A Distributed $O(|E|)$ Algorithm for Optimal Link-Reversal

Sukhamay Kundu

Computer Sc. Dept, Louisiana State University
Baton Rouge, LA 70803, USA
kundu@csc.lsu.edu

Abstract. We first characterize the minimal link-sets L whose directions must be reversed for reestablishing one or more directed paths from each node x to a fixed destination node d in a network when a link fails. Then, we give a distributed $O(|E|)$ algorithm for determining such a link-set L, where $|E| = \sharp$(links in the network). This improves the previous lower bound $O(n^2)$, where $n = \sharp$(nodes in the network). The minimality of the reversed link-set L has other important consequences.

1 Introduction

We consider a connected network $G = (V,\ E)$ with a fixed destination node d, where all links E are directed to form an acyclic digraph with one or more directed paths to d from each node x. A message arriving at $x \neq d$ along any of its incoming links is forwarded via one of its outgoing links. Thus, a message originating at any node follows an acyclic path and reaches d. The link-reversal is an important technique to reestablish one or more new xd-paths for each x when a failed link $(y,\ z)$ destroys all current yd-paths and possibly all xd-paths from some other nodes x as well. Our algorithm differs from other link-reversal algorithms in several ways: (1) we first determine a minimal set of links L whose directions can be reversed to reestablish at least one path to d from each node x, keeping the network acyclic, and then we reverse the direction of the links in L; in particular, we reverse a link at most once. (2) it improves the time complexity to $O(|E|)$ from $O(|V|^2)$ for the previous algorithms [1-4], and (3) each node x knows when its computation terminates so that it can begin to redirect the messages properly towards the destination node d along the new xd-paths.

The importance of minimizing the reversed link-set L can be seen follows. For each link $u \to v \in L$, the messages currently queued at u for transmission to d have to be redirected now along a new link $u \to w$. Also, the messages that are queued at v (some of which might have arrived at v via the link $u \to v$) may be directed now either along $v \to u$ or $v \to w'$ for $w' \neq u$ (which might also be in L). Clearly, it would be best if we could choose L that minimizes $|\{v\text{: } v$ is one of the end nodes of a link in $L\}|$. Our algorithm does not always achieve this. Some other link-reversal algorithms in the literature are TORA [2, 3] and LMR [4]; they are variations of the original algorithm in [1]. The performance analysis of a class of link-reversal algorithms is given in [5], [6] gives an overview of link-reversal algorithms, and [7, 8] give some recent surveys.

2 Terminology

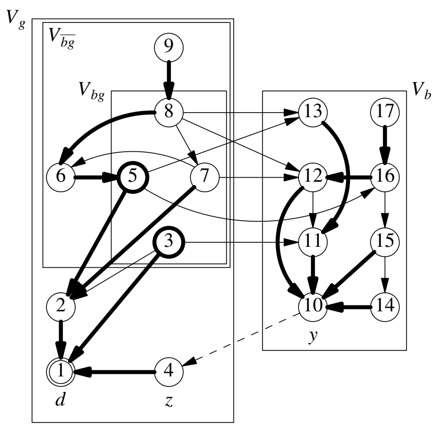
We sometimes write $x \to x'$ to emphasize that the undirected link $(x,\ x') \in E$ is currently directed from x to x'. We say that both x and x' are *incident* with or *belong* to the link $x \to x'$. A path π from x to y (in short, an xy-path) will mean a directed path; we say that a node u is incident with or belongs to π if it belongs to a link in π. We write $N^+(x) = \{x'$: the link $(x,\ x')$ is directed as $x \to x'\}$; the nodes in $N^+(x)$ are called the *downstream* neighbors of x. Likewise, we write $N^-(x) = \{x'$: the link $(x,\ x')$ is directed as $x' \to x\}$ and the nodes in $N^-(x)$ are called the *upstream* neighbors of x. The node x is a *source*-node if $N^-(x) = \emptyset$ and is a *sink*-node if $N^+(x) = \emptyset$. Initially, d is the only sink-node.

Let $G(y,\ z)$ denote the directed network G without the failed link $y \to z$. We assume that the elimination of the link $y \to z$ destroys all yd-paths without disconnecting the network itself, i.e., $G(y,\ z)$ is still connected as an undirected graph. This means $y \to z$ is the only downstream link from y, and y is the only sink-node other than d in $G(y,\ z)$. Let $V_b = \{x$: there is no xd-path in $G(y,\ z)\}$, the set of "bad" nodes; clearly, $y \in V_b$. For $x \in V_b$, all xd-paths in G use the link $y \to z$. Let $V_g = V - V_b$, the set of "good" nodes, i.e., the nodes which still have at least one path to d in $G(y,\ z)$. See Fig. 1(i). The nodes V_b form an acyclic digraph of G with node y as the only sink-node and the nodes V_g form an acyclic digraph of G with node d as the only sink-node. Also, all links connecting nodes in V_g and nodes in V_b are directed to nodes in V_b, and there is at least one such link. Note that node z can be arbitrarily far away from node d. We write G_b for the subdigraph of G on V_b and G_g for the subdigraph of G on V_g.

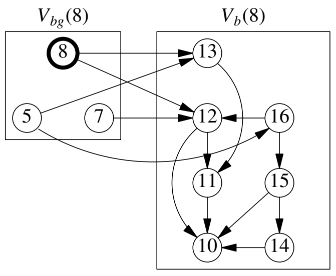
We write $V_{bg} = \{x \in V_g$: there is a link $x \to x'$ for some $x' \in V_b\} \subset V_g$. This is the subset of good nodes which have at least one path in $G(y,\ z)$ to y without using any link in G_g. We define $V_{\overline{bg}} = \{x \in V_g$: there is a path in $G(y,\ z)$ from x to some node in V_b and hence to $y\} \subset V_g$; this is the subset of good nodes which have at least one path to y using zero or more links in G_g. Clearly, $V_{bg} \subseteq V_{\overline{bg}}$ and the nodes d and z do not belong to $V_{\overline{bg}}$. We write G_{bg} for the subdigraph of G on V_{bg}. We sometimes call a node in V_b simply a b-node, a node in V_{bg} a bg-node, a node in $V_{\overline{bg}} - V_{bg}$ a $\overline{bg}$-node, and finally a node in $V_g - V_{\overline{bg}}$ a g-node. Note that a bg-node is a good-node that has a downstream b-node neighbor, and that not all good nodes are g-nodes. We write type(x) for the $b/g/bg/\overline{bg}$-type of node x. In Fig. 1(i), the sets V_b, V_g, V_{bg}, and $V_{\overline{bg}}$ are unchanged if we remove any one or more of the links in $\{8 \to 7,\ 8 \to 6,\ 7 \to 6\}$ except for removing both $8 \to 7$ and $8 \to 6$; in that case, the nodes 8 and 9 become b-nodes.

3 Minimal Link-Sets for Reversal

Definition 1. *A subset of links L in $G(y,\ z)$ is called a reversal link-set if the reversal of their directions reestablishes at least one xd-path in $G(y,\ z)$ from*



(i) A network G with all links directed to form an acyclic digraph with d as the only sink-node.



(ii) The links $L(8)$, the nodes $V_{bg}(8)$ and $V_b(8)$ belonging to $L(8)$, and the subdgraph $G(8)$ of G.

Fig. 1. Illustration of B_b, V_g, V_{bg}, etc. The bold lines show a spanning-tree T_g on V_g rooted at d and a spanning-tree T_b on V_b rooted at y.

each node x, keeping the digraph acyclic. We write L_{min} for a minimal reversal link-set set L; such a set does not contain any link joining the nodes in V_g.

In Fig. 1(i), the two possible minimal L are $L_{min} = \{3 \to 11,\ 11 \to 10\}$, which corresponds to the links on the paths from node 3 to node 10, and $L_{min} = \{5 \to 16,\ 5 \to 13,\ 16 \to 15,\ 16 \to 12,\ 15 \to 14,\ 15 \to 10,\ 13 \to 11,\ 12 \to 11,\ 12 \to 10,\ 11 \to 10\}$, which corresponds to the links on the paths from node 5 to node 10. There are many non-minimal L's. It is clear that for any reversal link-set L we have L contains at least one link from $L_{bg} = \{x \to x' \colon x \in V_{bg}$ and $x' \in V_b\}$. We now characterize all reversal link-sets L. Let $x \in V_{bg}$ and π_{xy} be an xy-path. If we reverse the directions of all links in π_{xy}, then each node in V_b now has a path to x and hence to d. To see this, let $x_b \in V_b$ which is not on π_{xy} and let π' be a path from x_b to a node x'' in π_{xy}, where x'' may equal y. Let π'' be the initial part of π_{xy} upto the node x''. Clearly, π' together with reversed version of π'' gives an $x_b x$-path and hence we have an $x''d$-path. However, just reversing the links in π_{xy} may create one or more cycles, as is the case in Fig. 1(i) for $x = 8$ and $\pi_{xy} = \langle 8, 12, 11, 10 \rangle$. Some of the cycles formed on reversing the links in the path π_{xy} are $\langle 8, 13, 11, 12, 8 \rangle$ and $\langle 12, 10, 11, 12 \rangle$. Indeed, any cycle formed would involve links of a path π_{uv} joining two nodes u and v on π_{xy} where π_{uv} is disjoint from π_{xy} except for the end nodes u and v. This suggests the following definition. (Note that choosing π_{xy} a shortest or longest xy-path does not resolve this problem, in general.)

Definition 2. *For a node $x \in V_{bg}$, let $L(x) = \{u \to v \colon u \to v$ is in some xy-path and $v \in V_b\} = \{u \to v \colon u \to v$ is in some xy-path and not both u and $v \in V_g\}$. Also, let $G(x)$ denote the subdigraph of $G(y,\ z)$ consisting of the links $L(x)$ and the nodes belonging to those links. See Fig. 1(ii).*

Theorem 1. *For each $x \in V_{bg}$, $L(x)$ is a reversal link-set. Also, each reversal link-set $L = \bigcup\{L(x) :\ x \in V_{bg}$ and x belongs to a link in $L\}$.* ♡

The proof of Theorem 1 and other proofs are omitted here for want of space. Def. 2 plays a critical role in the proof of Theorem 1.

Definition 3. *A node $x \in V_{bg}$ is called a bg-sink node if there is no path from x in $G(y,\ z)$ to some other node in V_{bg}. (Although a bg-sink node x is a sink-node in G_{bg}, the converse need not be true; nodes $\{3,\ 5\}$ are the only bg-sink nodes in Fig. 1(i).) We write $S_{bg} \subseteq V_{bg}$ for the set of bg-sink nodes.*

The following corollary is immediate from Theorem 1 and forms the basis of our algorithms in the next Section.

Corollary 1. *A necessary and sufficient condition for L to be a minimal reversal link-set is $L = L(x)$ for some bg-sink node x.* ♡

4 Minimal Link-Reversal Algorithm

There are several phases in the our algorithm using a minimal reversal link-set $L = L(x)$ for some bg-sink node x. We assume that each node x knows its $N^+(x)$ and $N^-(x)$.

We first identify the node-type (b, g, bg, or $\overline{bg}$) of each node of G by using a standard "flooding" technique on $G(y,\ z)$. We use four kinds of messages, but only one kind from each node (and only one message on any link). A node x sends the same (b, g, bg, or $\overline{bg}$) message to all its upstream neighbors $N^-(x)$ in the flooding, i.e., backwards along the link to x after it receives the messages from all its downstream neighbors and identifies its type. Thus, a b-node sends only b-messages, a g-node sends only g-messages, etc. In the process, we also determine a directed spanning tree T_b of V_b rooted at y and a directed spanning tree T_g of V_g rooted at d, with all links directed from a children to its parent; see Fig. 1(i). The trees T_b and T_g are used later for choosing a specific bg-sink node x, the associated minimal link-set $L(x)$, etc.

The flooding begins with node d marking itself as a g-node and sending a g-message to all nodes in $N^-(d)$ and similarly with node y marking itself as a b-node and sending a b-message to all nodes in $N^-(y)$. Subsequently, when a node x has received a message from each of its neighbors $N^+(x)$, it determines its node-type and the bg-sink node status according to the criteria (C1)-(C5) below and then forwards the corresponding $b/g/bg/\overline{bg}$-message to each node in $N^-(x)$. Note that a node x can determine its type to be bg before receiving a $b/g/bg/\overline{bg}$-message from each of $N^+(x)$, but we nevertheless wait to send the bg-message out from x till x receives the $b/g/bg/\overline{bg}$-messages from all of $N^+(x)$ for the sake of simplicity.

(C1) If x receives only b-messages, then type(x) = b.
(C2) If x receives only g-messages, then type(x) = g.
(C3) If x receives at least one b-message and at least one $g/bg/\overline{bg}$-message, then type(x) = bg.

(C4) If x receives no b-message and at least one $bg/\overline{bg}$-message (and possibly zero or more g-message), then type$(x) = \overline{bg}$.

(C5) Finally, if type$(x) = bg$ and x has not received any $bg/\overline{bg}$-message, i.e., it has received only b/g-messages, then x is a bg-sink node.

4.1 Algorithm FindSinkNodes S_{bg}

We assume that when the only outgoing link $(y,\ z)$ from a node y fails, node y initiates the algorithm FindSinkNodes; in addition, it asks node z to inform the destination node d of the link-failure situation (which can be done using at most $O(|E|)$ messages) and then node d on receiving this information initiates the algorithm FindSinkNodes. The other nodes x in $G(y,\ z)$ start executing the algorithm FindSinkNodes when they receive their first $b/g/bg/\overline{bg}$-message. We also assume that the node failures are sufficiently infrequent that all phases of the link-reversal algorithm terminates before a new link failure occurs. We simply use the conditions (C1)-(C5) as described above to determine the type of each node, including if it is a bg-sink node. For this purpose, each node x maintains a vector of four counts numBmssgRcvd(x), numGmssgRcvd(x), numBGmssgRcvd(x), and num$\overline{BG}$mssgRcvd(x). The termination of FindSinkNodes requires additional work, including identification of the spanning trees T_b and T_g, and is described later.

Theorem 2. *FindSinkNodes uses a total of $O(|E|)$ messages.* ♡

The first column in Fig. 2 shows a possible sequence in which the nodes in Fig. 1(i) can determine their types. Here, we assume that each message takes one unit of time to reach its recipient and zero processing time for the messages. The smallest node x which received all $|N^+(x)|$ messages first is listed first. We indicate type(x) in parentheses next to x in the first column. In the second column, we show the $b/g/bg/\overline{bg}$-message sent by x and the nodes $u \in N^-(x)$ receiving that message; we also show type(u) next to each u provided u can determine its type based on the messages received upto this point. Until all nodes have determined their types, there is always at least one node which is ready to forward its selected $b/g/bg/\overline{bg}$-message. The third column shows the determination of parent-child links in T_b and T_g, which is explained below.

4.2 Terminating Computation of bg-Sink Nodes

We use two other messages for this purpose: a p-message (p for "parent") and an np-message (np for "not parent"). Exactly one of these messages go forward along each link of $G(y,\ z)$.

We let each node $x \notin \{d,\ y\}$ maintain a unique parent(x), which is the node from which x receives its first $g/bg/\overline{bg}$-message if type$(x) \in \{g, bg, \overline{bg}\}$ or the node from which x receives its first b-message if type$(x) = b$. Note that x may have to wait till it receives a $b/g/bg/\overline{bg}$-message from each of the neighbors $N^+(x)$ to determine parent(x) simply because x may not know type(x) until

Node x and type(x)	$b/g/bg/\overline{bg}$-message sent to $N^{-}(x)$	Parent-child link in T_b or T_g
$1(g)$	g: $2(g)$, 3, $4(g)$	$2{\to}1$, $4{\to}1$ in T_g
$10(b)$	b: $11(b)$, 12, $14(b)$, 15	$11{\to}10$, $14{\to}10$ in T_b
$2(g)$	g: 3, 5, 7	
$4(g)$	g: –	
$11(b)$	b: $3(bg)$, $12(b)$, $13(b)$	$12{\to}10$, $13{\to}11$ in T_b and $3{\to}1$ in T_g
$14(b)$	b: $15(b)$	$15{\to}10$ in T_b
$3(bg)$	bg: –	
$12(b)$	b: $7(bg)$, 8, 16	$7{\to}2$ in T_g
$13(b)$	b: $5(bg)$, 8	$5{\to}2$ in T_g
$15(b)$	b: $16(b)$	$16{\to}12$ in T_b
$16(b)$	b: $17(b)$, 5	$17{\to}16$ in T_b
$5(bg)$	bg: $6(\overline{bg})$	$6{\to}5$ in T_g
$17(b)$	b: –	
$6(\overline{bg})$	$\overline{bg}$: 7, $8(bg)$	$8{\to}6$ in T_g
$7(bg)$	bg: 8	
$8(bg)$	bg: $9(\overline{bg})$	$9{\to}8$ in T_g
$9(\overline{bg})$	$\overline{bg}$: –	

Fig. 2. A possible sequence of the determination of type(x) for nodes x in Fig. 1(i)

that point. This means a node x has to maintain four potential parents, one for each possible value of type(x) and in the order they are found, and then finally select parent(x) to be one of those based on type(x). It is easy to see that the parent-links form two trees, a tree T_b spanning the nodes V_b with the root at y and a tree T_g spanning the nodes V_g with the root at d. The bold links in Fig. 1(i) show these trees assuming that each $b/g/bg$-message takes one unit time to reach its recipient. We have chosen parent(16) to be simply the smaller of $\{12,\ 15\}$ from which it receives b-message at time 3.

After a node $x \notin \{d,\ y\}$ has determined type(x) it can safely determine parent(x) and at that point it sends an np-message to each node in $N^{+}(x)$ other than parent(x). (Sometime, it is actually possible to determine parent(x) even earlier; for example, node 3 in Fig. 1(i) can determine parent$(3) = 1$ after it receives its first message from node 1, which is a g-message.) A node x waits, however, to send the p-message to parent(x) until the number of p/np-messages received by x equals $|N^{-}(x)|$, and it is at this point that x terminates its computation for determining the bg-sink nodes. This means, in particular, that p-messages are initiated by the source-nodes in $G(y,\ z)$, which includes node z, and perhaps zero or more other terminal nodes in the trees T_b and T_g. For Fig. 1(i), only the nodes $\{3,\ 4,\ 9,\ 17\}$ initiate p-messages. The terminal node 13 in T_b sends its p-message to node 11 only after receiving np-messages from the nodes 5 and 8; similarly, the terminal node 7 in T_g sends its p-message to node 2 only after receiving np-message from node 8. The special node y (d) terminates its computation last among the nodes in V_b (resp., V_g) and this happens when it receives $|N^{-}(y)|$ (resp., $|N^{-}(d)|$) many p/np-messages.

4.3 Selecting One *bg*-Sink Node

We briefly describes the steps in this task. Only the nodes $V_{bg} \cup V_b$ participates in this step; the only links that participate in this step are L_{bg} and the links in T_b. We use two kinds of s-messages (s for sink-node). (Note that we do not try to choose $x \in S_{bg}$ such that $|L(x)|$ is minimum because this can increase the total number of messages to $O(|V|^3)$.) We use the null s-message, $s()$, to coordinate the starting and ending of computations of the nodes participating in this phase, and use a non-null s-message $s(x)$, where x is a bg-sink node.

We begin by letting each node $x \in S_{bg}$ initiate the message $s(x)$ to the smallest numbered node from which it received a b-message during the execution of FindSinkNodes algorithm and initiate the message $s()$ to all other nodes from which it received b-message. For each node $x' \in V_{bg} - S_{bg}$, we let it initiate the message $s()$ to all nodes from which it received b-message. (This means each terminal node in T_b, which sends b-messages only to nodes in V_{bg}, will receive exactly a total of $|N^-(u)| \geq 0$ many s-messages.) For Fig. 1(i), node 3 initiates the message $s(3)$ to node 11, node 5 initiates the message $s(5)$ to node 13 and $s()$ to node 16, node 7 initiates the message $s()$ to node 12, node 8 initiates the message $s()$ to each of the nodes 12 and 13, and finally node 17 initiates the message $s()$ to node 16.

Each terminal node u in T_b on receiving $|N^-(u)|$ many s-messages does the following: if it does not receive any non-null s-message, then it sends $s()$ to its parent; otherwise, if $s(x)$ is the first non-null s-message received by u, then it sends the entire s-message $s(x)$ to its parent. For a non-terminal node u in T_b, exactly the same thing happens except that some of its $|N^-(u)|$ many s-messages come from its children before it sends an appropriate s-message to its parent. When the root node y of T_b receives an s-message from each of its children, it selects the node x in the first non-null s-message it received and terminates its computation for this phase. The other nodes in T_b terminate their computation in this phase on sending their s-message to their parents, and the nodes in V_{bg} terminate their computation on sending their s-messages to the nodes in V_b. Let x_s denote the bg-sink node x selected by y.

The following theorem is immediate from the fact that there will be exactly one s-message along each link of the tree T_b, in addition to at most $\sum |N^+(x)|$, summed over $x \in S_{bg}$, many s-messages originating from the nodes in S_{bg}.

Theorem 3. *The selection of a specific bg-sink node $x_s \in S_{bg}$ by the special node y takes $O(|E|)$ messages.* ♡

4.4 Actual Link-Reversal Phase

This is the final step of our algorithm and has two parts. In the first part, node y informs the bg-sink node x_s that it has been selected, and in the second part each link in the set $L(x_s)$ is reversed, i.e., each node x belonging to the links in $L(x_s)$ updates $N^-(x)$ and $N^-(x)$ to represent the new orientation of G.

For the first part, we assume that during the selection of x_s, each node u in T_b, save $u \neq y$, which forwarded a non-null s-message of the form $s(x)$ remembers both the parameter x in that s-message and the node u' from which it received that s-message. We have u' is either a child of u in T_b or $u' = x$, the gb-sink node where the message $s(x)$ originated. By abuse of notation, we write s-child$(u) = u'$, which is unique. We can follow the trail of nodes backward (from u to u') starting from the root y of T_b corresponding to the s-message $s(x_s)$. We begin by node y sending an f-message (f for find) to s-child(y), which is the child of y that sent the message $s(x_s)$ to y. Then $y' = s$-child(y) sends an f-message to s-child(y'), and so on until we reach the node $x \in V_{bg}$.

The second part begins with the bg-sink node x_s reversing each of the links $x_s \to u$ corresponding to the nodes u from which it received b-message, i.e., adding each such u to $N^-(x_s)$ and deleting u from $N^+(x_s)$, and then sending an r-message (r for reverse) to each of those node u. When a b-node u receives the first r-message, it does exactly the same thing as above, updating its $N^-(u)$ and $N^+(u)$, and then sending an r-message to each node added to $N^-(u)$. The second and subsequent r-messages received by a b-node u, if any, are ignored.

Theorem 4. *The reversal of the links in $L(x_s)$ for the selected bg-sink node x_s takes $O(|E|)$ messages.* $\heartsuit$

5 Conclusion

We have presented a $O(|E|)$ distributed algorithm for the link-reversal problem, where $|E| = \sharp$(links in the network). It first selects a minimal reversal link-set and reverses the direction of those links exactly once.

References

1. Gafni, E.M., Bertsekas, D.P.: Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans. on Communication 29, 11–18 (1981)
2. Park, V.D., Corson, M.S.: A highly adaptive distributed routing algorithm for mobile wireless networks. In: Proc. IEEE Conf. on Computer Communications (INFOCOM 1997) (1997)
3. Park, V.D., Corson, M.S.: A performance comparison of temporally-ordered routing algorithm and ideal link-state routing. In: Proc. IEEE Intern. Symp. on Systems and Communications (June 1998)
4. Corson, M.S., Ephremides, A.: A distributed algorithm for mobile wireless networks. ACM Wireless Networks Journal 1, 61–82 (1995)
5. Busch, C., Surapaneni, S., Tirthapura, S.: Analysis of link-reversal routing algorithms for mobile ad-hoc networks. In: Proc. of 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), San Diego, pp. 210–219 (June 2003)
6. Perkins, C.E.: Ad-hoc Networking. Addison-Wesley, Reading (2000)
7. Rajarama, R.: Topology control and routing in ad-hoc networks: a survey. SIGACT News (June 2002)
8. Samir, R., Robert, C., Jiangtao, Y., Rimli, S.: Comparative performance evaluation of routing protocols for mobile ad-hoc networks. In: Proc. IEEE 7th Intern. Conf. on Computer Communications and Networks (IC3N 1998) (1998)