

Routing without Ordering

Bernadette Charron-Bost
Ecole polytechnique
charron@lix.polytechnique.fr

Antoine Gaillard
Ecole polytechnique
gaillard@lix.polytechnique.fr

Jennifer L. Welch
Texas A&M University
welch@cse.tamu.edu

Josef Widder
Ecole polytechnique / TU Wien
widder@lix.polytechnique.fr

Abstract

We analyze the correctness and complexity of two well-known routing algorithms, introduced by Gafni and Bertsekas (1981): By reversing the directions of some edges, these algorithms transform an arbitrary directed acyclic input graph into an output graph with at least one route from each node to a special destination node (while maintaining acyclicity). The resulting graph can thus be used to route messages in a loop-free manner.

Gafni and Bertsekas implement these routing algorithms by assigning to each node of the graph an unbounded “height” in some total order. The relative order of the heights of two neighboring nodes induces a logical direction on the edge between them; the direction of an edge is reversed by modifying the height of one endpoint.

In this work, we present a novel formalization for these algorithms based only on directed graphs with binary labels for edges. Using this formalization, we define a distributed algorithm for establishing routes in acyclic graphs, and we derive requirements on the input graph for correctness of the algorithm. The algorithms of Gafni and Bertsekas are special cases of our more general algorithm. Moreover, this simple formalization allows us to give an exact complexity analysis. In particular, we provide an expression for the exact number of steps taken by each node during an execution of the algorithm, and prove that this complexity only depends on the input graph.

Research Report 41/2009. Technische Universität Wien, Institut für Technische Informatik.

1 Introduction

In 1981, Gafni and Bertsekas [GB81] introduced an elegant algorithmic technique, called *link reversal*, for routing messages in a network subject to frequent communication topology changes. Numerous papers have applied the concept of link reversal to various applications in dynamic networks, for example routing [PC97], mutual exclusion [Ray89, WWV01, WCM01], leader election [MWV00], distributed queueing [DH98, HTW01, KW04], and resource allocation [CM84, BG89, MMZ93].

In the routing problem under consideration, the network contains a unique destination and a virtual direction is associated with each communication link. The link directions should ensure that every node has a directed path to the destination, i.e., the graph is *destination-oriented*. Messages can then be routed through the network from a node to the destination by following the virtual directions of the links. If, due to the failure or movement of one node and the resulting loss of its incident links, another node no longer has a directed path to the destination, then the network should adjust itself to restore the property. The adjustments consist of nodes *reversing* their incident links. Nodes should be able to determine autonomously, based only on local information, if they need to perform reversals.

Gafni and Bertsekas [GB81] restricted the study to the setting of *acyclic* graphs, and presented two algorithms, each with an abstract description as well as an implementation. In the abstract description of both algorithms, any node (other than the destination) that becomes a sink reverses some of its incident links. The algorithms differ in which incident links are chosen for reversal. In the *full reversal* algorithm, all incident links are reversed, whereas in the *partial reversal* algorithm, only those that have not been reversed since the last time this node was a sink are reversed, roughly speaking. Both algorithms are implemented by assigning a unique *height*, drawn from a totally ordered set, to each node and considering a link to be directed from the endpoint with larger height to that with smaller height. A link reversal is implemented by increasing the height of a node with no outgoing link. The height for the full reversal implementation is an ordered pair consisting of an unbounded counter and the unique identifier of the node. The height for the partial reversal implementation is an ordered triple consisting of two unbounded counters and the unique identifier of the node. A node with no outgoing link applies a function, based on its current height and those of its neighbors, to calculate its new height; for the full reversal implementation, the function simply sets the counter to be larger than the counters of all neighbors, whereas the function for the partial reversal algorithm increases in a smoother mode.

Gafni and Bertsekas proved that the two algorithm implementations are correct, i.e., that they both terminate and when they do, the directions on the links form a destination-oriented acyclic graph. For the proof, they consider a generalization of the heights and the functions that manipulate them, and prove correctness for the generalization.

In this paper, we take a closer look at these algorithms. Our primary goal was to learn if there were alternative ways to implement the abstract descriptions of the full reversal and partial reversal routing algorithms, which we will refer to as *FR* and *PR* respectively. The order-based approach by Gafni and Bertsekas has several advantages, which may account for its popularity. First, it allows the unification of *FR* and *PR* in a very elegant way. As a result, the convergence of the two algorithms is proven in [GB81] by considering only some abstract properties of the heights and the functions that change them. Moreover, by varying the representation of heights as well as the functions, one may expect to get more algorithms than just *FR* and *PR*. Finally, the ordered structure of heights gives the acyclicity of the solution for free.

However, when examining the approach closely, several problems arise. First, the distributed computation of initial heights may be quite problematic. Second, the correctness of the approach relies on the fact that heights are not bounded, which is a drastic limitation of the approach from a practical viewpoint. In addition, the potential flexibility of the approach — resulting from different

types of heights and different ways to manipulate them — does not easily or intuitively provide new algorithms. Moreover, the order-based approach precludes an analysis of the underlying graph algorithms *FR* and *PR*.

In this work, we present a novel formalization for these algorithms based only on directed graphs with binary labels for links. Using this more direct formalization, we define a simple distributed algorithm for establishing routes in acyclic graphs, and we derive requirements on the labeling of the input graph sufficient for correctness of the algorithm. The *FR* and *PR* algorithms of Gafni and Bertsekas are specializations of our general algorithm for some specific initializations of link labels, namely uniform initializations. Interestingly, our algorithm captures more than just the *FR* and *PR* algorithms, by considering non-uniform initializations of link labels. Moreover, this formalization allows us to give a simple proof of correctness with respect to the destination-orientation issue. To the best of our knowledge, this is the first proof that the abstract *PR* algorithm preserves acyclicity, a nontrivial task when the ordered structure of the heights is unavailable.

Surprisingly, there was no methodical study of the complexity of link reversal algorithms until that of [BST03, BT05]. In that work, the authors analyzed the *global work complexity* (total number of reversals performed by all the nodes). Letting n be the number of nodes that initially do not have a path to the destination, they showed that for both the pair and the triple¹ algorithms, the global work complexity is $\Theta(n^2)$.

Our formalization of the general link reversal algorithm allows us to express the *exact* number of steps taken by each node during an execution of the algorithm. The expression depends only on the input graph, and is specialized to simple formulas for full reversal and for partial reversal. In contrast, the work complexity result in [BST03, BT05] is exact only for the pair algorithm (corresponding to full reversal). Having an exact formula facilitates determining the best and worst graph topologies and leads to interesting optimizations, depending on various possible initializations.

The remainder of the paper is organized as follows. In Section 2, we formally define the problem and review the previous solution. In Section 3, we present our solution to the problem and prove its correctness. The work complexity of our solution is analyzed in Section 4. Section 5 draws some conclusions. Most of the proofs are given in the Appendix.

2 The Destination-Orientation problem: background

2.1 Notation and terminology

First, we recall standard terminology in graph theory and introduce some notation. Let G be a connected directed graph with a special node D . The set of nodes other than D is finite and is denoted by V . Graph G is said to be *D(estination)-oriented* if it has the property that there exists at least one path from each node in V to D . Then we consider the following problem \mathcal{P}_D :

\mathcal{P}_D : Transform the directed connected graph G into a D -oriented directed graph by reversing the direction of some links.

For each node v , we denote the set of (incoming and outgoing) neighbors of v in G by N_v . If v has no outgoing link, then v is called a *sink* of G . A *chain* in G is a sequence of nodes v_0, \dots, v_k such that for any i , $0 \leq i \leq k-1$, either (v_i, v_{i+1}) or (v_{i+1}, v_i) is a link of G . A *path* in G is a chain v_0, \dots, v_k such that for any i , $0 \leq i \leq k-1$, (v_i, v_{i+1}) is a link of G . A *circuit* (resp. *cycle*) is a closed chain (resp. a closed path), that is to say a chain (resp. a path) v_0, \dots, v_k with $v_0 = v_k$.

Obviously, if G is D -oriented, then D is the sole sink of G . As shown in [GB81], the converse holds if G is acyclic:

¹assuming certain initial values of the heights

Lemma 2.1. *Let G be a directed acyclic graph with a special node D . The following conditions are equivalent: (i) G is D -oriented, and (ii) D is the sole sink of G .*

In a distributed setting, Lemma 2.1 leads to a promising strategy to solve \mathcal{P}_D : it consists in “fighting” sinks—i.e., changing the direction of links incident to sinks—while maintaining acyclicity. Such a strategy requires us to work under the assumption that a process is associated with each node of the graph, and for each node v in V , the process associated with v can both (a) determine the direction of all the links incident to v , and (b) change the direction of all incoming links incident to v . In this case, any process associated with a sink can locally detect that the graph is *not* destination oriented, and then it reverses some of its incoming links in order to be no more a sink. Then it remains to verify that this scheme maintains acyclicity and terminates.

In the sequel, we shall work within the context of (a) and (b), and we shall identify a node v with the process associated with v . Based on (a) and (b), we shall study various distributed algorithms, according to the link reversal strategy used by sink nodes. We shall consider the *asynchronous executions* of these link reversal algorithms: if v is a sink at some moment in an execution, then v is not required to execute a reversal immediately, but it has only to do so eventually². A link reversal algorithm thereby admits several asynchronous executions depending on the scheduler.

2.2 The order-based approach by Gafni and Bertsekas

Following the above strategy, Gafni and Bertsekas [GB81] provided two distributed algorithms, namely *Full Reversal (FR)* and *Partial Reversal (PR)*:

FR: At each iteration, some set of sinks that does not include D take steps. Each sink that takes a step reverses the direction of all its incident links.

PR: Each node maintains a list of its incident links that have just been reversed. At each iteration, some set of sinks that does not include D take steps. Each sink that takes a step reverses the directions of all its incident links that do not appear on its list, and then empties its list. If no such incident link occurs (i.e., its list is “full”), it reverses the directions of all its incident links, and then empties its list.

It is easy to see that in every step of *FR*, acyclicity is maintained. In sharp contrast, an elementary proof that *PR* maintains acyclicity appears to be extremely hard to conduct and to the best of our knowledge no such proof exists yet. Indeed, the argument for proving acyclicity given in [GB81] is not direct: Gafni and Bertsekas first showed that *FR* and *PR* are specializations of a general solution to an order-based problem that is dual to \mathcal{P}_D , and then trivially derived acyclicity from the antisymmetry property of any ordering.

In more detail, their approach consists in embedding the directed graph into a total order. Node v maintains a variable, denoted \mathbf{h}_v (for height), whose values are drawn from a totally ordered set $(A, <)$. The ordering between heights determines the *directions* of links by the basic rule: *if $\mathbf{h}_v > \mathbf{h}_w$, then the link connecting v and w is directed from v to w* . Hence, sinks correspond to local minima with respect to heights, and so each node $v \in V$ is assigned to increase its height, when the latter is a local minimum, according to some function g_v . Under the assumption that each node can read the heights of all its neighbors, this naturally leads to a general distributed algorithm \mathcal{IH} (for Increasing Heights).

Gafni and Bertsekas then supposed A to be unbounded and placed a condition on the asymptotic behavior of each g_v . They proved that under these conditions, the \mathcal{IH} algorithm solves \mathcal{P}_D , and that the resulting D -oriented graphs only depend on the input graphs.

²Note that the directions of all the links incident to some sink node v remain stable until v makes a reversal.

If we assume the existence of distinct identifiers for nodes, then FR and PR can be viewed as special cases of the \mathcal{IH} algorithm. Indeed, FR can be implemented by representing the height of v as a pair (α_v, i_v) with α_v being an integer and i_v being the unique identifier of node v . The relation $<$ is the lexicographical order, and the g_v 's correspond to transforming local minima into local maxima. Similarly, PR can be implemented by triples (α_v, β_v, i_v) where initially, $\alpha_v = 0$ for any node v , and the function g_v transforms any local minimum into an intermediate value (not necessarily a local maximum).

As explained in the Introduction, the order-based approach by Gafni and Bertsekas has several advantages. First, it allows the unification of FR and PR in a very elegant way. As a result, the convergence of the two algorithms is proven in [GB81] by considering only the abstract properties of the heights and the functions g_v . Moreover, the ordered structure of heights, i.e., the total ordering in A , gives the acyclicity of the solution for free. Finally, by varying the representation of heights as well as the increasing functions, one may expect to get more distributed solutions than just FR and PR to the destination-orientation problem.

However, when examining the approach closely, several problems arise. Firstly, a distributed initialization of the heights such that they represent the directed input graph is not straightforward. In the particular case of the FR and PR implementations described above, this is equivalent to assigning distinct identities to processes, that is to say to solve the *naming* problem. Secondly, the correctness of the approach relies on the fact that heights are *a priori* not bounded: given an implementation of \mathcal{IH} with some particular ordered set A , a process cannot determine in advance an upper bound on the heights that it will use. From a practical viewpoint, this is a drastic limitation of the approach. Thirdly, it appears that the flexibility of the approach — given the various possible choices for the ordered set A and the functions g_v — does not provide solutions different in essence from FR and PR . To the best of our knowledge, the only implementation different from FR and PR has been given by Busch *et al.* in [BST03, BT05], but it is basically equivalent to PR after some preliminary phase. So the approach makes use of a rather heavy formalism without exploiting its expressiveness. Moreover, the order-based approach gives no new insights into the very nature of \mathcal{P}_D , and precludes an analysis of the underlying graph algorithms FR and PR .

3 The Link Reversal Algorithm

In this section, we present a distributed graph algorithm, which we call the *Link Reversal* algorithm (or \mathcal{LR} for short), that encompasses both FR and PR . To explain our graph-based approach, let us first examine more closely how PR runs. Denote node v 's list by $v.list$. One observes that for any two neighbors v and w , v is in $w.list$ only if the link (v, w) is directed from v to w . Hence, one cannot have $v \in w.list$ and $w \in v.list$ at the same time. Therefore, for each directed link from v to w , there are only two possible cases: either $v \notin w.list$ and $w \notin v.list$, or $v \in w.list$ and $w \notin v.list$. Our basic idea is to code these two cases with labels 0 and 1 on the link from v to w , accordingly. We thus propose an algorithm which works on directed graphs with *binary* labels for *links*, contrary to the \mathcal{IH} algorithm which assigns *unbounded* labels (called heights) to *nodes*.

More generally, let $\mu : E \rightarrow \{0, 1\}$ be a binary labeling of links in G , and consider the resulting link-labeled directed graph. If $\mu(e) = 1$, we say e is marked; otherwise it is said to be unmarked. In all the following, the superscript “†” is used to denote directed graph structures when they are equipped with binary link labels.

For each vertex v , let In_v and Out_v denote the sets of incoming and outgoing links incident to v . In In_v , we distinguish the subset of marked incoming links In_v^1 from the subset of unmarked incoming links In_v^0 . Then, the \mathcal{LR} algorithm is described with the two following transition rules executed by any *sink* node v other than D :

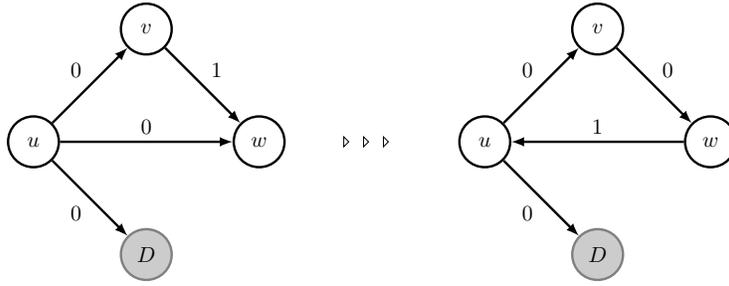


Figure 1: An example execution of \mathcal{LR} that is different from executions of \mathcal{IH} .

LR1 If $In_v^0 \neq \emptyset$, then v reverses the directions of all the links in In_v^0 , and marks them; moreover v unmarks all the links in In_v^1 .

LR2 Otherwise (i.e., if $In_v^0 = \emptyset$), v reverses the direction of all its incident links and the labels remain unchanged.

As explained in the above section, we consider asynchronous executions of \mathcal{LR} : if v is a sink at some moment in an execution, then v is not required to execute a reversal at the very next step, but it has to do so eventually. Given an initial labeled directed graph G^\dagger , there are thereby several possible executions for \mathcal{LR} .

Observe that if all labels are initially equal to 1, then all links remain marked during the whole execution, and only rule LR2 is executed. The \mathcal{LR} algorithm thus coincides with FR from the operational point of view. Besides, if initially all labels are 0, the above discussion about the runs of PR indicates that \mathcal{LR} executes as PR (a formal proof is given in the extended version). As with the formalization of [GB81], our approach thus unifies FR and PR : in [GB81], one derives different implementations by varying the infinite total ordering $(A, <)$ and the update functions g_v , whereas our unification consists in varying only the initial labeling. Interestingly, our approach also allows us to express more implementations than \mathcal{IH} does. As an example, consider the input graph given in Figure 1. The \mathcal{LR} algorithm converges to a D -oriented graph G_* after only one reversal executed by node w . Obviously, such a one-step run does not correspond to any implementation of the \mathcal{IH} algorithm since the resulting directed graph G_* contains a cycle. Hence, despite the use of a more sophisticated combinatorial structure – namely an unbounded total ordering – and of more elaborate transition rules, it turns out that the \mathcal{IH} algorithm is not more expressive than \mathcal{LR} .

3.1 Convergence and delay-insensitivity

We are going to prove that the \mathcal{LR} algorithm converges, i.e., from any given directed connected graph with link labels, every execution of \mathcal{LR} is finite. Moreover, we shall show that the algorithm is *delay-insensitive* in the sense that the directed graph generated by the algorithm only depends on the input graph, and not on the scheduler. Note that by definition of the \mathcal{LR} rules, the latter graph has no sink node, except possibly D .

Theorem 3.1. *Given a directed connected input graph G_0^\dagger with binary link labels, in every execution of \mathcal{LR} , each node makes a finite number of steps. Moreover, the directed graph generated by \mathcal{LR} only depends on G_0^\dagger .*

3.2 The acyclicity issue

We have thus shown that starting with a finite connected input graph, whether acyclic or not, the algorithm converges, i.e., reaches a state where there is no sink node different from D . However, “fighting sinks” is just one part of the required solution, and it remains to study whether acyclicity can be maintained by \mathcal{LR} . For that, our strategy will consist in reducing certain properties of the algorithm to invariants of the graph, or more precisely to some of its subgraphs as circuits.

First, we introduce some additional notation. As no confusion can arise³, we denote by $v.G^\dagger$ the graph obtained after the node v (and only v) in G^\dagger takes a step of \mathcal{LR} . For any circuit C^\dagger of G^\dagger , $v.C^\dagger$ denotes the corresponding circuit in $v.G^\dagger$ just after v 's reversal. Note that $v.G^\dagger$ and $v.C^\dagger$ have the same underlying non-directed graph (called support) as G^\dagger and C^\dagger , respectively.

For each circuit C^\dagger , we fix an arbitrary orientation of its support. Let $\ell^1(C^\dagger)$ be the number of marked links in C^\dagger oriented according to the orientation and $r^1(C^\dagger)$ be the number of marked links in C^\dagger oriented against the orientation. Let $s^0(C^\dagger)$ be the number of nodes in C^\dagger with two distinct incoming unmarked links *relative to* C^\dagger . Then we define the following two measures λ and ϱ :

$$\lambda(C^\dagger) = \ell^1(C^\dagger) + s^0(C^\dagger) \text{ and } \varrho(C^\dagger) = r^1(C^\dagger) + s^0(C^\dagger).$$

Proposition 3.2. *Let C^\dagger be any circuit in G^\dagger . For any node v , $\lambda(v.C^\dagger) = \lambda(C^\dagger)$ and $\varrho(v.C^\dagger) = \varrho(C^\dagger)$.*

Proposition 3.3. *If C^\dagger is a circuit such that*

$$\lambda(C^\dagger) \cdot \varrho(C^\dagger) > 0, \tag{AC}$$

then C^\dagger is not a cycle.

Proof. Observe that if C^\dagger is a cycle, then $s^0(C^\dagger) = 0$, and either $\ell^1(C^\dagger) = 0$ or $r^1(C^\dagger) = 0$. It follows that condition (AC) is sufficient for guaranteeing C^\dagger is not a cycle. \square

From Propositions 3.2 and 3.3, we immediately derive the following acyclicity result:

Theorem 3.4. *If the input graph G_0 is equipped with an initial link labeling such that every circuit in the resulting directed labeled graph G_0^\dagger satisfies condition (AC), then all the directed graphs generated in any execution of \mathcal{LR} are acyclic.*

Combining Theorems 3.1 and 3.4, we get the following corollary:

Corollary 3.5. *Let G_0 be any acyclic connected directed graph. If the initial link labeling is such that all the circuits in the resulting labeled graph G_0^\dagger satisfy (AC), then \mathcal{LR} solves \mathcal{P}_D for G_0 .*

In the case of *FR*, all labels are initially equal to 1, which gives $\ell^1(C^\dagger) \neq 0$ and $r^1(C^\dagger) \neq 0$ for any circuit C^\dagger that is not a cycle. In contrast, $s^0(C^\dagger) \neq 0$ is trivially guaranteed with the uniform null labeling (and so with *PR*) when C^\dagger is not a cycle.

More generally, condition (AC) naturally leads to the labeling policy where each node v is allowed to choose $\mu_v \in \{0, 1\}$ for the initial labels of all its *incoming* links. With such a policy, condition (AC) is ensured for C^\dagger whenever C^\dagger is not a cycle. *FR* and *PR* thus correspond to the uniform choices $\mu_v = 1$ and $\mu_v = 0$, respectively. Note that the above policy can be easily implemented with a distributed scheme, and potentially it leads to solutions other than *FR* and *PR* that we have analyzed in a companion paper [CWW09].

³since each individual node v has a deterministic behavior in \mathcal{LR}

4 Work complexity

We now analyze the work complexity of the \mathcal{LR} algorithm. We define the *work complexity of a node* in an execution to be the number of reversals performed by that node, and the *global work complexity* to be the sum, over all nodes, of the work complexity of each node. By the results shown in the previous section, the work complexity depends only on the initial graph.

We first extend the definitions for the circuits introduced in Section 3.2 to chains. Since the input graph is connected and \mathcal{LR} maintains connectivity, for any node v , there is a chain connecting D and v with a stable support during any execution of \mathcal{LR} . Due to the statement of the destination-orientation problem, such a chain is naturally oriented from v to D , and is called a *D-chain from v* . If each link along a D -chain is directed towards D , then the D -chain is said to be a *D-path*. Further we will denote by $\mathcal{C}(v, G^\dagger)$ the set of all D -chains from v with the link directions and the link labels inherited from G^\dagger .

For any D -chain C^\dagger , we define the *residue* $Res(C^\dagger)$ to be 1 if the first link (with respect to the natural orientation towards D) in C^\dagger is unmarked and against the orientation, and 0 otherwise. As for circuits, we consider the following two quantities $\lambda(C^\dagger) = \ell^1(C^\dagger) + s^0(C^\dagger)$ and $\varrho(C^\dagger) = r^1(C^\dagger) + s^0(C^\dagger)$, and we define two additional quantities

$$\delta(C^\dagger) = \lambda(C^\dagger) - \varrho(C^\dagger) \text{ and } \gamma(C^\dagger) = \lambda(C^\dagger) + \varrho(C^\dagger) + Res(C^\dagger).$$

From the very definitions of $\delta(C^\dagger)$ and $\gamma(C^\dagger)$, we immediately derive the following proposition:

Proposition 4.1. *For any D -chain C^\dagger , we have $\delta(C^\dagger) \leq \gamma(C^\dagger)$. Moreover, if C^\dagger is a D -path, then $\delta(C^\dagger) = \gamma(C^\dagger)$.*

The latter proposition naturally leads us to consider the nonnegative quantity:

$$\omega(C^\dagger) = \gamma(C^\dagger) - \delta(C^\dagger),$$

Now we study how $\gamma(C^\dagger)$ and $\delta(C^\dagger)$ evolve along with executing \mathcal{LR} .

Proposition 4.2. *For any node v and any D -chain C^\dagger , we have $\gamma(v.C^\dagger) = \gamma(C^\dagger)$.*

Since $\gamma(C^\dagger)$ never changes, the key point now is to show that $\delta(C^\dagger)$ regularly increases during any execution of \mathcal{LR} . To do that, we are led to partition the set of nodes V into three disjoint subsets: $\mathcal{S}(G^\dagger)$ is the set of nodes all of whose incident links are unmarked and incoming, $\mathcal{N}(G^\dagger)$ is the set of nodes with no unmarked incoming link, and $\mathcal{O}(G^\dagger)$ consists of the remaining nodes.

Proposition 4.3. *Let v be a sink node and C^\dagger be any D -chain in G^\dagger . If v is not the first node in C^\dagger , then $\delta(v.C^\dagger) = \delta(C^\dagger)$. Otherwise, C^\dagger is in $\mathcal{C}(v, G^\dagger)$, and $\delta(v.C^\dagger)$ is given by the following rule: if v is in $\mathcal{N}(G^\dagger)$ then $\delta(v.C^\dagger) = \delta(C^\dagger) + 2$; else (i.e., $v \in \mathcal{S}(G^\dagger) \cup \mathcal{O}(G^\dagger)$), $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$.*

Now we study how this node partitioning evolves during the execution of \mathcal{LR} .

Proposition 4.4. *If v is a sink node in G^\dagger , then*

$$\mathcal{S}(v.G^\dagger) = \mathcal{S}(G^\dagger) \setminus \{v\}, \mathcal{N}(v.G^\dagger) = \mathcal{N}(G^\dagger) \cup \left(\mathcal{S}(G^\dagger) \cap \{v\} \right), \mathcal{O}(v.G^\dagger) = \mathcal{O}(G^\dagger).$$

Note that both Propositions 4.3 and 4.4 trivially extend to the case when several sink nodes make simultaneous reversals: if we extend the notation $v.G^\dagger$ to $S.G^\dagger$, where S is any subset of sinks in G^\dagger other than D , then we have

$$\mathcal{S}(S.G^\dagger) = \mathcal{S}(G^\dagger) \setminus S, \mathcal{N}(S.G^\dagger) = \mathcal{N}(G^\dagger) \cup \left(\mathcal{S}(G^\dagger) \cap S \right), \text{ and } \mathcal{O}(S.G^\dagger) = \mathcal{O}(G^\dagger).$$

We need one more definition before stating our main result. For each node v , we consider the set of D -chains from v and define the quantity:

$$\omega(v, G^\dagger) = \min_{C^\dagger \in \mathcal{C}(v, G^\dagger)} \left(\omega(C^\dagger) \right).$$

As an immediate consequence of Propositions 4.2 and 4.3, we derive the following lemma:

Lemma 4.5. *The D -chains that achieve the minimum value of $\omega(C^\dagger)$ are the same all along the execution.*

We fix an arbitrary execution of the \mathcal{LR} algorithm, and we denote by $G_0^\dagger, \dots, G_i^\dagger, \dots$ the sequence of directed connected graphs with binary link labels that is generated in the execution. By Corollary 3.5, this sequence is finite and the last graph, denoted G_k^\dagger , is D -oriented.

Theorem 4.6. *In any execution of \mathcal{LR} starting from G_0^\dagger whose circuits all satisfy (AC), the number of reversals made by any node v is $\omega(v, G_0^\dagger)/2 + 1/2$, $\omega(v, G_0^\dagger)/2$, or $\omega(v, G_0^\dagger)$ if $v \in \mathcal{S}(G_0^\dagger)$, $v \in \mathcal{N}(G_0^\dagger)$, or $v \in \mathcal{O}(G_0^\dagger)$, accordingly.*

Proof. Consider the finite sequence of nonnegative integers $\omega(v, G_0^\dagger), \dots, \omega(v, G_k^\dagger)$, and the subsequence $\omega(v, G_0^\dagger), \dots, \omega(v, G_{k_v}^\dagger)$, obtained when considering only the steps in which v makes a reversal. By Lemma 4.5 and Propositions 4.2 and 4.3, it follows that the sequence $\omega(v, G_0^\dagger), \dots, \omega(v, G_k^\dagger)$ is non-increasing, and the subsequence $\omega(v, G_0^\dagger), \dots, \omega(v, G_{k_v}^\dagger)$ is decreasing. More precisely, we deduce from Propositions 4.2, 4.3 and 4.4 that the subsequence decreases regularly: it decreases either by 1 the first time and then by 2 each later time if $v \in \mathcal{S}(G_0^\dagger)$, by 2 each time if $v \in \mathcal{N}(G_0^\dagger)$, and otherwise ($v \in \mathcal{O}(G_0^\dagger)$) by 1 each time. Moreover, the first value of this sequence is $\omega(v, G_0^\dagger)$. By Proposition 4.1, $\omega(v, G_k^\dagger) = 0$ since G_k^\dagger is D -oriented. The subsequence thus decreases from $\omega(v, G_0^\dagger)$ to 0, and then we deduce the number of reversals made by v . \square

As an immediate consequence of Theorem 4.6, the number of reversals by v depends solely on the input graph and the initial labels and does *not* depend on the order in which nodes take steps.

4.1 Work complexity of full and partial reversal

From Theorem 4.6, we derive the work complexity of \mathcal{LR} in the two particular cases of uniform link labelings, i.e., for FR and PR . That will allow us to compare these two link reversal strategies, and more generally the various possible initializations for link labels.

Since FR corresponds to the \mathcal{LR} algorithm with all initial link labels equal to 1 and PR corresponds to the \mathcal{LR} algorithm with all initial link labels equal to 0, we deduce the exact formulas for the work complexities of FR and PR . For that, we specialize the notation introduced above for the particular cases of 0 and 1 uniform labelings. Given a directed acyclic graph G , for any node v we denote the set of all D -chains from v by $\mathcal{C}(v, G)$; for any $C \in \mathcal{C}(v, G)$, let $r(C)$ be the number of links in C that are directed away from D and $s(C)$ be the number of nodes in C which have two distinct incoming links relative to C . We also define $Res(C)$, the residue of C , to be 1 if and only if the first link of C is directed toward v . Note that in the case of 1 uniform labeling (FR), both \mathcal{S} and \mathcal{O} are empty, and so \mathcal{N} contains all the nodes other than D . As for 0 uniform labeling (PR), \mathcal{S} and \mathcal{N} consist of all the sinks and all the sources, respectively.

Corollary 4.7. *1. The number of reversals made by node v in FR is*

$$\min_{C \in \mathcal{C}(v, G)} (r(C)).$$

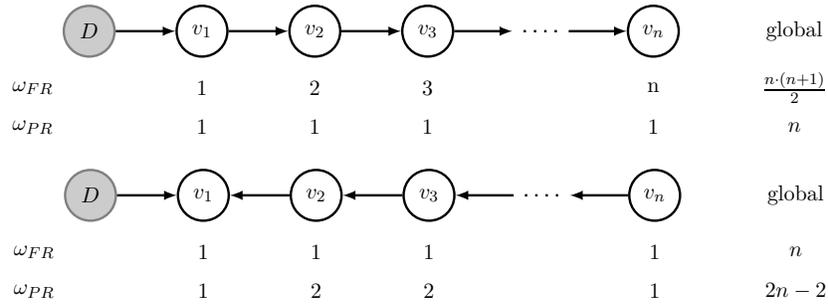


Figure 2: Example chains showing that neither FR nor PR performs better in general.

2. The number of reversals made by node v in PR is

$$\min_{C \in \mathcal{C}(v, G)} (s(C) + Res(C)), \text{ if } v \text{ is a sink or a source in } G$$

$$\min_{C \in \mathcal{C}(v, G)} (2s(C) + Res(C)), \text{ otherwise.}$$

As a consequence of this corollary, we now understand what properties of the input graphs generate steps for FR and for PR . In particular, this leads us to design the two (directed) chains in Figure 2 for which there is a large discrepancy between the global work complexities of FR and PR , respectively. Indeed, for the upper chain, the number of steps taken by FR is $O(n^2)$ while it is n for PR . In contrast, the global work complexity of FR for the lower chain is n , while that of PR is $2n - 2$; i.e., PR is worse by about a factor of 2 for this graph.

Based on the formulae in Theorem 4.6, we have analyzed in [CWW09] the work complexity for the edge labeling policy where each node v is allowed to choose $\mu_v \in \{0, 1\}$ for the initial labels of all its incoming links. For this policy, we studied whether there is a globally optimum edge labeling strategy which minimizes the total number of link reversals. Interestingly, we have characterized the best strategies for each node v to minimize its work, and have shown that a selfish strategy for v in general increases the work done by other nodes.

4.2 Comparison with previous work

Busch *et al.* [BST03, BT05] initiated the study of the work complexity of the link reversal algorithms presented in [GB81]. They analyzed the total number of reversals as a function of n , the number of nodes that have no directed path to D in the initial graph (called *bad* nodes). In this subsection we compare our results to theirs.

For the pair-based implementation of full reversal, they determined the exact number of reversals performed by any node. They partitioned the set of initially bad nodes into “layers” and proved that if a node belongs to the i -th layer, then it performs exactly i reversals. Their definition of layer is such that a node v is in the i -th layer if and only if our formula for v ’s work equals i . Thus our results are the same as those in [BST03, BT05] for the work complexity of full reversal.

In both approaches, each of the n nodes can be seen to perform at most n reversals: there are at most n layers in their approach, and there are at most n links in a path from a node to the destination in our approach. Thus the global work complexity is at most $O(n^2)$. As in [BST03, BT05], we derive from our formula a particular graph in which $\Omega(n^2)$ reversals are done: the graph consists of a chain of $n + 1$ nodes in which D is at one end of the chain and all links are directed away from D (cf. Figure 2).

For the triple-based implementation of partial reversal, Busch *et al.* calculated upper and lower bounds on the number of reversals performed by any node. Their bounds are not tight, and, in addition, the lower bounds are only defined for certain graphs, not all graphs. In more detail,

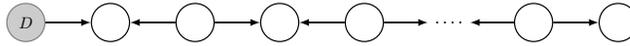


Figure 3: A worst case graph for global work complexity of PR .

Busch *et al.* partitioned the initially bad nodes into “levels” (not to be confused with layers) and showed that the level index of a node is an upper bound on the number of reversals performed by the node. It can be shown that for nodes that are in \mathcal{S} or \mathcal{N} initially, their upper bound is at least twice as large as our (exact) expression.

For the lower bound, Busch *et al.* partitioned the initially bad nodes into “layers” (not the same notion as for the full reversal analysis) and showed that the number of reversals performed by a node is at least half the index of the layer to which the node belongs. It can be shown that their lower bound is at most half as large as our work expression for nodes in \mathcal{N} or \mathcal{S} , and is at most one-fourth as large as the one for nodes in \mathcal{O} . Not all graphs can be decomposed into layers and thus their lower bounds are only relevant for nodes in a subset of all graphs. In contrast, our results give an exact formula for the work complexity of any node in any graph.

Moreover, Corollary 4.7 establishes a formal understanding of the graphs used by Busch *et al.* to prove that their bounds are asymptotically tight. In particular, the worst case graph for the global work complexity of PR given in Figure 3 is obtained by maximizing s . Thus, both approaches establish a global work complexity for PR of $\Theta(n^2)$.

Finally, the results in [BST03, BT05] only address slight variations of the pair and triple algorithms, whereas our results hold for a larger class of solutions, namely all those that can be encompassed by the \mathcal{LR} algorithm. Hence, our approach gives results for work complexity that are more general, in that they apply to a larger class of solutions.

5 Conclusions

In this paper, we have taken a fresh approach to specifying and analyzing the link reversal algorithms introduced by Gafni and Bertsekas [GB81]. We have discovered a generalization of these algorithms, which allows simple and bounded implementations of the full and partial reversal algorithms, as well as allowing new implementations. For our general solution, we have established an exact expression for the work complexity of each node. These complexity results improve upon those known previously [BST03, BT05].

A natural topic of study is the *time complexity* of link reversal algorithms, which can be defined as the number of rounds until termination, where one or more sinks take steps in parallel at each round. The global work complexity gives the time complexity of sequential executions in which only one node takes a step in each round. A more interesting question is how many rounds are required until termination, where in each round *every* sink node takes a step. Busch *et al.* [BST03, BT05] provided asymptotically tight lower bounds on the time complexity for the pair and triple algorithms of [GB81] in the latter case. For each algorithm, they described a particular graph on which the relevant algorithm requires $\Omega(n^2)$ rounds, by using the work lower bounds and the fact that nodes that are neighbors cannot be sinks simultaneously. However, the results in [BST03, BT05] give no insight regarding the number of rounds needed for an arbitrary graph, or regarding the number of rounds until a particular node has finished all its reversals. An interesting open question is whether, using our approach, one can derive a general and exact expression for the number of rounds until termination depending on the input graph.

References

- [BG89] V. C. Barbosa and E. Gafni. Concurrency in heavily loaded neighborhood constrained systems. *ACM Transactions on Programming Languages and Systems*, 11(4):562–584, 1989.
- [BST03] C. Busch, S. Surapaneni, and S. Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 210–219, 2003.
- [BT05] C. Busch and S. Tirthapura. Analysis of link reversal routing algorithms. *SIAM Journal on Computing*, 35(2):305–326, 2005.
- [CM84] K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, 1984.
- [CWW09] B. Charron-Bost, J. L. Welch, and J. Widder. Gambling with link reversal. Unpublished manuscript, February 2009.
- [DH98] M. J. Demmer and M. Herlihy. The arrow distributed directory protocol. In *Proceedings of the 12th International Symposium on Distributed Computing (DISC)*, pages 119–133, 1998.
- [GB81] E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, C-29(1):11–18, 1981.
- [HTW01] M. Herlihy, S. Tirthapura, and R. Wattenhofer. Competitive concurrent distributed queueing. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 127–133, 2001.
- [KW04] F. Kuhn and R. Wattenhofer. Dynamic analysis of the arrow distributed protocol. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 294–301, 2004.
- [MMZ93] Y. Malka, S. Moran, and S. Zaks. A lower bound on the period length of a distributed scheduler. *Algorithmica*, 10:383–398, 1993.
- [MWV00] N. Malpani, J. 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 Communications (DIAL M)*, pages 966–103, 2000.
- [PC97] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the 16th IEEE Conference on Computer Communications (INFOCOM)*, pages 1405–1413, 1997.
- [Ray89] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.
- [WCM01] J. Walter, G. Cao, and M. Mohanty. A k-mutual exclusion algorithm for wireless ad hoc networks. In *Proceedings of the ACM Workshop on Principles of Mobile Computing (POMC)*, 2001.
- [WWV01] J. Walter, J. L. Welch, and N. Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7(6):585–600, 2001.

Appendix

A Proofs for Section 3

Theorem 3.1. *Given a directed connected input graph G_0^\dagger with binary link labels, in every execution of \mathcal{LR} , each node makes a finite number of steps. Moreover, the directed graph generated by \mathcal{LR} only depends on G_0^\dagger .*

Proof. Consider any execution of \mathcal{LR} , and v any node other than D . Let $LR_i(v)$ denote the i th step of node v in this execution. We break the proof into the three following lemmas.

Lemma A.1. *Each neighbor of D makes at most two link reversals.*

Proof. Let v be any neighbor of D . From the \mathcal{LR} rules, after at most two steps of node v , the link between v and D is outgoing from v . Since D takes no step in \mathcal{LR} , v will never be a sink again, and thus v takes at most two steps. \square

Lemma A.2. *Each neighbor of v takes at least one step between $LR_i(v)$ and $LR_{i+2}(v)$.*

Proof. There are two cases to consider.

1. Node v executes LR2 at $LR_i(v)$, and thus reverses all its incident links. The next time v reverses its links, it must be a sink again, i.e., the directions of all the links incident to v must have been changed after $LR_i(v)$. As nodes reverse only incident links, all neighbors of v must have taken at least one step before $LR_{i+1}(v)$, and *a fortiori* before $LR_{i+2}(v)$.
2. Node v executes LR1 at $LR_i(v)$, and thus reverses and marks some of its incident links and unmarks the others. Let N_r be the set of v 's neighbors to which the links have been reversed at $LR_i(v)$ and N_s be set of all the other neighbors of v . By LR1, we know that all the links connecting v to the nodes in N_s have been unmarked at $LR_i(v)$. The next time v reverses its links, it is necessarily a sink again, i.e., the directions of all the links reversed at $LR_i(v)$ have been changed after $LR_i(v)$. Consequently, all nodes in N_s must have taken at least one step before $LR_{i+1}(v)$. There are now two subcases:
 - (a) $N_s = \emptyset$. In this case, all of v 's neighbors made a step before $LR_{i+1}(v)$, and we are done.
 - (b) $N_s \neq \emptyset$. At $LR_{i+1}(v)$, v again executes LR1 and thus reverses all links which connect v to nodes in N_s while it unmarks all the links which connect v to nodes in N_r . By the same reasoning as before, all nodes in N_s make a step before $LR_{i+2}(v)$, and thus all neighbors of v make a step between $LR_i(v)$ and $LR_{i+2}(v)$.

\square

For convergence, we proceed by contradiction, and we assume that there exists a node v which takes an infinite number of steps. Since the input graph is connected, there is a chain relating D to v . Lemma A.2 implies that each node in this chain makes an infinite number of link reversals. In particular, the neighbor of D in the chain takes an infinite number of steps, which contradicts Lemma A.1.

As for delay-insensitivity, it is a straightforward consequence of the following commutativity lemma.

Lemma A.3. *Let v and w be two distinct sink nodes of G^\dagger , i.e., both v and w can make a reversal. Let G_v^\dagger and G_w^\dagger denote the resulting graphs just after the reversals by v and w , respectively. Then w can still make a reversal in G_v^\dagger , and v can make a reversal in G_w^\dagger , and both result in the same directed graph with the same link labels.*

Proof. The result follows at once from the fact that since v and w are two distinct sinks, they are not neighboring nodes. \square

\square

Proposition 3.2. *Let C^\dagger be any circuit in G^\dagger . For any node v , $\lambda(v.C^\dagger) = \lambda(C^\dagger)$ and $\varrho(v.C^\dagger) = \varrho(C^\dagger)$.*

Proof. If v is not a node of C^\dagger , then C^\dagger remains unchanged, i.e., $v.C^\dagger = C^\dagger$, and the result immediately follows. Otherwise, v is a sink node of C^\dagger , and there are three cases to consider depending on the labels of the two links incident to v in C^\dagger .

1. None of the two links is marked. Then v executes LR1, and both reverses and marks the two links. Hence,

$$\ell^1(v.C^\dagger) = \ell^1(C^\dagger) + 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger) + 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger) - 1.$$

2. The two links are marked. We then consider two subcases:

- (a) Node v executes LR1 and hence does not reverse either of them and unmarks both of them. In this case, we have:

$$\ell^1(v.C^\dagger) = \ell^1(C^\dagger) - 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger) - 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger) + 1.$$

- (b) Node v executes LR2, and so reverses and marks both of them. We then have

$$\ell^1(v.C^\dagger) = \ell^1(C^\dagger) - 1 + 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger) - 1 + 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger).$$

3. If only one of the two incoming links of v that belong to C^\dagger is marked then v executes LR1 and thus unmarks the marked link and reverses and marks the other one. We thus have

$$\ell^1(v.C^\dagger) = \ell^1(C^\dagger), \quad r^1(v.C^\dagger) = r^1(C^\dagger), \quad s^0(v.C^\dagger) = s^0(C^\dagger).$$

\square

B Proofs for Section 4

Proposition 4.2. *For any node v and any D -chain C^\dagger , we have $\gamma(v.C^\dagger) = \gamma(C^\dagger)$.*

Proof. Let w and e be the first node and the first link in C^\dagger , respectively. By definition of the \mathcal{LR} rules, v is a sink node. There are three possible cases.

1. Node v does not belong to C^\dagger . Then $v.C^\dagger = C^\dagger$, and thus $\gamma(v.C^\dagger) = \gamma(C^\dagger)$ trivially holds.

2. Node v belongs to C^\dagger , and $v \neq w$. The same proof as for Proposition 3.2 shows that in this case, $\lambda(v.C^\dagger) = \lambda(C^\dagger)$ and $\varrho(v.C^\dagger) = \varrho(C^\dagger)$. Consequently, it remains to show $Res(v.C^\dagger) = Res(C^\dagger)$. If v is not a neighbor of w , then the result follows immediately from the fact that in \mathcal{LR} , nodes change the state of incident links only. Otherwise, v is a neighbor of w , and the link e between v and w is incoming to v since v is a sink node. Hence, $Res(C^\dagger) = 0$. Depending on the label of e in C^\dagger and on whether LR1 or LR2 is executed, the link e in $v.C^\dagger$ is either unmarked and incoming to v , or it is outgoing and marked. In both cases we have $Res(v.C^\dagger) = 0$ and thus $Res(v.C^\dagger) = Res(C^\dagger)$ as needed.

3. Node v is in C^\dagger and $v = w$. There are three possible subcases:

(a) The link e is unmarked in C^\dagger . Node v executes LR1, and so reverses and marks e . It follows that

$$Res(v.C^\dagger) = Res(C^\dagger) - 1, \quad \ell^1(v.C^\dagger) = \ell^1(C^\dagger) + 1, \quad r^1(v.C^\dagger) = r^1(C^\dagger).$$

As e is marked in $v.C^\dagger$, its reversal does not modify the number s^0 , i.e., $s^0(v.C^\dagger) = s^0(C^\dagger)$.

(b) The link e is marked in C^\dagger and v executes LR1. According to this rule, v only unmarks e . In this case, $Res(v.C^\dagger) = Res(C^\dagger) + 1$. Moreover, we have:

$$\ell^1(v.C^\dagger) = \ell^1(C^\dagger), \quad r^1(v.C^\dagger) = r^1(C^\dagger) - 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger).$$

(c) The link e is marked in C^\dagger and v executes LR2. Node v thus reverses and marks e , and so $Res(v.C^\dagger) = Res(C^\dagger)$. Moreover,

$$\ell^1(v.C^\dagger) = \ell^1(C^\dagger) + 1, \quad r^1(v.C^\dagger) = \ell^1(C^\dagger) - 1, \quad s^0(v.C^\dagger) = s^0(C^\dagger).$$

In all cases we conclude that $\gamma(v.C^\dagger) = \gamma(C^\dagger)$.

□

Proposition 4.3. *Let v be a sink node and C^\dagger be any D -chain in G^\dagger . If v is not the first node in C^\dagger , then $\delta(v.C^\dagger) = \delta(C^\dagger)$. Otherwise, C^\dagger is in $\mathcal{C}(v, G^\dagger)$, and $\delta(v.C^\dagger)$ is given by the following rule: if v is in $\mathcal{N}(G^\dagger)$ then $\delta(v.C^\dagger) = \delta(C^\dagger) + 2$; else (i.e., $v \in \mathcal{S}(G^\dagger) \cup \mathcal{O}(G^\dagger)$), $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$.*

Proof. If v does not belong to C^\dagger , then $\delta(v.C^\dagger) = \delta(C^\dagger)$ trivially holds. Moreover, as shown in the proof of Proposition 4.2, we have $\lambda(v.C^\dagger) = \lambda(C^\dagger)$ and $\varrho(v.C^\dagger) = \varrho(C^\dagger)$ whenever v is a node in C^\dagger but is not the first one. This also gives $\delta(v.C^\dagger) = \delta(C^\dagger)$ in this case.

When v is the first node in C^\dagger , there are two cases to consider.

1. $v \in \mathcal{N}(G^\dagger)$. In this case, v executes LR2 and thus reverses and marks all its incoming links. It follows that $\ell^1(v.C^\dagger) = \ell^1(C^\dagger) + 1$ and $r^1(v.C^\dagger) = r^1(C^\dagger) - 1$. Thereby, $\delta(v.C^\dagger) = \delta(C^\dagger) + 2$.
2. $v \in \mathcal{S}(G^\dagger) \cup \mathcal{O}(G^\dagger)$. Then v necessarily executes LR1, and so unmarks but does not reverse all of its marked incoming links, and reverses and marks the others. Thus we have either $\ell^1(v.C^\dagger) = \ell^1(C^\dagger) + 1$ and $r^1(v.C^\dagger) = r^1(C^\dagger)$, or $\ell^1(v.C^\dagger) = \ell^1(C^\dagger)$ and $r^1(v.C^\dagger) = r^1(C^\dagger) - 1$, according to the label of the first edge in C^\dagger . In both cases, this gives $\delta(v.C^\dagger) = \delta(C^\dagger) + 1$.

□

Proposition 4.4. *If v is a sink node in G^\dagger , then*

$$\mathcal{S}(v.G^\dagger) = \mathcal{S}(G^\dagger) \setminus \{v\}, \quad \mathcal{N}(v.G^\dagger) = \mathcal{N}(G^\dagger) \cup (\mathcal{S}(G^\dagger) \cap \{v\}), \quad \mathcal{O}(v.G^\dagger) = \mathcal{O}(G^\dagger).$$

Proof. Clearly, the step taken by v in \mathcal{LR} cannot affect, in either direction or label, the links incident to nodes that are outside the neighborhood of v . So let w be any node in the neighborhood of v . We are going to study where w migrates in the new node partitioning of $v.G^\dagger$. There are two possible cases:

1. $w = v$. From the \mathcal{LR} rules, it follows that $v \in \mathcal{O}(v.G^\dagger)$ or $v \in \mathcal{N}(v.G^\dagger)$ depending on whether $v \in \mathcal{O}(G^\dagger)$ or not.
2. There is a link e from w to v . Since v is a sink node, it follows that $w \notin \mathcal{S}(G^\dagger)$, i.e., $w \in \mathcal{N}(G^\dagger)$ or $w \in \mathcal{O}(G^\dagger)$. In $v.G^\dagger$, there are only two possible configurations for e : e remains an incoming link for v and is unmarked, or e is an outgoing link for v and is marked. In both cases, the new configuration of e cannot modify the status of w , i.e., if $w \in \mathcal{N}(G^\dagger)$ then $w \in \mathcal{N}(v.G^\dagger)$ and if $w \in \mathcal{O}(G^\dagger)$ then $w \in \mathcal{O}(v.G^\dagger)$.

Thereby, when v makes a reversal, our node partitioning remains unchanged except for the migration of v from \mathcal{S} to \mathcal{N} in the case v is initially a sink all of whose links are unmarked. \square