Distributed Computation of All Node Replacements of a Minimum Spanning Tree*

Paola Flocchini[1], Toni Mesa Enriquez[2], Linda Pagli[3], Giuseppe Prencipe[3], and Nicola Santoro[4]

[1] University of Ottawa, Canada
flocchin@site.uottawa.ca
[2] Universidad de La Habana, Cuba
tonymesa@matcom.uh.cu
[3] Università di Pisa, Italy
{pagli,prencipe}@di.unipi.it
[4] Carleton University, Canada
santoro@scs.carleton.ca

Abstract. In many network applications the computation takes place on the *minimum-cost* spanning tree (MST) of the network; unfortunately, a single link or node failure disconnects the tree.

In this paper we consider for the first time the problem of computing all the replacement minimum-cost spanning trees *distributively*, and we efficiently solve the problem. We design a solution protocol and we prove that the total amount of data items communicated during the computation is $O(n^2)$. This communication can be achieved either transmitting $O(n)$ long messages, if the system so allows, or $O(n^2)$ standard messages. Even in systems that do not allow long messages, the proposed protocol constitutes a significant improvement over the individual computation of the replacement trees.

Keywords: Minimum Spanning Tree, Replacement Tree, Node Failure, Distributed Algorithms.

1 Introduction

1.1 The Framework

In most network applications, the computation takes place not on the entire network but solely on a spanning subnet. There are several reasons for this fact; first and foremost, it is done to reduce the amount of communication and thus the associated costs; it is done also for security reasons, e.g. to minimize the exposure of messages to external eavesdroppers. The subnet used is typically a special spanning tree of the network G; in particular, the *minimum-cost* spanning tree (MST) is used for basic network tasks such as broadcasting, multicasting, leader election and synchronization. The major drawback of using a MST is the

* Research partially supported by NSERC Canada.

high vulnerability of its tree structure to link and/or node failures: a single failure disconnects the spanning tree, interrupting the message transmission. Hence it is crucial to update the MST after changes in network topology. In this paper we update MST after single node deletions. In a graph $G = (V, E)$, with n nodes there are n possible instances of a single node deletions. Let T be the MST of G. Informally, the *All Node Replacements* (ANR) problem is to update T in each of the instances of single node deletion. Observe that this problem is much more difficult than the related *All Link Replacement* (ALR) problem where the goal is to update T in each of the instances of single *edge* deletion. In fact, the deletion of a single node u is equivalent to the simultaneous deletion of all its $deg(u)$ incident edges.

The re-computation of the new MST in each instance is rather expensive. This is particularly true if the re-computation is done distributively in the network after a failure; in addition, if the failures in the system are mostly temporary, the usefulness of these re-computations is limited and the rational for affording their cost becomes questionable. For these reasons, to solve the All Node Replacements problem in reality means to *pre-compute* the n replacement minimum spanning trees, one for each possible node failure in the tree [4,10,13]; the computed information is then used only if a node fails, and only as long as the failure persists. The computational challenge is to be able to combine work among the n different pre-computations, so that the the total cost is much less than that incurred by computing each replacement tree individually. This problem has been extensively investigated, and efficient solutions have been developed for both the sequential and parallel settings (e.g., see [3,4,10,13,15]).

In this paper we consider the *distributed* version of this problem. That is we investigate the All Node Replacements problem when the computational entities are nodes of G themselves, and each can only communicate by exchanging messages with its neighbours. The network itself must pre-compute the n replacement minimum spanning trees; the information so obtained is then stored (distributively) together with the original MST tree T, and used whenever a node failure is detected; the original minimum spanning tree T is reactivated once the network has recovered from the transient fault.

The repeated application of a distributed MST construction protocol (e.g., [7,11]) will cost at least $O(nm + n^2 \log n)$ messages, where m denotes the number of edges. Surprisingly, no more efficient distributed solutions exist for this problem, prior to this work. As stated in [4] (where efficient serial and parallel solutions were presented): *Designing an efficient distributed algorithm for ANR remains an open problem.*

1.2 Main Result

In this paper we consider the problem of computing all the replacement minimum-cost spanning trees *distributively*, and we efficiently solve the problem.

We design a distributed algorithm for computing all the replacement MSTs of the minimum cost spanning tree T of the network G, one for each possible node failure, and we show how to store the computed information in order to restore the tree's connectivity when the temporary fault occurs.

We prove that the total amount of data items communicated during the computation (the data complexity) is $O(n^2)$. This communication can be achieved transmitting only $O(n)$ long messages between neighbours, if the system so allows; otherwise $O(n^2)$ standard messages suffice. In other words, with this complexity, our protocol constructs a MST that maintains its minimum-cost properties even after a single (but arbitrary) link or node failure.

Even in systems that do not allow long messages, the proposed protocol constitutes a significant improvement over the individual computation of the replacement trees. Indeed, for dense graphs, our protocol constructs all the n replacement MSTs of the minimum spanning tree T with the same number of messages required just to compute T.

The communication structure of the algorithm is surprisingly simple, as it consist of a single broadcast phase followed by a convergecast phase. The difficulty is to determine what information is locally needed, which items of data have to be transmitted in these two phases, and how the communicated information must be locally employed. This schema is reminescent of the one used for computing all the swap-edges of a shortest-path tree [8,9], but the similarity is limited to the structure. In fact, since the failure of a single node u is equivalent to the simultaneous deletion of all its $deg(u)$ incident edges, the nature of the problem changes dramatically, and those approaches can not be used here.

They can however be employed, as we show, to solve the simpler All Link Replacement (ALR) problem where the goal is to update T in each of the instances of single *edge* deletion.

1.3 Related Work

The All Node Replacements (ANR) problem was first studied in a *serial* environment by Chin and Houck [3]. A more efficient solution has been developed by Das and Loui [4], and later improved by Nardelli, Proietti and Widmayer [13]. When G is *planar*, improved bounds have been obtained by Gaibisso, Proietti and Tan [10]. The simpler All Edge Replacements (AER) problem is implicitly solved by Dixon, Rauch and Tarjan [5]; an improved solution was later developed by Nardelli, Proietti and Widmayer [13].

In the *parallel* setting, Tsin presented an algorithm to update a MST after a single node deletion [15]; thus, concurrent use of this algorithm solves ANR in parallel. A subsequent parallel solution to ANR is obtained by combining the parallel algorithms presented by Johnson and Metaxas [12]. A more efficient parallel technique has been designed by Das and Loui [4]. The simpler All Edge Replacements (AER) problem is efficiently solved by using the parallel verification algorithm of Dixon and Tarjan [6].

In the *distributed* setting, the *construction* of the MST of a network has received considerable attention. The well known protocol by Gallager, Humblet and Spira uses $O(m + n \log n)$ messages, where m denotes the number of edges [11]. This protocol is not only elegant but also optimal, since $\Omega(m + n \log n)$

messages are needed regardless of their size [14]. In fact, all subsequent work (e.g., [7]) has been dedicated to reducing the time needed in synchronous executions.

To solve AER and ANR, one may use repeated applications of a distributed MST construction protocol; this brute-force approach will cost at least $O(nm + n^2 \log n)$ messages. The more complex problem of updating a MST with multiple node and edge deletions was considered by Cheng, Cimet and Kumar [2]; however, when used in the ANR and in the AER problems, their solution would not yield any improvement over the brute-force approach (it would actually be worse). Indeed, prior to this work, no efficient distributed solutions exist for either problems.

2 Terminology and Problems

2.1 Definitions

Let $G = (V, E)$ be an undirected graph, with $n = |V|$ vertices and $m = |E|$ edges. A *label* of length $l \leq \log n$ is associated to each vertex of G. A non negative real *weight* $w(e)$ is associated to each edge e. A *subgraph* $G' = (V', E')$ of G is such that $V' \subseteq V$ and $E' \subseteq E$. If $V' \equiv V$ and G' is connected, then G' is a *spanning* subgraph. A graph G is *2-edge connected* or *2-node connected* if it remains connected after the removal of any one of its edges (or any one of its nodes). Let $T = (V, E(T))$ be a spanning tree of graph G rooted in r, arbitrary node of T. A spanning tree $T = (V, E(T))$ is called *minimum spanning tree MST* of G if the sum of tree edge weights is minimum over all spanning trees.

A subtree rooted at some node x is denoted by T_x. The *parent* of a node x is indicated as *parent*(x) and its *children* as *children*(x). Consider an edge $e = (x, y) \in E(T)$ with y closer to r, the root of T; if such an edge is removed, the tree is disconnected in two subtrees: T_x and $T \backslash T_x$. A *swap* edge for $e = (x, y)$ is any edge $e' = (u, v) \in E \setminus \{e\}$ that connects the two subtrees. It can be easily seen that the MST of $G - e$, called the replacement tree T_{G-e} can be computed by selecting the swap edge of minimum weight connecting T_x and $T \setminus T_x$.

We consider a *distributed computing system* with communication topology G. Each computational entity x is located at a node of G, has local processing and storage capabilities, has a unique label $\lambda_x(e)$ from a totally ordered set associated to each of its incident edges e, knows the weight of its incident edges, and can communicate with its neighboring entities by transmission of bounded sequences of bits called messages. The nodes do not know the topology G, but only their incident edges with their labels. The communication time includes processing, queueing, and transmission delays, and it is finite but otherwise unpredictable. In other words, the system is *asynchronous*. All the entities execute the same set of rules, called *distributed algorithm* (e.g., see [14]).

In the following, when no ambiguity arises, we will use the terms entity, node and vertex as equivalent; analogously, we will use the terms link, arc and edge interchangeably.

2.2 The All Edges Replacement Problem and Its Solution

Let G be 2-edge connected. The *All Edges Replacement* problem, denoted as *AER(G,T)* with input G and T is that of finding T_{G-e} for every edge $e \in E(T)$.

The *AER(G,T)* problem can be solved distributively by applying one of the algorithmic shells of [9], where the input tree is now an MST of G, instead of a shortest-path tree, and where the *best swap edge* e' for e is the one leading to the minimal total weight; hence, this function can be computed locally by each node by simply summing the weight of e' and subtracting the weight of e from the total MST's weight. The overall message complexity is then the same as in [9] amounting to $O(n_r^*)$, where n_r^* is the number of edges of the transitive closure of $T \setminus \{r\}$ and $0 \leq n_r^* \leq (n-1)(n-2)/2$, which is of $O(n^2)$.

2.3 The All Nodes Replacement Problem

Let $G = (V, E)$ be 2-node connected. Consider a node $x \in V$; if such node is removed from T together with its incident edges, the tree is disconnected into the subtrees T_{x_1}, ... , T_{x_k}, where $x_1, ..., x_k$ are the children of x; let $T' = T \setminus \{T_{x_1}, ..., T_{x_k}, \{x\}\}$. Let x_0 be the parent of x, and E' be the set of non tree edges; we will call $\mathcal{U}_x = \{e = (u, v) \in E' | u \in T_{x_i}, 1 \leq i \leq k, v \in T'\}$ the set of *upwards edges* of x and $\mathcal{H}_x = \{e = (u, v) \in E' | u \in T_{x_i}, v \in T_{x_j}, 1 \leq i, j \leq k, i \neq j\}$ the set of *horizontal edges* of x. For node x, the set of the *best upward edges* $\mathcal{U}'_x \subseteq \mathcal{U}_x$ is the set containing the edges of minimum weight (if any) connecting $T_{x_i}, 1 \leq i \leq k$ and T', and the set of the *best horizontal edges* is the set $\mathcal{H}'_x \subseteq \mathcal{H}_x$ containing the edges of *minimum weight* connecting T_{x_i} and $T_{x_j}, 1 \leq i \neq j \leq k$ (if any). In the following, we will use also the notation $\mathcal{U}$, $\mathcal{U}'$, $\mathcal{H}$, and $\mathcal{H}'$, when the reference to the removed node is clear from the context.

From [13] we know that the MST of $G - x$ can be computed through the computation of the MST of the *contracted graph* $G_x = (V_x, E_x)$, where $V_x = x_0, x_1, ...x_k$ and $E_x = \mathcal{H}' \cup \mathcal{U}'$, obtained contracting to a single vertex each subtree $T_{x_i}, 1 \leq i \leq k$, and T'. The edges of the obtained MST, say T_{G-x}, are the *replacement set* of edges for x.

The computation of all the replacement sets for each node failure will be called the *All Nodes Replacement* or simply *ANR(G,T)* problem in the following. We are interested in the distributed solution of the *ANR(G,T)* problem.

3 Solving the *ANR* Problem

Consider the problem of computing the replacement edges for the failure of node x of T; the computation is performed simultaneously for all possible node failures. We first present a distributed algorithm described at high level, while the details of each module will be discussed later. At high level the algorithm consists of a *broadcast* phase started by the children of the root, followed by a *convergecast* phase started by the leaves. The idea is that each node x is able to compute its replacement set, when all its children have already computed their replacement sets in the convergecast phase. Node x determines also a set of edges, useful to compute the replacement sets for all its ancestors (except for the root), that is for $a_i, 2 \leq i \leq s$, where a_2 is the parent of x in T and a_s a child of r.

Once node x has computed its replacement set, composed of edges having at least one endpoint in its subtrees, it sends them back to its children, each one to the root of the proper subtree. In the case node x fails, each child knows which edges have to be activated in its subtree.

```
ALL NODES REPLACEMENT (ANR(G,T))

[Broadcast.]

1. Each child $x$ of the root starts the broadcast by sending to its children a list
   containing its name.
2. Each node $y$, receiving a list of names from its parent, appends its name to the
   received list and sends it to its children.

[Convergecast.]

1. Each leaf $z$ selects, among its non tree incident edges, the best upwards edge
   and the best horizontal edges for each ancestor $a$ in the received list. Then sends
   the lists of those edges to its parent (if different from $r$).
2. An internal node $y$ waits until it receives the information computed from each
   of its children: this information contains the set of the upwards edges $\mathcal{U}'$ and
   the set of horizontal edges $\mathcal{H}'$ for $y$.
   (a) $y$ computes the $MST$ of the graph $G_y = (V_y, E_y)$ where $V_y =$
       $\{parent(y), children(y)\}$ and $E_y = \{\mathcal{U}' \cup \mathcal{H}'\}$ and sends the edges of $T_{G_y}$,
       that is the replacement set $RS_y$ for $y$ to its proper subtrees.
   (b) $y$ then selects, among its incident non tree edges and the information re-
       ceived from its children, the best upwards edge and the best horizontal edges
       for each of its ancestor.
   (c) $y$ finally sends the lists of these edges to its parent (if different from $r$).
```

To show how this high level algorithmic structure works we must specify in more details the convergecast phase and, in particular, the operations executed by each node. First of all, let us define the structure of the information received by a node x from each of its children: it is composed by s lists, one for x and one for each of the other $s-1$ ancestors a_j, $2 \leq j \leq s$ (except for the root). For each $x_i, 1 \leq i \leq k$ let $L^i_j, 1 \leq j \leq s$ be the list from x_i for x and for the other ancestors a_j. Each L^i_j is composed of two fields, called UP and HOR. For $L^i_1, 1 \leq i \leq k$, the field UP , denoted as $UP(L^i_1)$ will contain the best upwards edge from T_{x_i} for x. The set composed by $UP(L^1_1)$, ... , $UP(L^k_1)$ are used to compute $\mathcal{U}'$; $UP(L^i_j), 1 \leq i \leq k, 2 \leq j \leq s$, will contain the best upwards edge encountered until now for a_j, that is the best upwards edge for a_j outgoing from T_{x_i}. Note that every edge is always stored together with its weight.

The field HOR of each list L^i_1, $1 \leq i \leq k$, denoted as $HOR(L^i_1)$, is a pointer to a possibly empty list of at most $k-1$ best horizontal edges connecting T_{x_i} and T_{x_h}, $1 \leq h \leq k$, $h \neq i$. The edges in the lists $HOR(L^1_1), \ldots, HOR(L^k_1)$ form the set $\mathcal{H}'$. Let $d(a_j)$ be the degree of a_j in T; the size of the lists $HOR(L^i_j)$, $1 \leq i \leq k$ and $1 \leq j \leq s$, is at most equal to $d(a_j) - 1$. For $j > 1$, such lists contain the best horizontal edges found until now for a_j, that is the best horizontal edges outgoing from T_{x_i} for a_j.

Some of the information sent to a node from its children is shown in Figure 1(a). Note that, since the horizontal edges are computed independently by each subtree, each edge will appear twice in the lists.

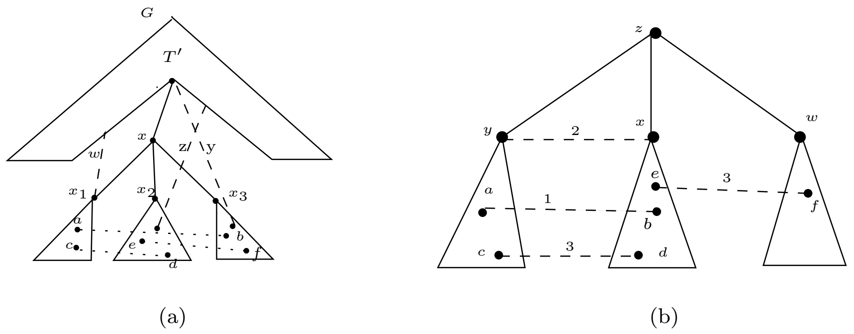

Fig. 1. (a) The upwards and horizontal edges sent to x by its children x_1, x_2, x_3 used for the computation of the replacement set for x. $UP(L_1^1)$ contains the upwards edge w, $UP(L_1^2)$ contains z and $UP(L_1^3)$ contains y. $\mathcal{U}' = \{w, y, z\}$ $HOR(L_1^1)$ contains the edges (a, b) and (c, d); $HOR(L_1^2)$ contains (d, c) and (e, f) and $HOR(L_1^3)$ contains (b, a) and (f, e). $\mathcal{H}' = \{(a, b)(c, d)(d, c)(e, f)(b, a)(f, e)\}$. (b) Selection of the horizontal edges for z in algorithm *MyAUH* executed by x.

Once a node has received the sets $\mathcal{U}'$ and $\mathcal{H}'$ from its children, it has to compute the MST of the contracted graph G_x. This can be done locally with an optimal sequential algorithm, with no exchange of additional messages. The only problem is that in the sets $\mathcal{U}'$ and $\mathcal{H}'$, the edges are indicated by their endpoints, while the nodes of the contracted graph G_x are the children and the parent of x. For this purpose, the endpoints of these edges must be relabeled.

Let us describe in detail the operations executed by node x. First of all x computes the new MST for the contracted graph $G_x = (V_x, E_x)$ by considering the lists transmitted to it from its children (Algorithm *MyMST*).

```
MyMST
(* Algorithm for node $x$*)
1. Construct the contracted graph $G_x = (V_x, E_x)$ of $G - x$ where
   $V_x = \{parent(x), children(x)\}$. $E_x$ is obtained by the union of the sets
   $\{UP(L_1^i), HOR(L_1^i)\}, 1 \le i \le k$, relabeled as follows: any edge $e = (a, b) \in$
   $UP(L_1^i)$ becomes $(x_i, parent(x))$. For any edge $e = (a, b) \in HOR(L_1^i)$ search
   the list $HOR(L_1^j), 1 \le i, j \le k, i \neq j$, containing the edge $e = (b, a)$ and rename
   $(a, b)$ as $(x_i, x_j)$.
2. Compute the $MST$ of $G_x$ locally with an optimal algorithm.
3. Reassign to the set of edges $ET_{G_x}$ of $T_{G_x}$ their original names besides the new
   names. $ET_{G_x}$ is the replacement set for $x$.
4. Send any edge $e = (a, b) \in ET_{G_x}$, relabelled as $(x_i, x_j)$ to child $x_i$.
```

Note that the relabeling operation is needed because even if node x knows the label i of the child from which it receives the information, an edge (a,b) coming from x_i does not explicitly specify to which subtree of x the node b belongs.

We now describe the algorithm of x which computes the *best upwards edge* for each ancestor $a_j, 2 \leq j \leq s$, among its incident upwards edges and the edges in $UP(L_j^i), 1 \leq i \leq k$. In addition x computes the *best horizontal edges* among its incident edges that are horizontal with respect to a_j and the edges in $HOR(L_j^i), 1 \leq i \leq k, 2 \leq j \leq s$ (Algorithm *MyAUH*).

Node x will produce the new $s-1$ lists $L_j^x, 2 \leq j \leq s$ to send to its parent. Note that while the best upward and horizontal edges that x computes for its parent are the final ones, the edges computed for all the other ancestors can be worse than the final ones; they will be ultimately computed for each node when *their children* execute Algorithm *MyAUH*.

Algorithm *MyAUH* makes use of the boolean function *anc(x,y)* which is *true* if and only if node x is an ancestor of y, and of the function *nca(x,y)* which returns the nearest common ancestor of x and y in a given tree, that is the common ancestor of x and y, whose distance from x and y is smaller than the distance of any other ancestor. Let $In(x)$ be the set of non tree edges incident to x. With respect to a node x, the horizontal edges connecting the same pair of subtrees of x will be called *analogous* in the following.

```
MyAUH

(* Algorithm for node $x$ *)

1. Among the edges in $In(x)$: select those for which $nca(x,y) = z$, $z \neq x$ and
   $z \neq y$; let $min$ be the one of minimum weight; For each ancestor node $a_j, 2 \leq$
   $j \leq s$: compute the best upwards edge as the one of minimum weight among
   $UP(L_j^i), 1 \leq i \leq k, min$, and the edges belonging to $In(x)$ such that $anc(a_j, x) =$
   $true$; store the best upwards edge in $UP(L_j^x)$.
2. Among the edges $e = (x,y) \in In(x)$: select those for which $nca(x,y) = a_j, 2 \leq$
   $j \leq s$. For each $j$ if there is a set of analogous edges, then choose the one of
   minimum weight. For each ancestor node $a_j, 2 \leq j \leq s$, $d(a_j) = d$, consider
   the selected incident edges $e = (x,y)$ such that $nca(x,y) = a_j$ and the edges
   $e = (h,h') \in HOR(L_j^i), 1 \leq i \leq k$; if there is a set of analogous edges then
   choose the one of minimum weight. All the selected edges are then stored in
   $HOR(L_j^x)$.
```

4 Correctness and Complexity

4.1 Basic Properties

We first introduce some properties needed to show how a node x can locally efficiently perform the operations in Algorithm *MyAUH*.

In order for a node to decide if the other endpoint of an incident edge is its ancestor it is sufficient to check the information collected in the broadcast phase.

Property 1. *Given $e = (x,y) \in In(x)$, $anc(y,x)$ can be checked at node x and no communication is needed.*

Property 1 derives from the fact that, after the broadcast phase, x knows all of its ancestors, and if y does not belong to the list of ancestors the function is false.

The *nearest common ancestor* of pairs of nodes $x, y \in T$, $nca(x, y)$ must be also computed. In a recent work [1], it has been shown that this information can be locally computed in constant time, through a proper labeling of the tree that requires labels of $O(logn)$ bits, denoted as $l(x)$, that can be precomputed by a depth first traversal of the tree. Therefore, our basic algorithm *ANR(G,T)* has to be slightly modified to transmit, for each node x, $l(x)$ instead of x. Once such labeling is computed for T, each node can be distinguished by its label. Then, from [1] and since $l(y)$ is accessible at x, we have:

Property 2. *Let $e = (x, y) \in In(x)$. $nca(x, y)$ can be computed at x and no communication is needed.*

In the selection of the horizontal edges we need to check whether two edges, having the same nearest common ancestor z, connect the same pair of subtrees of z, that is they are *analogous*: only the one with minimal weight, must be selected. In this way, node x selects at most one edge from T_x to any other subtree rooted in its siblings and this is important to bound the size of the information sent by every node.

The situation is depicted in Figure 1(b), where all horizontal edges (x, y), (b, a), (d, c), and (e, f) have the same nearest common ancestor z, but (x, y), (b, a), and (d, c) are analogous since they connect the same pair of subtrees T_x and T_y; only the one of minimum weight (b, a) is chosen; edge (e, f) is the unique connecting T_x and T_w, then is directly chosen. Besides the other information, x will then send to z the list $HOR(L_z^x)$ containing $(b, a)(e, f)$.

The problem is now how to detect the *analogy* between two horizontal edges. We have the following:

Lemma 1. *Let (a, b) and (c, d) be two edges such that $a \in T_y$, $c \in T_y$, and $nca(a, b) = nca(c, d) = z$. These edges are* analogous *if $nca(b, d) = x$, $x \neq z$. The condition can be checked at y for each z and no communication is needed.*

The proof of Lemma 1 can be followed observing Figure 2, where, for the edges (x, y) and (d, c), $nca(y, c)$ is different from z, hence they are analogous. Viceversa, for (x, y) and (e, f), $nca(y, f)$ is equal to z, hence the condition does not hold.

4.2 Analysis

We now prove the correctness of our basic algorithm ALL NODES REPLACEMENT *ANR(G,T)*. We have:

Theorem 1. *In algorithm* ANR(G,T) *each node $z \neq r$:*

(i) correctly computes the best upwards edge and the best horizontal edges for its parent.
(ii) determines for each ancestor a, different from the parent and the root, the best upward edges and the best horizontal edges for a in T_z.

We now establish the data complexity required by the algorithm. We recall that the preprocessing phase consists of a depth first search of the tree requiring $O(n)$ messages. We have:

Theorem 2. *The data complexity of algorithm* ANR(G,T)*is* $O(n^2)$.

The algorithm ALL NODES REPLACEMENT terminates leaving, in the children of each node, the edges to activate in case of failure. Let x be the node which fails, $x_1, ...x_k, 1 \leq i \leq k$ its children, and let RS_x be the replacement set of edges for x_i. Every x_i will contain the subset $RS_{x_i} \subseteq RS_x$ of edges having an endpoint in T_{x_i}; it starts a broadcast phase sending RS_{x_i} down in its subtree; in this phase the nodes that discover to be incident to one edge $e \in RS_{x_i}$ activate the edge. This activation phase requires a data complexity of order $O(d_{x-1} \times n)$, since at most d_{x-1} edges have to reach $O(n)$ nodes.

References

1. Alstrup, S., Gavoille, C., Kaplan, H., Rauhe, T.: Nearest common ancestor: A survey and a new distributed algorithm for a distributed environment. Theory of Computing System 37, 441–456 (2004)
2. Cheng, C., Cimet, I.A., Kumar, S.P.R.: A protocol to maintain a minimum spanning tree in a dynamic topology. Comput. Commun. Rev. 18(4), 330–338 (1988)
3. Chin, F., Houck, D.: Algorithms for updating minimal spanning trees. J. Comput. System Sci. 16(3), 333–344 (1978)
4. Das, B., Loui, M.C.: Reconstructing a minimum spanning tree after deletion of any node. Algorithmica 31, 530–547 (2001)
5. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Computing 21(6), 1184–1192 (1992)
6. Dixon, B., Tarjan, R.E.: Optimal parallel verification of minimum spanning trees in logarithmic time. Algorithmica 17(1), 11–17 (1997)
7. Faloutsos, M., Molle, M.: A linear-time optimal-message distributed algorithm for minimum spanning trees. Distributed Computing 17(2), 151–170 (2004)
8. Flocchini, P., Mesa Enriques, A., Pagli, L., Prencipe, G., Santoro, N.: Point of failure shortest-path rerouting. IEICE Trans. Inf. Syst. E89-D (2), 700–708 (2006)
9. Flocchini, P., Pagli, L., Prencipe, G., Santoro, N., Widmayer, P., Zuva, T.: Computing all the best swap edges distributively. In: Higashino, T. (ed.) OPODIS 2004. LNCS, vol. 3544, pp. 154–168. Springer, Heidelberg (2004)
10. Gaibisso, C., Proietti, G., Tan, R.B.: Optimal MST maintenance for transient deletion of every node in planar graphs. In: Warnow, T.J., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 404–414. Springer, Heidelberg (2003)
11. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum spanning tree. ACM Trans. Prog. Lang. and Systems 5(1), 66–77 (1983)
12. Johnson, D.B., Metaxas, P.: A parallel algorithm for computing minimum spanning trees. J. Algorithms 19, 383–401 (1995)
13. Nardelli, E., Proietti, G., Widmayer, P.: Nearly linear time minimum spanning tree maintenance for transient node failures. Algoritmica 40, 119–132 (2004)
14. Santoro, N.: Design and Analysis of Distributed Algorithms. Wiley, Chichester (2007)
15. Tsin, Y.H.: On handling vertex deletion in updating minimum spanning trees. Information Processing Letters 27(4), 167–168 (1988)