorithm in reacting to topological changes.

Time: The number of parallel steps till stabilization, which is an indication of the speed in reacting to topological changes. We model reversals so that each reversal requires one time step, and reversals may occur simultaneously whenever possible.

Reversals are implemented using *heights*. A reversal algorithm assigns a *height* to every node in the network. The link between adjacent nodes is directed from the node of greater height to the node of lesser height. Formally, a node v is a *sink* if all of v's adjacent links are pointing in, and v is not the destination. A sink performs a reversal by increasing its height by a suitable amount. This will reverse the direction of some or all of its incident links. Unless otherwise stated, we consider *deterministic* link reversal algorithms, in which a sink node increases its height according to some deterministic function of the heights of the adjacent nodes. The GB link reversal algorithms are deterministic.

We say that a node is *bad* if there is no route from the node to the destination. Any other node, including the destination, is *good*. Note that a bad node is not necessarily a sink.

Our main results are as follows:

Full Reversal Algorithm For the full reversal algorithm, we show that when started from an initial state with n bad nodes, the work and time needed to stabilize is $O(n^2)$. This bound is tight. We show that there are networks with initial states which require $\Omega(n^2)$ time for stabilization.

Our result for full reversal is actually stronger. For any network, we present a decomposition of the bad nodes in the initial state into *layers* which allows us to predict *exactly* the work performed by each node in *any* distributed execution. A node in layer j will reverse exactly j times before stabilization. Our lower and upper bounds follow easily from the exact analysis.

Partial Reversal Algorithm For the partial reversal algorithm, we show that when started from an initial state with n bad nodes, the work and time needed to stabilize is $O(n \cdot a^* + n^2)$, where a^* corresponds to the difference between the maximum and minimum height of the nodes in the initial state. This bound is tight. We show that there are networks with initial states which require $\Omega(n \cdot a^* + n^2)$ time for stabilization.

The a^* value can grow unbounded as topological changes occur in the network. Consequently, in the worst-case, the full reversal algorithm outperforms the partial reversal algorithm. This suggests that it might be worth rethinking the popular partial reversal algorithm to see if it can have good average case and worst case performance.

Deterministic Algorithms We show a lower bound on the worst case work and time till stabilization for *any* deterministic reversal algorithm. We show that for any deterministic reversal algorithm, there exist networks and initial states with n bad nodes such that the algorithm needs $\Omega(n^2)$ work and time till stabilization. As a consequence, from the worst-case perspective, the full reversal algorithm is work and time optimal, while the partial reversal algorithm is not.

Equivalence of Executions We show that for any deterministic reversal algorithm, all distributed executions of the algorithm starting from the same initial state are equivalent: (1) each node performs the same number of reversals till stabilization in all executions, and (2) the resulting final state of the network upon stabilization is the same. As a result, the work of the algorithm as a whole is independent of the execution schedule.

1.2 Related Work

Link reversal algorithms were introduced by Gafni and Bertsekas in [5]. In that paper the authors provide a proof that shows that a general class of link reversal algorithms, including the partial and full reversal algorithms, eventually stabilize when started from any initial state.

The TORA [8] algorithm (Temporally Ordered Routing Algorithm) builds on a variation of the GB partial reversal algorithm, and adds a mechanism for detecting and dealing with partitions in the network. The practical performance of TORA has been studied in [7]. Another link reversal routing algorithm is the LMR [3, 4] algorithm (Lightweight Mobile Routing Algorithm). An overview of link reversal routing algorithms can be found in [9, Chapter 8]. A performance comparison of various ad hoc routing algorithms, including TORA, is presented in [1]. Further surveys can be found in [10, 11].

A mobility aware leader election algorithm is built in [6] on top of TORA, and the authors present partial correctness proofs (TORA does not have any) showing the stability of the algorithm. None of the above works have any formal analysis of the performance of link reversal algorithms.

The rest of the paper is organized as follows. Section 2 contains a description of the GB partial and full reversal algorithms. In Section 3 we show that the equivalence of executions of a given deterministic algorithm. Sections 4 and 5 contain the analyses of the full and partial reversal algorithms respectively. In Section 6, we show the general lower bound for deterministic link reversal algorithms. Finally, in Section 7, we conclude with a discussion and open problems.

2 Link Reversal Algorithms

We assume that each node has an unique integer id, and denote the node with id i by v_i. The nodes have heights which are guaranteed to be unique (ties broken by node ids), and are chosen from a totally ordered set. The destination has the smallest height. Since any directed path in such a graph always proceeds in the direction of decreasing height, *the directed graph will be acyclic* (DAG). If the graph is destination oriented, all directed paths end at the destination. There could possibly be multiple paths from any node to the destination. Note that the graph remains a DAG even when topological changes occur. If the underlying graph is connected, the link reversal algorithms bring the directed graph from its initial state to a state where it is destination oriented. In our analysis, we only consider connected graphs. We now describe the GB algorithms, adapting the discussion from [5], and then define the class of deterministic algorithms.

Full Reversal Algorithm In the full reversal algorithm, when a node becomes a sink it simply reverses the directions of all its incoming links (see Figure 1). The algorithm can be implemented with heights as follows. The height h_i of node v_i is the pair (a_i, i) (the second field is used to break ties). The height of the destination (say v_d) is $(0, d)$. Heights are ordered lexicographically. If v_i is a sink, then its height upon reversal is updated to be larger than the heights of all its neighbors. Let $N(v_i)$ denote the set of adjacent nodes to v_i. Formally, the height of v_i after its reversal is $(\max\{a_j \mid v_j \in N(v_i)\} + 1, i)$.

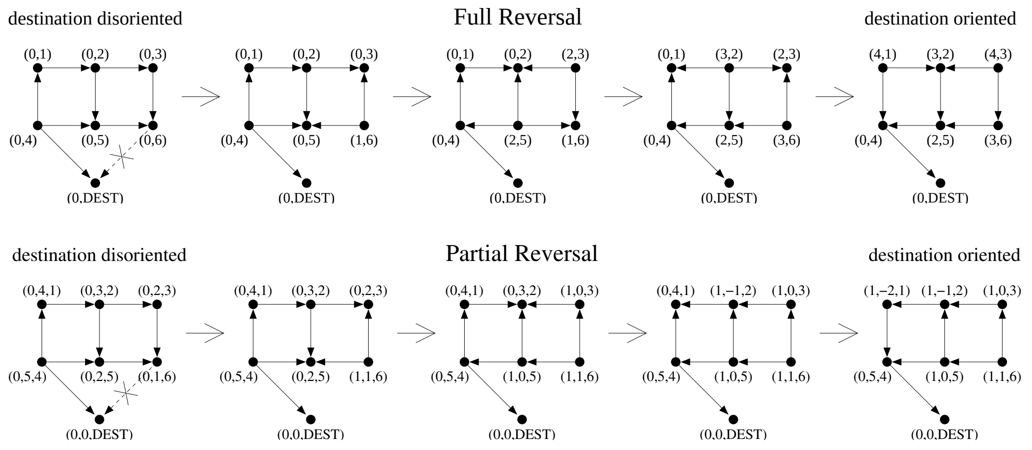
destination disoriented
Full Reversal
destination oriented
(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,DEST)
(0,1) (0,2) (0,3) (0,4) (0,5) (1,6) (0,DEST)
(0,1) (0,2) (2,3) (0,4) (2,5) (1,6) (0,DEST)
(0,1) (3,2) (2,3) (0,4) (2,5) (3,6) (0,DEST)
(4,1) (3,2) (4,3) (0,4) (2,5) (3,6) (0,DEST)
destination disoriented
Partial Reversal
destination oriented
(0,4,1) (0,3,2) (0,2,3) (0,5,4) (0,2,5) (0,1,6) (0,0,DEST)
(0,4,1) (0,3,2) (0,2,3) (0,5,4) (0,2,5) (1,1,6) (0,0,DEST)
(0,4,1) (0,3,2) (1,0,3) (0,5,4) (1,0,5) (1,1,6) (0,0,DEST)
(0,4,1) (1,–1,2) (1,0,3) (0,5,4) (1,0,5) (1,1,6) (0,0,DEST)
(1,–2,1) (1,–1,2) (1,0,3) (0,5,4) (1,0,5) (1,1,6) (0,0,DEST)

Figure 1: Sample executions of the GB full and partial reversal algorithms

Partial Reversal Algorithm In the partial reversal algorithm, every node v_i other than the destination keeps a "list" of its neighboring nodes v_j that have reversed links into v_i. If v_i becomes a sink then it reverses the directions of the links to every neighbor v_j which was not present in this list, and empties the list. If no such v_j exists (i.e. the list contains all its neighbors), then v_i empties the list and reverses all its links (see Figure 1). This can be implemented using heights in the following way.

The height h_i of each node v_i is the triple (a_i, b_i, i). Essentially, the field a_i represents the height of v_i, and b_i implements the list of nodes which have reversed since the last reversal of v_i. The height of the destination v_d is $(0, 0, d)$. Heights are ordered lexicographically. If node v_i is a sink then, when it reverses, its height is updated to be bigger than the height of every neighbor which is not in the list. Formally, let $\bar{h}_i = (\bar{a}_i, \bar{b}_i, i)$ denote the height of v_i after its reversal. We have, $\bar{a}_i = \min\{a_j \mid v_j \in N(v_i)\} + 1$. Moreover, $\bar{b}_i = \min\{b_j \mid v_j \in N(v_i) \text{ and } \bar{a}_i = a_j\} - 1$, if there exists a neighbor v_j with $\bar{a}_i = a_j$, otherwise, $\bar{b}_i = b_i$. Note that if an adjacent node of v_i is in the "list" of v_i before v_i reverses, then it must be that $\bar{a}_i = a_j$. In that case, $\bar{b}_i$ will be smaller than the b_j of any node in the list, and the links from these nodes towards v_i are not reversed.

Deterministic Algorithms A deterministic reversal algorithm is defined by a "height increase" function g. We assume that the heights are chosen from some totally ordered universe, and that the heights of different nodes are unique. If node v is a sink whose current height is h_v, and adjacent nodes $v_1, v_2 \ldots v_d$ have heights $h_1, h_2 \ldots h_d$ respectively, then v's height after reversal is $g(h_1, h_2 \ldots h_v)$. The GB full and partial reversal algorithms are deterministic.

3 Equivalence of Executions

In this section, we prove some properties about general reversal algorithms. The main result of this section is that for any deterministic reversal algorithm, all executions that start from the same initial state are essentially equivalent. We first prove a basic lemma that holds for any reversal algorithm, whether deterministic or not. This result is also proved in [5], however we believe our proof is simpler.

Lemma 3.1 *For any reversal algorithm starting from any initial state, a good node never reverses till stabilization.*

Proof: If v is a good node, then by definition there exists a path $v = v_k, v_{k-1}, \dots v_1, v_o = s$ where s is the destination, and there is a edge directed from v_i to v_{i-1} for $i = 1 \dots k$. For every $i = 0 \dots k$, we prove that node v_i never reverses, using induction on i.

The base case ($i = 0$) is obvious since the destination $s = v_0$ never reverses. Suppose the hypothesis is true for $i = l < k$. Then v_l never reverses, so that the edge between v_{l+1} and v_l is always directed from v_{l+1} to v_l. Thus, there is always an outgoing edge from v_{l+1}, which implies that v_{l+1} never reverses. ■

When started from an initial state, the algorithm reverses nodes until no more reversals are possible, and the network is destination oriented. The *execution* of a reversal algorithm is a sequence of reversals. A *full execution* starts in an initial state and ends in a destination oriented graph. At each step of the execution, the algorithm non-deterministically chooses any of the current sinks and reverses it, according to some strategy. Clearly, there are many possible executions starting from the same initial state, since there is a choice of many possible reversals at each execution step. For a deterministic reversal algorithm, a reversal r can be viewed as a tuple $r = (v, h, H)$ where v is the sink executing the reversal, h is v's height before reversal, and H is the set of the heights of all of v's neighbors before the reversal.

Any execution imposes a partial order on the reversals. The partial order induced by execution $R = r_1, r_2, \dots, r_k$ where $r_i = (v_i, h_i, H_i)$, is defined as a directed graph in which the nodes are the reversals r_i, $i = 1, \dots, k$. In this graph, there is a directed edge from $r_i = (v_i, h_i, H_i)$ to $r_j = (v_j, h_j, H_j)$ if (1)v_j is a neighbor of v_i, and (2)r_j is the first reversal of v_j after r_i in execution R. We will refer to this graph as the *dependency graph* of the execution. Intuitively, if there is a directed path between reversals r_i and r_j in the dependency graph, then the order of reversals r_i and r_j cannot be interchanged in the execution. Moreover, if there is no directed path from r_i to r_j and vice versa, then these two reversals are independent and can be performed in *parallel* (in the same time step). We define the *depth* of a reversal in the dependency graph as follows. A reversal which does not have any incoming edges has depth 0. The depth of any other reversal r is one more than the maximum depth of a reversal which points to r. The depth of the dependency graph is the maximum depth of any reversal in the graph.

We say that two executions are *equivalent* if they impose the same dependency graph. We will show that all executions of a link reversal algorithm are equivalent. We first show a lemma which will be of use in further proofs.

Lemma 3.2 *If a node is a sink, it remains a sink even if other nodes in the network reverse.*

Proof: If a node v is a sink, then clearly none of its neighbors can be sinks at the same time. The only node which can change the direction of the incoming links to v is v itself. Reversals by other nodes in the network do not affect this. ■

The following is the main theorem of this section.

Theorem 3.3 *Any two executions of a deterministic reversal algorithm starting from the same initial state are equivalent.*

Proof: Consider two executions starting from the same initial state, say $R = r_1, r_2, \ldots, r_k$ and $S = s_1, s_2, \ldots, s_l$. Let p_R and p_S be the dependency graphs induced by R and S respectively. In order to show that P and R are equivalent, we need to show that p_R and p_S are identical. We will show by induction that for for every $k = 0, 1 \ldots$, the induced subgraph of p_R, consisting of vertices at depths $\leq k$, is identical to the similar induced subgraph of p_S consisting of vertices at depths $\leq k$.

Base case $k = 0$: Consider any reversal in p_R at depth 0, say $r = (v, h, H)$. Since r does not have any incoming edges in p_R, v must be a sink in the initial state of the network. From Lemma 3.2, v must also reverse in S. Since h and H are the heights of v and its neighbors respectively in the initial state, the first reversal of v in S is also (v, h, H), and is at depth 0. Similarly, we can show that every reversal at level 0 in p_S is a reversal at level 0 in p_R. This completes the proof of the base case.

Inductive case: Suppose the hypothesis was true for all $k < l$. We show that it is true for $k = l$. Consider any reversal $r = (v, h, H)$ at depth l in p_R. We show that this reversal is also present in p_S with the same set of incoming edges. Let V be the set of vertices that are pointing into r in p_R. Once all reversals in V are executed, node v is a sink in execution R. From the inductive step, all reversals in V are also present in p_S and hence in S.

Case 1: r is the first reversal of v in R. Then, the reversal of every node in V will also cause v to be a sink in S. So, v will reverse in S. Its height before reversal in S is h, since the height has not changed from the initial state. Consider the heights of the neighbors of v in S during v's reversal. These are the same as H. The reason is as follows. The neighbors of v who haven't reversed so far in S have the same height as in the initial state. The other neighbors are present in V and hence their heights are the same as in H. Thus, there is a node (v, h, H) at level l in p_S whose incoming edges are the same as in p_R.

Case 2: r is not the first reversal of v in R. This case can be treated similar to Case 1.

Thus, we have shown that every node in level l of p_R is present in level l of p_S, with the same incoming edges. The same argument goes the other way too: every node in p_S is present in p_R. This proves the inductive case for $k = l$, and concludes the proof. ■

It is easy to see that the dependency graph uniquely determines the final state and the work needed by each processor. Therefore, we derive the following corollaries from Theorem 3.3.

Corollary 3.4 *For all executions of a deterministic reversal algorithm starting from the same initial state: (1) the final state is the same, and (2) the number of reversals of each node is the same.*

Corollary 3.5 *The time of execution of a deterministic reversal algorithm is lower-bounded by the depth of the dependency graph corresponding to the initial state, and is the minimum possible when all the sink nodes reverse simultaneously.*

Proof: Suppose the depth of the partial order graph was d. There exists a directed path of length d in the dependency graph. No two reversals on this path can execute in parallel, and the time taken for the all reversals in this path to complete is at least $d + 1$. Hence $d + 1$ is a lower bound on the time for the execution.

Now, if all sink nodes reversed immediately, we have the invariant that after k time steps, all the reversals at depth $k - 1$ have completed. Thus, the execution would be complete in time $d + 1$, which is the minimum possible. ■

4 Full Reversal Algorithm

In this section, we present the analysis of the full reversal algorithm. Our analysis is *exact.* We present a decomposition of the bad nodes in the initial state into *layers* which allows us to predict *exactly* the work performed by each node in any distributed execution till stabilization. From these, the worst case bounds follow easily.

4.1 State Sequence for Full Reversal

We show that starting from any initial state, there exists an execution which consists of consecutive segments, such that each bad node reverses exactly once in each segment.

Lemma 4.1 *Consider a state I in which a node v is bad. Then, node v will reverse at least one time before it becomes a good node.*

Proof: If v is a sink, then clearly node v has to reverse at least one time. Now consider the case where v is not a sink in state I. Suppose, for contradiction, that node v becomes good without performing any reversals after state I. Let E be an execution which brings the graph from state I to a state I^g in which node v is good. A *non-reversed* node is any node w such that in state I node w is bad, while in state I^g node w is good, and w didn't reverse between I and I^g. In state I^g, node v is good; thus, in I^g there must exist a directed path $v, v_1, \ldots, v_{k-1}, v_k$, $k \geq 1$, in which all nodes are good, while in state I, $v_1, \ldots, v_{k-1}$ are bad, and v_k is good.

We will show that nodes $v_1, \ldots, v_{k-1}$ are non-reversed. Consider node v_1. Assume for contradiction that node v_1 has reversed between states I and I^g. Since in I^g there is a link directed from node v to node v_1, and v_1 has reversed, it must ne that node v has reversed at least one time; a contradiction. Thus, node v_1 is non-reversed. Using induction, we can easily show in a similar fashion that nodes $v_2, \ldots, v_{k-1}$ are also non-reversed. Since nodes $v_1, \ldots, v_{k-1}$ are non-reversed, it must be that in state I there is a directed path $v, v_1, \ldots, v_{k-1}, v_k$. Thus, in state I node v is a good node. A contradiction. ■

Lemma 4.2 *Consider some state I which contains bad nodes. There exists an execution E which brings the system from state I to a state I', such that every bad node of state I reverses exactly one time in E.*

Proof: Assume for contradiction that there is no such execution E in which each bad node reverses exactly one time. There must exist an execution E^f which brings the system from state I to a state I^f such that the following conditions hold: (i) there is at least one bad node in I which hasn't reversed in E^f; let A denote the set of such bad nodes of I; (ii) any other bad node v of I, with $v \notin A$, has reversed exactly one time; let B denote the set of such bad nodes of I; (iii) the number of nodes in set B is maximal.

From condition (iii), it must be that all the nodes that are sink in state I^f belong to set B, that is, only nodes of set B are ready to reverse in I^f, since B is maximal. From Lemma 4.1, we have that each node of set A is bad in state I^f. We will show that at least one node in A is a sink in state I^f, which violates condition (iii).

Assume for contradiction that no node of A is a sink in I^f. Then, each node in A has an outgoing edge in I^f. These outgoing edges from A cannot be towards nodes in B, since the nodes in B have reversed their edges, while the nodes in A haven't. Moreover, these outgoing edges cannot be towards good nodes of state I, since this would imply that nodes in A are good. Thus, these outgoing edges must be toward nodes in set A. Since each node in set A has an outgoing edge in set A, it must be, from the pigeonhole principle, that there is a walk in which a node in A is repeated. Thus, there is a cycle in the graph, violating the fact that the graph is acyclic. A contradiction. Thus, it must be that a node in A is a sink. A contradiction. ■

Consider some initial state I_1 of the graph which contains bad nodes. Lemma 4.2 implies that there is an execution $E = E_1, E_2, E_3, \ldots$, and states $I_1, I_2, I_3, \ldots$, such that execution segment E_i, $i \geq 1$, brings the network from a state I_i to a state I_{i+1}, and in E_i each bad node of I_i reverses exactly one time. The *node-state* of a node v is the directions of its incident links. We show that each execution segment leaves the node-state of bad nodes unchanged (when the bad nodes are not adjacent to good nodes).

Lemma 4.3 *At a state I_i, $i \geq 1$, any bad node not adjacent to a good node will remain in the same (bad) node-state in I_{i+1}.*

Proof: Let $A(v)$ denote the set of nodes adjacent to v in state I_i. Since all nodes in $A(v)$ are bad in state I_i, each of them reverses in execution E_i. Moreover, v also reverses in E_i. These reversals leave the directions of the links between v and $A(v)$ in state I_{i+1} the same as in state I_i. ■

4.2 Layers for Full Reversal

Consider the nodes of the network in some state I which contains bad nodes. We can partition the bad nodes into layers $L_1^I, L_2^I, \ldots, L_m^I$, as follows (see Figure 2). A bad node v is in layer L_1^I if the following conditions hold: (i) there is an incoming link to node v from a good node, or (ii) there is an outgoing link from node v to a node in layer L_1^I. A node v is in layer L_k^I, $k > 1$, if k is the smallest integer for which one of the following hold: (i) there is an incoming link to node v from a node in layer L_{k-1}^I, or (ii) there is an outgoing link from node v to a node in layer L_k^I. From the above definition, it easy to see that any node of layer L_k^I, where $k > 1$, can be connected only with nodes in layers L_{k-1}^I, L_k^I and L_{k+1}^I. The nodes of layer L_1^I are the only ones that can be connected with good nodes. The links connecting two consecutive layers L_{k-1}^I and L_k^I can only be directed from L_{k-1}^I to L_k^I. Note that the number of layers m is $m \leq n$, where n is the number of bad nodes in the network.

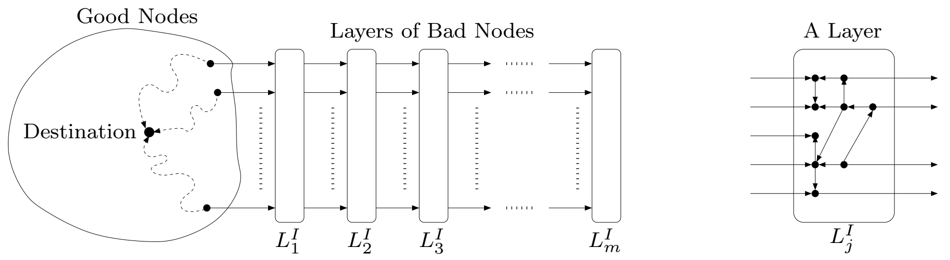

Figure 2: Partitioning the nodes into layers

Consider now the states $I_1, I_2, \ldots$ and execution segments $E_1, E_2, \ldots$, as described above. For each of these states we can divide the bad nodes into layers as described above. In the next results, we will show that the layers of state I_1 become good one by one, at the end of each execution segment E_i, $i \geq 1$. First we show that the first layer of state I_i becomes good at the end of execution E_i.

Lemma 4.4 *At the end of execution E_i, $i \geq 1$, all the bad nodes of layer $L_1^{I_i}$ become good, while all the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad.*

Proof: First we show that the bad nodes of layer $L_1^{I_i}$ become good. There are two kinds of bad nodes in layer $L_1^{I_i}$ at state I_i: (i) nodes which are connected with an incoming link to a good node, and (ii) nodes which are connected with an outgoing link to another node in layer $L_1^{I_i}$. It is easy to see that there is a direct path from any type-ii node to some type-i node, consisting from nodes of layer $L_1^{I_i}$. Since all bad nodes reverse exactly once in execution E_i, all type-i nodes become good in state I_{i+1}. Moreover, from Lemma 4.3, the paths from type-ii nodes to type-i remain the same in state I_{i+1}. Thus, the type-ii nodes become also good in state I_{i+1}. Therefore, the bad nodes of layer $L_1^{I_i}$ become good in state I_{i+1}.

Now we show that the bad nodes in layers $L_j^{I_i}$, $j > 1$ remain bad in state I_{i+1}. From Lemma 4.3, in state I_{i+1}, the links connecting layers $L_1^{I_i}$ and $L_2^{I_i}$ are directed from $L_1^{I_i}$ to $L_2^{I_i}$. Thus, in state I_{i+1}, there is no path connecting nodes of layer $L_2^{I_i}$ to good nodes. Similarly, there is no path from the nodes of layer $L_j^{I_i}$, for any $j > 2$, to good nodes. Thus all nodes in layers $L_j^{I_i}$, $j > 1$, remain bad. ■

We now show that the basic structure of layers of the bad nodes remains the same from state I_i to state I_{i+1}, with the only difference that the first layer of I_{i+1} is now the second layer of I_i.

Lemma 4.5 $L_j^{I_{i+1}} = L_{j+1}^{I_i}$, $i, j \geq 1$.

Proof: From Lemma 4.4, at the end of execution E_i, all the bad nodes of layer $L_1^{I_i}$ become good, while all the bad nodes in layers $L_j^{I_i}$, $j > 1$ remain bad. From Lemma 4.3 all bad nodes in layers $L_j^{I_i}$, $j > 1$, remain in the same node-state in I_{i+1} as in I_i. Therefore, $L_j^{I_{i+1}} = L_{j+1}^{I_i}$, $j \geq 1$. ■

From Lemmas 4.4 and 4.5, we have that the number of layers is reduced by one from state I_i to state I_{i+1}. If we consider the layers of the initial state I_1, we have that all the bad nodes in the layers become good one by one at the end of executions $E_1, E_2, E_3, \ldots$ with the order $L_1^{I_1}, L_2^{I_1}, L_3^{I_1}, \ldots$. Since at each execution E_i all the bad nodes reverse exactly one time, we obtain the following:

Lemma 4.6 *Each node in layer $L_j^{I_1}$, $j \geq 1$, reverses exactly j times before it becomes a good node.*

From Corollary 3.4, we know that all possible executions when started from the same initial state require the same number of reversals. Thus, the result of Lemma 4.6, which is specific to the particular execution E applies to all possible executions. Therefore, we obtain the following result.

Theorem 4.7 *For any initial state I, and any execution of the full reversal algorithm, each node in layer L_j^I, $j \geq 1$, reverses exactly j times before it becomes a good node.*

4.3 Bounds for Full Reversal

From Theorem 4.7, we have that for any initial state I, each node in layer L_j^I reverses exactly j times until it becomes good. Thus, the total number of reversals of the nodes of layer j is $j \cdot |L_j^I|$. If there are k layers, the total number of reversals is $\sum_{j=1}^{k} j \cdot |L_j^I|$. If I contains n bad nodes, there are in the worst case at most n layers (each layer contains one bad node). Thus, each node reverses at most n times. Since there are n bad nodes, the total number of reversals in the worst case is $O(n^2)$. Moreover, since a node reversal takes one time step and in the worst case all reversals are executed sequentially, the total number of reversals gives an upper bound on the stabilization time. Thus, we have:

Corollary 4.8 *For any graph with an initial state with n bad nodes, the full reversal algorithm requires at most $O(n^2)$ work and time until stabilization.*

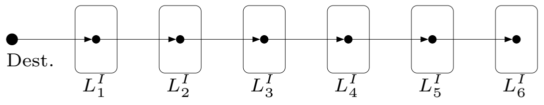

Figure 3: Graph G_1 with 6 bad nodes

Actually, the upper bound of Corollary 4.8 is tight in both work and time in the worst case. First we show that the work bound is tight. Consider a graph G_1 with an initial state in which the destination is the only good node and the remaining nodes are bad and partitioned into n layers such that each layer has exactly one node (see Figure 3). From Theorem 4.7, each node in the ith layer will reverse exactly i times. Thus, the sum of all the reversals performed by all the bad nodes is $n(n+1)/2$. Therefore, we have the following corollary.

Corollary 4.9 *There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ work until stabilization.*

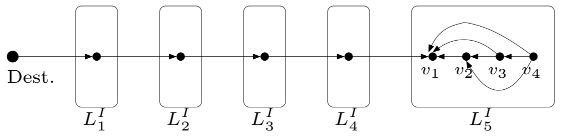

Figure 4: Graph G_2 with 8 bad nodes

We will show that the time bound of Corollary 4.8 is tight (within constant factors) in the worst case. Consider a graph G_2 in an initial state in which there are n bad nodes, such that it consists of $m_1 = \lfloor n/2 \rfloor + 1$ layers. The first $m_1 - 1$ layers contain one node each, while the last layer contains $m_2 = \lceil n/2 \rceil$ nodes. The last layer m_1 is as follows: there are m_2 nodes $v_1, v_2, \ldots, v_{m_2}$. Node v_i has outgoing links to all nodes v_j such that $j < i$. The node of layer $m_1 - 1$ has an outgoing link to node v_1 (see Figure 4).

From Theorem 4.7, we know that each node in layer m_1 requires exactly m_1 reversals before it becomes good. Since there are m_2 nodes in layer m_1, $m_1 \cdot m_2 = \Omega(n^2)$ reversals are required before these nodes become good. All these reversals have to be performed sequentially, since the nodes of layer m_1 are adjacent, and any two of these nodes cannot be sinks simultaneously. We obtain the following corollary.

Corollary 4.10 *There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ time until stabilization.*

5 Partial Reversal Algorithm

In this section, we present the analysis of the partial reversal algorithm. We first give a general upper bound, and then present lowers bounds for a class of worst case graphs.

5.1 Upper Bounds for Partial Reversal

According to the partial reversal algorithm, each node v_i has a height (a_i, b_i, i). We will refer to a_i as the *alpha value* of node v_i. Consider an initial state I of the network containing n bad nodes. We say that a bad node v of state I is in level i if the shortest undirected path from v to a good node has length i. Note that the number of levels is between 1 and n. Let $a^{\max}$ and $a^{\min}$ denote the respective maximum and minimum alpha values of any node in the network in state I. Let $a^* = a^{\max} - a^{\min}$.

Lemma 5.1 *When a node in level i becomes good, its alpha value does not exceed $a^{\max} + i$.*

Proof: We prove the claim by induction on the number of levels. For the induction basis, consider a node v in level 1. If the alpha value of v becomes at least $a^{\max} + 1$, then v must have become a good node, since its height is more that the height of the adjacent nodes which are good in state I (these good nodes don't reverse, and thus their alpha values remain the same in any state of the network). We only need to show that during its final reversal, the alpha value of v will not exceed $a^{\max} + 1$. According to the partial reversal algorithm, the alpha value of v is equal to the smallest alpha value of its neighbors plus one. Moreover, the smallest alpha value of the neighbors cannot be greater than $a^{\max}$, since in I, v is adjacent to good nodes which don't reverse in future states. Thus, the alpha value of v will not exceed $a^{\max} + 1$, when v becomes a good node.

For the induction hypothesis, let's assume that the alpha value of any node in level i, where $1 \le i < k$, does not exceed $a^{\max}+i$, when that node becomes good. For the induction step, consider layer L_k. Let v be a node in level k. Clearly, node v is adjacent to some node in level $k-1$. From the induction hypothesis, the alpha value of every node in level $k-1$ can not exceed $a^{\max} + (k-1)$ in any future state from I. If the alpha value of v becomes at least $a^{\max} + k$, then v must have become a good node, since its height is more than that of the adjacent nodes in level $k-1$ when these nodes become good. We only need to show that during its final reversal, the alpha value of v will not exceed $a^{\max} + k$. According to the partial reversal algorithm, the alpha value of v is not more than the smallest alpha value of its neighbors plus one. Moreover, the smallest alpha value of the neighbors cannot exceed $a^{\max} + (k-1)$ which is the maximum alpha value of the nodes in level $k-1$ when these node become good. Thus, the alpha value of v will not exceed $a^{\max} + k$, when v becomes a good node. ■

At each reversal, the alpha value of a node increases by at least 1. Since the alpha value of a node can be as low as $a^{\min}$, Lemma 5.1 implies that a node in level i reverses at most $a^{\max} - a^{\min} + i$ times. Furthermore, since there are at most n levels, we obtain the following corollary.

Corollary 5.2 *A bad node will reverse at most $a^* + n$ times before it becomes a good node.*

Considering now all the n bad nodes together, Corollary 5.2 implies that the work needed until the network stabilizes is at most $n \cdot a^* + n^2$. Since in the worst case the reversal of the nodes may be sequential, the upper bound for work is also an upper bound for the time needed to stabilize. Thus we have:

Theorem 5.3 *For any initial state with n bad nodes, the partial reversal algorithm requires at most $O(n \cdot a^* + n^2)$ work and time until the network stabilizes.*

We would like to note that there are scenarios that result in initial states, such that, the a^* value may be arbitrarily large. For example, while topological changes occur in the network, two or more adjacent nodes may alternate between bad and good nodes, which may cause them to increase their height to high alpha values. At the same time, some nodes in the network may remain good, with low alpha values. In such a scenario, a^* is large.

5.2 Lower Bounds for Partial Reversal

In a state of a network, we say that a node is a *source* if all the links incident to the node are outgoing. A *full reversal* is a reversal in which a node reverses all of its links. Note that after a full reversal, a node becomes a source. We show that bad nodes which are sources always perform full reversals whenever they become sinks.

Lemma 5.4 *Consider any state I of the network in which a bad node v is a source with alpha value a. In a subsequent state I', in which node v becomes a sink for the first time after state I, the following occur: (1) v performs a full reversal, and (2) after the reversal of v, the alpha value of v becomes $a + 2$.*

Proof: In state I, since v is a source, all the adjacent nodes of v have alpha value at most a. Between states I and I', each adjacent node of v has reversed at least once. We will show that in state I', the alpha value of each adjacent node of v is $a + 1$.

Let w be any adjacent node of v. First, we show that the alpha value of v in I' is at least $a+1$. If in I' the alpha value of w is less than a then v must have an outgoing link towards w, and thus v cannot possibly be a sink in I', a contradiction. Therefore, in I' the alpha value of w, has to be at least a. Next, we show that this alpha value cannot be equal to a. If the alpha value of w in I' is a then it must be that the alpha value of v in I was less than a (since w reversed between I and I'). When w was a sink the last time before I', w must have been adjacent to another node u with height $a - 1$. When w reversed, its alpha value became a, but its incoming link from v didn't change direction since u had a smaller alpha value. Thus v cannot possibly be a sink in I', a contradiction. Therefore, the alpha value of w in I' cannot be equal to a, and it has to be at least $a + 1$.

Next, we show that the alpha value of v cannot be greater than $a + 1$. When w reverses, its alpha value is at most the minimum alpha value of its neighbors, plus one. Therefore, since v is a neighbor of w with alpha value a, when w reverses its alpha value cannot exceed $a + 1$.

Therefore, the alpha value of w in state I' is exactly $a+1$. This implies that in I' all the neighbors of v have alpha value $a+1$. Thus, when v reverses, it performs a full reversal and its alpha value becomes $a+2$. ■

Here, we consider special cases of graphs in which the bad nodes are partitioned into layers in a particular way as we describe below. Consider a graph with an initial state I containing n bad nodes such that the bad nodes are partitioned into an even number m of layers $L_1, L_2, \dots, L_{m-1}, L_m$ in the following way. The odd layers $L_1, L_3, \dots, L_{m-1}$ contain only nodes which are non-sources, while the even layers $L_2, L_4, \dots, L_m$ contain only nodes which are sources. The nodes in layer L_1 are the only bad nodes adjacent to good nodes. Let G denote the set of good nodes adjacent to layer L_1. Nodes in layer L_i may be adjacent only to nodes of the same layer and layers L_{i-1} and L_{i+1}, such that each node of L_i is adjacent to at least one node of L_{i-1} and at least one node of L_{i+1}.[†]

Let $a^{\max}$ and $a^{\min}$ denote the respective maximum and minimum alpha values of any node in the network in state I. Let $a^* = a^{\max} - a^{\min}$. State I is such that all good nodes in the network have alpha value $a^{\max}$, while all the bad nodes have alpha value $a^{\min}$. First we show an important property.

Lemma 5.5 *When the network stabilizes, the alpha values of all the nodes in layers L_{2i-1} and L_{2i}, $1 \le i \le m/2$, are at least $a^{\max} + i$.*

Proof: Let I' denote the state of the network when it stabilizes. We prove the claim by induction on i. For the basis case, where $i = 1$, we consider layers L_1 and L_2. In state I, all the nodes of layer L_1 have only incoming links from G. In state I', there must exist a set S, consisting from nodes of L_1, such that the nodes in S have outgoing links towards G.

Let v be a node in S. In state I', the alpha value of v is at least $a^{\max}$, since the nodes in G have alpha value $a^{\max}$. Moreover, we can show that the alpha value of v in I' is not $a^{\max}$. Assume for contradiction that this value is $a^{\max}$. When node v reversed and obtained the alpha value $a^{\max}$, it cannot possibly have reversed its links towards G, since for these links, v adjusted only its second field on its height. Thus, in state I' node v is still bad, a contradiction. Therefore, in state I', node v has alpha value at least $a^{\max} + 1$; thus, in state I', all nodes in set S have alpha value at least $a^{\max} + 1$.

Now, consider the rest of the nodes in layers L_j, $j \ge 1$. Let w be any such node. In state I', w is good, and thus there exists a directed path from w to a good node in G. This path has to go through the nodes of S; thus each node in the path must have alpha value at least $a^{\max} + 1$, which implies that w has alpha value at least $a^{\max} + 1$. Therefore, in state I', all nodes in L_1 and L_2 (including S) have alpha value at least $a^{\max} + 1$.

Now, let's assume that the claim holds for all $1 \le i < k$. We will show that the claim is true for $i = k$. We consider layers L_{2k-1} and L_{2k}. In state I all the nodes of layer L_{2k-1} have only incoming links from L_{2k-2}. In state I', there must exist a set S, consisting from nodes of L_{2k-1}, such that the nodes in S have outgoing links towards L_{2k-2}. The rest of the proof is very similar with the induction basis, where now we show that the nodes in S in state I', have alpha values at least $a^{\max} + k$, which implies that all nodes in L_{2k-1} and L_{2k} have alpha value at least $a^{\max} + k$. ■

We are now ready to show the main result which is the basis of the lower bound analysis.

[†]If $i = 1$, substitute G for L_{i-1}. If $i = m$, don't consider L_{i+1}.

Theorem 5.6 *Until the network stabilizes, each node in layers L_{2i-1} and L_{2i}, $1 \leq i \leq m/2$, will reverse at least $\lfloor (a^* + i)/2 \rfloor$ times.*

Proof: Consider a bad node v of L_{2i}. Node v is a source in state I. Lemma 5.4 implies that whenever v reverses in the future, it reverses all of its incident links and therefore it remains a source. Moreover, Lemma 5.4 implies that every time that v reverses its alpha value increases by 2. From Lemma 5.5, we know that when the network stabilizes, the alpha value of v is at least $a^{\max} + i$. Since in state I the alpha value of v is $a^{\min}$, node v reverses at least $\lfloor (a^* + i)/2 \rfloor$ times after state I. Similarly, any node in L_{2i} reverses at least $\lfloor (a^* + i)/2 \rfloor$ times.

Consider now a bad node w of L_{2i-1}. Node w is adjacent to at least one node u in layer L_{2i}. In state I, node u is a source, and it remains a source every time that u reverses (Lemma 5.4). Since u and w are adjacent, the reversals of u and w should alternate. This implies that node w reverses at least $\lfloor (a^* + i)/2 \rfloor$ times, since node u reverses at least $\lfloor (a^* + i)/2 \rfloor$ times. Similarly, any node in L_{2i-1} reverses at least $\lfloor (a^* + i)/2 \rfloor$ times. ■

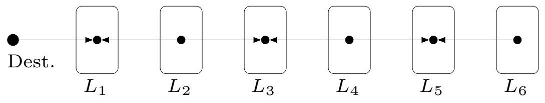

Figure 5: Graph G_3 with 6 bad nodes

Next, we give the lower bound on work. This lower bound implies that the work bound of Theorem 5.3 is tight in the worst case. Consider a graph G_3 which is in state I as described above, such that the destination is the only good node and there are n bad nodes, where n is even (see Figure 5). From Theorem 5.6, each node in the ith layer will reverse at least $\lfloor (a^* + \lceil i/2 \rceil)/2 \rfloor$ times before the network stabilizes. Thus, the sum of all the reversals performed by all the bad nodes is at least $\sum_{i=1}^{n} \lfloor (a^* + \lceil i/2 \rceil)/2 \rfloor$, which is $\Omega(n \cdot a^* + n^2)$. Thus, we have the following corollary.

Corollary 5.7 *There is a graph with an initial state containing n bad nodes, such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ work until stabilization.*

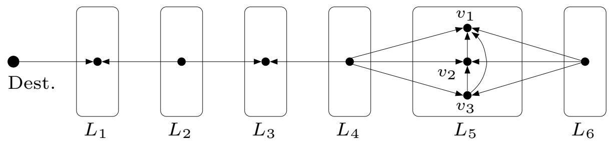

Figure 6: Graph G_4 with 8 bad nodes

Now, we give the lower bound on time. The lower bound implies that the time bound of Theorem 5.3 is tight in the worst case. Consider a graph G_4 in a state I as described above, in which there are n bad nodes, where $n/2$ is even. The graph consists of $m_1 = n/2 + 2$ layers. The first $m_1 - 2$ layers contain one node each, while layer $m_1 - 1$ contains $m_2 = n/2 - 1$ nodes, and layer m_1 contains 1 nodes. The layer $m_1 - 1$ is as follows: there are m_2 nodes $v_1, v_2, \ldots, v_{m_2}$. Node v_i has outgoing links to all nodes v_j such that $j < i$ (see Figure 6).

From Theorem 5.6, we know that each node in layer $m_1 - 1$ requires at least $k_1 = \lfloor (a^* + \lceil (m_1 - 1)/2 \rceil)/2 \rfloor$ reversals before it becomes a good node. Since layer $m_1 - 1$ contains m_2 nodes, at least $k_1 \cdot m_2 = \Omega(n \cdot a^* + n^2)$ reversals are required before these bad nodes become good nodes. All these reversals have to be performed sequentially, since the nodes of layer $m_1 - 1$ are adjacent, and any two of these nodes cannot be sinks simultaneously. Thus, we have the following corollary.

Corollary 5.8 *There is a graph with an initial state containing n bad nodes, such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ time until stabilization.*

6 Deterministic Algorithms

We now show a general lower bound on the worst case number of reversals for any deterministic reversal algorithm. In this proof, we have assumed that the heights of the nodes can be unbounded; the reversal algorithms in the literature also make the same assumption. We say that a bad node v is in *level* i if the shortest undirected path from v to a good node is i.

Theorem 6.1 *Given any height increase function g, and any network graph G, there exists an assignment of heights to the nodes in G such that a node in level d reverses at least $d - 1$ times.*

Proof: (Sketch) We assign the initial heights as follows. Let ℓ be the maximum node level. Nodes in level ℓ are all assigned the lowest possible heights. For the other levels 1 till $\ell - 1$, we guarantee that the initial heights will satisfy the following condition: if node v is at a higher numbered level than node w, then v gets a lower height than w.

We show the result for one particular execution schedule E which proceeds as follows (Theorem 3.3 generalizes the result to any execution). If the system is not yet destination oriented, then next reverse the node with the smallest height in the graph (except for the destination). The node with the smallest height is surely a sink, and hence a candidate for reversal.

We divide E into $\ell - 1$ stages, numbered 1 to $\ell - 1$. Stage i consists of all reversals in E starting from the first reversal in level $(\ell - i + 1)$ until, but not including the first reversal in level $\ell - i$.

In Lemma 6.4, we show that there exists an assignment of heights to nodes in levels $\ell - 1$ till 1 which satisfies the following condition: for $i = 1 \ldots \ell - 1$, each node in levels $(\ell - i + 1)$ till ℓ reverses at least once in stage i. Thus, a node in level d reverses at least once in every stage from $(\ell - d + 1)$ till $\ell - 1$ (both limits inclusive), and thus at least $d - 1$ times. This completes the proof sketch. ■

Before proving Lemma 6.4, we will need two other lemmas.

Lemma 6.2 *If at the start of stage i, the height of every node in levels 1 till $\ell - i$ is greater than the height of every node in levels $(\ell - i + 1)$ till ℓ, then each node in levels $(\ell - i + 1)$ till ℓ will reverse at least once in stage i.*

Proof: We prove this by contradiction. Suppose a node v in level ℓ_v where $(\ell - i + 1) \leq \ell_v \leq \ell$ did not reverse during stage i. This implies that v's height remained unchanged during stage i. The very first reversal after stage i is a reversal of a node in level $\ell - i$, say w. Thus w reverses before v in the execution, though w's height was greater than that of v. This contradicts the way we chose our execution schedule, which mandated that the lowest height node reversed first. ■

Lemma 6.3 *At the end of stage i, the heights of nodes in levels $(\ell - i + 1)$ till ℓ do not depend on the heights of nodes in levels 1 till $(\ell - i - 1)$.*

Proof: Proof by contradiction. Consider two nodes, u and v at levels ℓ_u and ℓ_v respectively. Suppose $1 \leq \ell_u \leq (\ell - i - 1)$ and $(\ell - i + 1) \leq \ell_v \leq \ell$, and at the end of stage i, v's height depended on u's height. Then, there must have been a sequence of reversals $u_1, u_2, \ldots, u_j, v$ such that u_1 was adjacent to u, u_2 adjacent to u_1 and so on and finally u_j adjacent to v. But, this is impossible since no node which was adjacent to u has reversed so far. Thus, at the end of stage i, v's height cannot depend on u's height. ■

Lemma 6.4 *There exists an assignment of heights to nodes such that, for each $i = 1 \ldots \ell$, every node in levels $(\ell - i + 1)$ till ℓ reverses at least once in stage i. This assignment is specific to the function g.*

Proof: We show that there exists an assignment of heights such that for each $i = 1 \ldots \ell$, at the beginning of stage i, the precondition for Lemma 6.2 is satisfied. From Lemma 6.2, it follows that all nodes in levels $\ell - i + 1$ till ℓ reverse at least once in level i.

Base Case: For $i = 1$, the precondition for Lemma 6.2 is true, since we guarantee that the initial heights decrease with increasing levels.

Inductive Case: Suppose the precondition was true for $i = k$. Then, we know from Lemma 6.3 that at the end of stage k, the heights of nodes in levels $(\ell - k + 1)$ till ℓ do not depend on the heights of nodes in levels 1 till $(\ell - k - 1)$. We now assign the heights of every node in level $(\ell - k - 1)$ to be a value greater than the maximum height in levels $\ell - k$ till ℓ. We don't assign specific heights to nodes in levels k' where $k' < (l - k - 1)$ yet, but guarantee that their heights will be greater than the current maximum height in levels $(\ell - k)$ till ℓ. By this assignment, the precondition for Lemma 6.2 is satisfied for stage $i = k + 1$, and the Theorem follows by induction. ■

Theorem 6.1 applies to any graph. Consider the list graph G_1, shown in Figure 3 with n bad nodes. There is one node in each level 1 till $n - 1$. From the above theorem, the lower bound for the worst case number of reversals of any reversal algorithm on the list is $1 + 2 + \ldots n - 1 = \Omega(n^2)$. Thus we have the following corollary.

Corollary 6.5 *There is a graph with an initial state containing n bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ work until stabilization.*

We can derive a similar lower bound on the time needed for stabilization. For this, we use the graph G_4 with n bad nodes, shown in Figure 6. The structure of the graph, and the parameters m_1 and m_2 are the same as defined in Section 5. From Theorem 6.1, we know that each node in layer $m_1 - 1$ of G_4 requires at least $(m_1 - 2)$ reversals before it becomes a good node. Layer $m_1 - 1$, contains m_2 nodes. Therefore, at least $(m_1 - 2) \cdot m_2 = \Omega(n^2)$ reversals are required before these nodes become good nodes. All these reversals have to be performed sequentially, since the nodes of layer $m_1 - 1$ are adjacent, and no two of these nodes can be sinks simultaneously. Thus, we have the following corollary.

Corollary 6.6 *There is a graph with an initial state containing n bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ time until stabilization.*

7 Discussion

We presented a worst-case analysis of link reversal routing algorithms in terms of work and time. We showed that for n bad nodes, the GB full reversal algorithm requires $O(n^2)$ work and time, while the partial reversal algorithm requires $O(n \cdot a^* + n^2)$ work and time. The above bounds are tight in the worst case. Furthermore, we showed that the worst-case work and time of any deterministic algorithm is $\Omega(n^2)$.

Since a^* can grow arbitrarily large, the full reversal algorithm outperforms the partial reversal algorithm in the worst case. It would be interesting to find a variation of the partial reversal algorithm which is as good as full reversal in the worst case. Another research problem is to analyze the average performance of link reversal algorithms.

It would be also interesting to extend our analysis to non-deterministic algorithms. An example of such an algorithm is TORA, in which the height of a sink upon reversal may depend on the current real time. Other classes of algorithms are randomized algorithms, in which the new height of a sink is some randomized function of the neighbors' heights.

Acknowledgments We would like to thank the anonymous referees for their comments.

References

[1] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '98)*, pages 85–97, New York, Oct. 25–30 1998. ACM Press.

[2] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis. An efficient communication strategy for ad-hoc mobile networks. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC '01)*, number 2180 in LNCS, pages 285–199. Springer-Verlag, 2001.

[3] M. S. Corson and A. Ephremides. A distributed routing algorithm for mobile radio networks. In *Proceedings of the IEEE Military Communications Conference (MILCOM '89)*, October 1989.

[4] M. S. Corson and A. Ephremides. A distributed routing algorithm for mobile wireless networks. *ACM/Baltzer Wireless Networks Journal*, 1(1):61–82, February 1995.

[5] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE trans. on commun.*, COM-29:11–18, 1981.

[6] N. Malpani, J. L. Welch, and 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 communication*, Aug. 11, 2000.

[7] V. Park and M. S. Corson. A performance comparison of the temporally-ordered routing algorithm and ideal link-state routing. In *Proceedings of IEEE International Symposium on Systems and Communications*, June 1998.

[8] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *IEEE Infocom '97 - 16th Conference on Computer Communications.* IEEE, 1997.

[9] C. E. Perkins. *Ad Hoc Networking.* Addison Wesley, 2000.

[10] R. Rajaraman. Topology control and routing in ad hoc networks: A survey. *SIGACT News*, June 2002.

[11] R. Samir, C. Robert, Y. Jiangtao, and S. Rimli. Comparative performance evaluation of routing protocols for mobile, ad hoc networks. In *Proceedings of IEEE the Seventh International Conference on Computer Communications and Networks (IC3N '98)*, Oct. 1998.