Distributed Minimum Spanning Tree Maintenance for Transient Node Failures

Paola Flocchini, T. Mesa Enriquez, Linda Pagli, Giuseppe Prencipe, Nicola Santoro

Abstract—In many network applications the computation takes place on the *minimum-cost* spanning tree (MST) of the network G; unfortunately, a single link or node failure disconnects the tree. The ALL NODES REPLACEMENT (*ANR*) problem is the problem of precomputing, for each node u in G, the new MST should u fail. This problem has been extensively investigated for serial and parallel settings, and efficient solutions have been designed for those environments. The situation is surprisingly different in distributed settings. In fact, no distributed solution exists to date which performs better than the brute-force repeated application of MST construction.
In this paper we consider for the first time the problem of computing all the replacement minimum-cost spanning trees *distributively*. We design a solution protocol and we prove that the total amount of communication exchanges taking place is $O(n)$, each exchange using at most $O(n)$ data items. Hence the total amount of data items communicated during the computation (the data complexity) is $O(n^2)$. We also show how the simpler problem ALL EDGES REPLACEMENT (*AER*) dealing with single edge failures, can be solved with the same costs using some existing techniques. Also for the *AER* problem, efficient solutions exist in the serial and parallel setting but, prior to this work, no distributed solution other than brute force was known.

Index Terms—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; or when it is not possible or convenient to maintain the indeces of the distribution of data around the network and every request is processed by broadcasting to everybody (Gnutella). 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 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 NODES REPLACEMENT (*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 EDGES REPLACEMENT (*AER*) 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 NODES REPLACEMENT problem in reality means to *pre-compute* the n replacement minimum spanning trees, one for each possible node failure in the tree [6], [12], [18]; 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 [4], [6], [12], [18], [21]).

In this paper we consider the *distributed* version of this problem. That is we investigate the ALL NODES REPLACEMENT problem when the computational entities are nodes of G themselves, and each can only communicate by exchanging messages with its neighbours. The

Research partially supported by NSERC Canada, and by MIUR of Italy under projects AlgoDEEP and MadWeb. A preliminary version of this work was presented at the 13^{th} International European Conference on Parallel and Distributed Computing (EuroPar).
Paola Flocchini is with the University of Ottawa, Canada. E-mail: flocchin@site.uottawa.ca
Toni Mesa Enriquez is with the Universidad de La Habana, Cuba, E-mail: tonymesa@matcom.uh.cu
Linda Pagli and Giuseppe Prencipe are with Università di Pisa, Italy, E-mail: {pagli,prencipe}@di.unipi.it
Nicola Santoro is with Carleton University, Canada. E-mail: santoro@scs.carleton.ca

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., [9], [13]) 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. Indeed, in their work on efficient serial and parallel solutions, Das and Loui state: *"... designing an efficient distributed algorithm for ANR remains an open problem ..."* [6].

1.2 Our Contributions

In this paper we consider the problem of computing all the replacement minimum-cost spanning trees *distributively*, and we efficiently solve both the ALL NODES REPLACEMENT (*ANR*) and ALL EDGES REPLACEMENT (*AER*) problems.

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) node failure. 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 [10], [11], 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.

We then consider the simpler ALL EDGES REPLACEMENT (*AER*) problem, where the goal is to update T whenever a single edge fails. We show that, not surprisingly, *AER* can be solved with the same complexity as *ANR*. Surprisingly, the approaches used for computing all the swap-edges of a shortest-path tree [10], [11], can be employed for the *AER* problem.

1.3 Related Work

Computation of swap links in case of link or node failures have been studied in the sequential setting for maintaining several spanning structures. For example, shortest paths tree have been studied in [1], [17], optimal tree spanners in [5], minimum diameter spanning trees in [14], [19]. Distributed algorithms have been devised for determining swap edges for shortest paths spanning trees in [10] and [11], and for minimum diameter spanning trees in [15].

Let us now turn to *minimum-cost* spanning trees and their maintenance in presence of link and node failures.

The ALL NODES REPLACEMENT (*ANR*) problem was first studied in a *serial* environment by Chin and Houck [4]. A more efficient solution has been developed by Das and Loui [6], and later improved by Nardelli, Proietti and Widmayer [18]. When G is *planar*, improved bounds have been obtained by Gaibisso, Proietti and Tan [12]. The simpler ALL EDGES REPLACEMENT (*AER*) is implicitly solved by Dixon, Rauch and Tarjan [7]; an improved solution was later developed by Nardelli, Proietti and Widmayer [18].

In the *parallel* setting, Tsin presented an algorithm to update a MST after a single node deletion [21]; 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 [16]. A more efficient parallel technique has been designed by Das and Loui [6]. The simpler *AER* problem is efficiently solved by using the parallel verification algorithm of Dixon and Tarjan [8].

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 [13]. This protocol is not only elegant but also optimal, since $\Omega(m + n \log n)$ messages are needed regardless of their size [20]. In fact, all subsequent work (e.g., [9]) 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 [3]; 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 existed for either problems.

2 TERMINOLOGY AND DEFINITIONS

Let $G = (V, E)$ be an undirected graph, with $n = |V|$ vertices and $m = |E|$ edges, where a non negative real *weight* $w(e)$ is associated to each edge e. A *subgraph* $G' = (V', E')$ of G is a graph 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* if it remains connected after the removal of any one of its edges; it is *2-node connected* if it remains connected after the removal of any one of its nodes.

Let $T = (V, E(T))$ be a spanning tree of graph G rooted in $r \in V$. A spanning tree $T = (V, E(T))$ is a *minimum spanning tree* of G, denoted by $MST(G)$, 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 the set of its *children* as $children(x)$. The ancestors of x in T, with the exception of the root, will be denoted as $\mathcal{A}(x)$. The boolean function $anc(x, y)$ is $true$ if and only if node x is an ancestor of y; the function $nca(x, y)$ 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.

Consider an edge $e = (x, y) \in E(T)$, with y closer to r; if such an edge is removed, the tree is then disconnected in two subtrees: T_x and $T \setminus T_x$. A *swap* edge for e is an edge $e' = (u, v) \in E \setminus \{e\}$, if any, that re-connects the two subtrees. It can be easily seen that the MST of the graph $G - e = (V, E \setminus \{e\})$, called *replacement tree*, can be computed by selecting the swap edge of minimum weight connecting T_x and $T \setminus T_x$. Given a node $x \in G$, we will call the *replacement set* of x, denoted by $RepSet_x$, the set of non-tree edges that need to be activated in case x fails. In the following, the horizontal edges connecting the same pair of subtrees of a given node x will be called *analogous*.

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 [20]).

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.

Let $G = (V, E)$ be a 2-node connected graph, T any minimum spanning tree of G, and x any node in V; if x is removed from T together with all of its incident edges, the tree disconnects into the subtrees $T_{x_1}, \dots, T_{x_k}$, where $x_1, \dots, x_k$ are the children of x.

Let $T' = T \setminus \{T_{x_1}, \dots, T_{x_k}, \{x\}\}$, x_0 be the parent of x, and E' be the set of non tree edges; we define the set of *upwards edges* of x as $\mathcal{U}_x = \{e = (u, v) \in E' | u \in T_{x_i}, 1 \leq i \leq k, v \in T'\}$; and the set of *horizontal edges* of x as $\mathcal{H}_x = \{e = (u, v) \in E' | u \in T_{x_i}, v \in T_{x_j}, 1 \leq i, j \leq k, i \neq j\}$. For node x, the set of its *best upward edges* $\overline{\mathcal{U}}_x \subseteq \mathcal{U}_x$ 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 its *best horizontal edges* is the set $\overline{\mathcal{H}}_x \subseteq \mathcal{H}_x$ 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 $\mathcal{U}$, $\overline{\mathcal{U}}$, $\mathcal{H}$, and $\overline{\mathcal{H}}$, when the reference to the removed node is clear from the context.

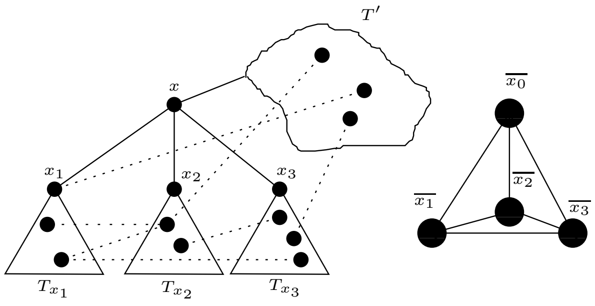

Fig. 1. An example of the contraction of $G - x$.

We now introduce the notion of *contracted graph*, which will be used in our algorithm. Given a node x, the contracted graph of $G - x$ is a graph $G_x = (V_x, E_x)$, with: $V_x = \{\overline{x_0}, \overline{x_1}, \dots, \overline{x_k}\}$, where $\overline{x_1}, \dots, \overline{x_k}$ represent the contraction of each subtree $T_{x_i}, 1 \leq i \leq k$ and $\overline{x_0}$ represents the contraction of T'; $E_x = \{(\overline{x_i}, \overline{x_j}) | \exists u \in T_{x_i}, v \in T_{x_j}, 1 \leq i \neq j \leq k : (u, v) \in \overline{\mathcal{H}}_x\} \cup \{(\overline{x_i}, \overline{x_0}) | \exists u \in T_{x_i}, v \in T', 1 \leq i \leq k : (u, v) \in \overline{\mathcal{U}}_x\}$ (see the example depicted in Figure 1). In the following we will refer to $\overline{x_i}$ as the *representative* of T_{x_i}, $0 \leq i \leq k$; also, we will refer to $(\overline{x_a}, \overline{x_b}) \in E_x$ as the *representative* of any minimum weight edge connecting T_{x_a} and T_{x_b}, $0 \leq a, b \leq k$.

The ALL NODES REPLACEMENT problem is that of computing all the replacement sets for each possible node failure, except for the root[1]. Our algorithm is described in Section 3.

The ALL EDGES REPLACEMENT problem is that of finding, given a 2-edge connected graph G and one of its minimum spanning trees T, the minimum spanning tree of $G - e$, for every edge $e \in T$. In Section 6 we describe a simple solution which can be derived by previous results.

3 SOLVING THE ANR PROBLEM

In the following, we will focus on the distributed solution of the ALL NODES REPLACEMENT problem. In particular, given a 2-node connected graph G, any minimum spanning tree T of G, and any node $x \in T$, in this section we will provide a distributed solution to the problem of computing the replacement set for the failure of node $x \in T$.

1. Notice that it is possible to compute the replacement set also for the root by running the algorithm a second time choosing a different node as the root.

At an 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 converge-cast phase. Node x determines also a set of edges, useful to compute the replacement sets for all of its ancestors.

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 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.

```
AllNodesReplacement($G$,$T$)
[Phase 1: Broadcast.]
  Label($T$);
  Each child of the root starts the broadcast by sending to its
  children a list containing its name and its label;
  Each node receiving a list of names from its parent, appends
  its name and its label to the received list and forwards it to
  its children.
    NOTE: From now on, when a node Sends a set of edges
    to any other node, we assume that it is sending the labels
    assigned to the endpoints of the edges, and computed by
    Label($T$).
[Phase 2: Convergecast.] $\backslash *$ with respect to node $x$ $*\backslash$
  $\mathcal{A}(x)$ = Set of the ancestors of $x$ in $T$, with the exception of
  the root;
  $s = |\mathcal{A}(x)|$;
  If I am a LEAF Then
    $\{(\mathtt{MyBestUp_j}, \mathtt{MyBestHor_j}),\ 1 \leq j \leq s\}$ =
    BestEdgesComp($x, \emptyset, \mathcal{A}(x)$);
    Send to my parent the edges in $\bigcup_j \mathtt{MyBestUp_j}$ and
    $\bigcup_j \mathtt{MyBestHor_j}$;
  If I am INTERNAL Then
    Wait until all the information computed from all $x$'s
    children are received;
    RecAnc = Set of all best upwards and horizontal edges
    received for $a_j$, $a_j \in \mathcal{A}(x)$ from $x$'s children;
    RecMe = Set of all best upwards and horizontal edges
    received for $x$ from $x$'s children;
    $\{(\mathtt{MyBestUp_j}, \mathtt{MyBestHor_j}),\ 1 \leq j \leq s\}$ =
    BestEdgesComp($x$,RecAnc,$\mathcal{A}(x)$);
    If My parent is not $r$ Then
      Send to my parent the edges in $\bigcup_j \mathtt{MyBestUp_j}$ and
      $\bigcup_j \mathtt{MyBestHor_j}$.
    $RepSet_x$ = ComputeRepSet($x$,RecMe);
    For All Children of $x$, $x_1, x_2, \ldots, x_k$ Do
      $RepSet_{x_i}$ = Subset of $RepSet_x$ containing edges hav-
      ing one endpoint in $T_{x_i}$;
      Send $RepSet_{x_i}$ to $x_i$.
```

The first phase of the algorithm begins with labeling the nodes of T according to the labeling technique introduced in [2]; such labeling allows the computation of the nearest common ancestor of any two given nodes whose labels are known. The labels have $O(\log n)$ bits size, and can be computed sequentially in $O(n)$ time. In a tree one can collect all the information about the tree at every node with $O(n^2)$ messages, and hence each node can locally compute the labels with the algorithms of [2]. Thus we will assume that these labelings are available when executing the broadcast. Then, this phase continues by broadcasting these labels, from the root to the leaves.

```
Routine BestEdgesComp($x$,RecAnc,$\mathcal{A}(x)$)
Input: RecAnc is the set of best upwards and best
  horizontal edges for $x$'s ancestors received from the
  children of $x$.
  For All Ancestors node $a_j \in \mathcal{A}(x)$ Do
    $\mathtt{RecUp_j}$ = Set of best upwards edges for $a_j$, $a_j \in$
    $\mathcal{A}(x)$, contained in RecAnc;
    $\mathtt{RecHor_j}$ = Set of best horizontal edges for $a_j$, $a_j \in$
    $\mathcal{A}(x)$, contained in RecAnc;
    $\mathtt{MyUp_j} = \min_{w(x,y)}\{(x,y) \in In(x) | anc(y,a_j) =$
    $true \vee anc(z,a_j) = true,\ with\ nca(x,y) = z,\ z \neq$
    $x \wedge z \neq y\}$;
    $\mathtt{MyHor_j}$ = NoAnalog($a_j, \{(x,y) \in$
    $In(x) | nca(x,y) = a_j\}$);
    If I am INTERNAL Then
      $\mathtt{MyBestUp_j} = \min_{w(e)}\{e \in \mathtt{RecUp_j} \cup \mathtt{MyUp_j}\}$;
      $\mathtt{MyBestHor_j}$ = NoAnalog($a_j, \mathtt{RecHor_j} \cup \mathtt{MyHor_j}$);
  Return $\bigcup_j\{(\mathtt{MyBestUp_j}, \mathtt{MyBestHor_j})$.
```

```
Routine NoAnalog($z, S$)
Input: $S$ is the set of edges to check for analogy with
  respect to node $z$.
  $\overline{S} = \emptyset$;
  $\{S_1, \ldots, S_q\}$ = Subsets of $S$ containing analogous
  edges with respect to node $z$;
  For $1 \leq i \leq q$ Do
    $e$ = Edge with minimum weight in $S_i$;
    $\overline{S} = \overline{S} \cup e$;
  $\overline{S} = \overline{S} \cup \{S \setminus \{S_1, \ldots, S_q\}\}$;
  Return $\overline{S}$.
```

After the broadcast phase, the converge-cast phase starts, from the leaves of T. In particular, each leaf x first computes locally the best upwards and horizontal edges among its non-tree incident links, with respect to all its ancestors node; this task is carried out in routine `BestEdgesComp()`. In particular, for each ancestor a_j of x, the sets $\mathtt{MyUp_j}$ and $\mathtt{MyHor_j}$ are computed. The first set contains the non-tree edge incident to x with minimum weight that is *upward* for a_j (in case more than one exists, one is chosen arbitrarily). The selection is performed considering the two possible cases (see the example depicted in Figure 2): an edge (x,y) is an upward edge for a_j either if y is an ancestor of a_j ($anc(y,a_j) = true$) or if a node z, with $z \neq x$ and $z \neq y$, is an ancestor of a_j and the nearest common ancestor of both x and y ($nca(x,y) = z$). The second set, $\mathtt{MyHor_j}$, contains the edges that are horizontal edges for a_j; it is easy to see that any non-tree edge (x,y) incident on x such that a_j is the nearest common ancestor of both x and y ($nca(x,y) = a_j$) is an horizontal edge for a_j; let us denote

```
Routine ComputeRepSet($x$,RecMe)
Input: RecMe is the set of best upwards and best horizontal edges for $x$ received from the children of $x$.
  Let $V_x = \{\overline{x_0}, \overline{x_1}, \ldots, \overline{x_k}\}$ be the representatives nodes for the contracted graph $G_x = (V_x, E_x)$ of $G - x$;
  $E_x = \emptyset$;
  For All Children of $x$, namely $x_1, x_2, \ldots, x_k$ Do
    $\overline{\mathcal{U}}_x(i)$ = Set of best upwards edges for $x$ received from $x_i$ contained in RecMe;
    $\overline{\mathcal{H}}_x(i)$ = Set of best horizontal edges for $x$ received from $x_i$ contained in RecMe;
    If $\overline{\mathcal{H}}_x(i) \neq \emptyset$ Then
      For All Edges $(a,b) \in \overline{\mathcal{H}}_x(i)$ Do
        Look for $1 \leq j \neq i \leq k$ such that edge $(b,a) \in \overline{\mathcal{H}}_x(j)$;
        If $(\overline{x_i}, \overline{x_j}) \notin E_x$ Then
          $E_x = E_x \cup (\overline{x_i}, \overline{x_j})$;
          Mark $(\overline{x_i}, \overline{x_j})$ as representing $(a,b)$;
    If $\overline{\mathcal{U}}_x(i) \neq \emptyset$ Then
      $(a,b)$ = Any edge in $\overline{\mathcal{U}}_x(i)$;
      $E_x = E_x \cup (\overline{x_i}, \overline{x_0})$;
      Mark $(\overline{x_i}, \overline{x_0})$ as representing $(a,b)$;
  $T_{G_x} = MST$ of $G_x$, computed locally at $x$ with any optimal algorithm (e.g., the one in [18]);
  $RepSet_x$ = Edges being represented by edges in $T_{G_x}$;
  Return $RepSet_x$.
```

this set as S_j. Here, a crucial step to bound the size of the information sent by every node in the converge-cast phase is to choose for each subset of analogous edges in S only the edge with minimum weight: this step is carried out by Routine NoAnalog(). The situation is depicted in the example reported in Figure 3, where the horizontal edges (x_1, x_2), (a,b), (c,d), and (e,f) have all the same nearest common ancestor x; however, (x_1, x_2), (a,b), and (c,d) connect the same pair of subtrees T_{x_1} and T_{x_2}; in this case, x_1 and x_2 would send to x only edge (a,b) as best horizontal edge for x. Also, x would receive edge (e,f) from x_3, being the only edge that connects T_{x_2} and T_{x_3}. In Section 4, we will show how to determine locally at x the analogous edges (Lemma 1). As its final act, x sends $\bigcup_j(\texttt{MyUp}_\texttt{j}$ and $\texttt{MyHor}_\texttt{j})$ to its parent.

Each internal node x waits to receive the sets computed by all of its children; then, it splits this set into two subsets: RecAnc, containing all best upwards and horizontal edges computed from x's children for nodes in $\mathcal{A}(x)$; and RecMe, containing all best upwards and horizontal edges computed from its children for x. Then, as done by the leaves, it invokes BestEdgesComp(). The computation of sets $\texttt{MyUp}_\texttt{j}$ and $\texttt{MyHor}_\texttt{j}$ is performed as detailed for the leaves; now, x further compares the edges in $\texttt{MyUp}_\texttt{j}$ and $\texttt{MyHor}_\texttt{j}$ with the edges contained in RecAnc: in particular, x computes the best horizontal and upwards edges for each of its ancestors $a_j \in \mathcal{A}(x)$ by comparing the edges received from its children and the one it just computed and placed in $\texttt{MyUp}_\texttt{j}$ and $\texttt{MyHor}_\texttt{j}$. Also here, Routine NoAnalog() is invoked to purge the set of best horizontal edge to send to a_j from possible analogous edges.

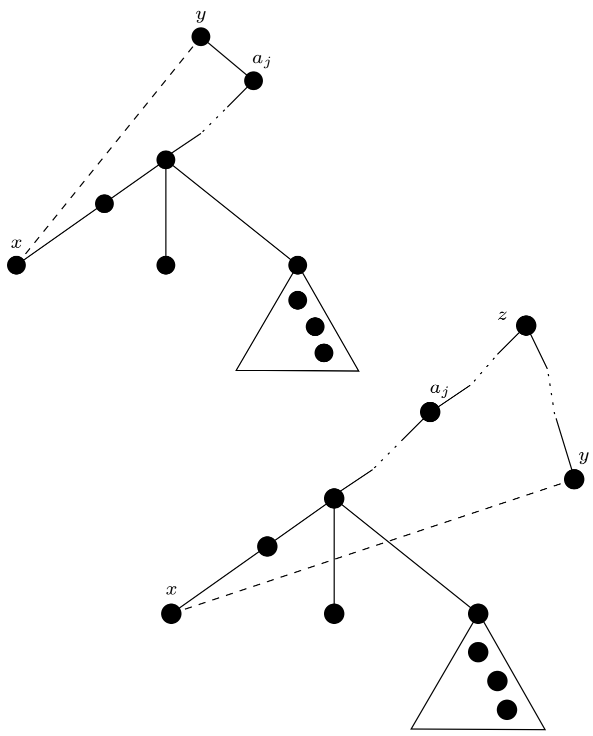

Fig. 2. The two cases considered in the computation of $\texttt{MyUp}_\texttt{j}$ and $\texttt{MyHor}_\texttt{j}$ performed by leaf x in routine BestEdgesComp().

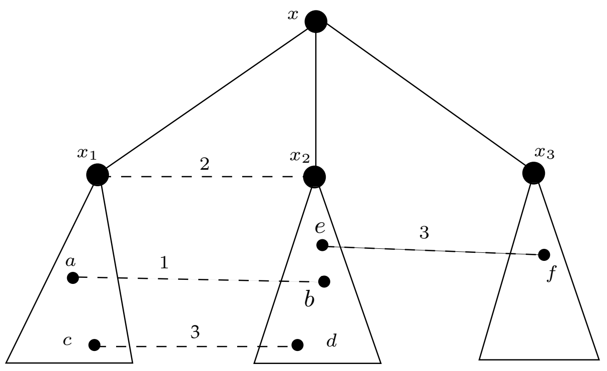

Fig. 3. Selection of the horizontal edges for x in algorithm BestEdgesComp(), when executed by x.

We here note that, the best horizontal and upwards edges that x computes for its parent (i.e., node a_1) are *final*: in fact, by the way the converge-cast phase is constructed, any upward edge for $parent(x)$ must have one end-point in T_x, and the other one in $T \setminus T_{parent(x)}$; all these edges have been considered by the nodes in T_x; similarly, this holds for the horizontal edges for $parent(x)$. In contrast, the horizontal and upwards edges computed by x for all its other ancestors can only be considerate as *candidate best*.

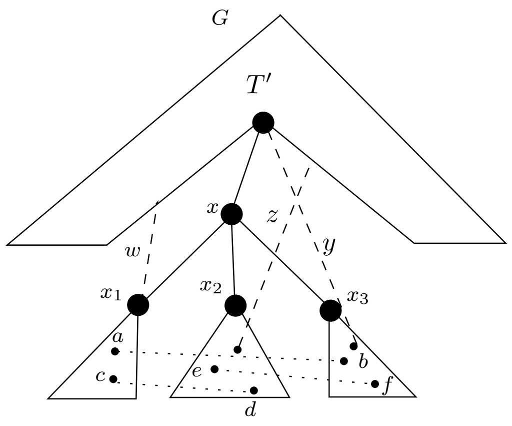

Fig. 4. The upwards and horizontal edges sent to x by its children x_1, x_2, x_3 used for the computation of the replacement set for x. Here, $\overline{\mathcal{U}}_x(1) = w$, $\overline{\mathcal{U}}_x(2) = z$, and $\overline{\mathcal{U}}_x(3) = y$; hence, $\overline{\mathcal{U}}_x = \{w, y, z\}$. Also, $\overline{\mathcal{H}}_x(1) = \{(a, b), (c, d)\}$, $\overline{\mathcal{H}}_x(2) = \{(d, c), (e, f)\}$, and $\overline{\mathcal{H}}_x(3) = \{(b, a), (f, e)\}$; hence, $\overline{\mathcal{H}}_x = \{(a, b)(c, d)(d, c)(e, f)(b, a)(f, e)\}$.

Then, the internal node x uses the information received in `RecMe` from its children to compute, locally, the contracted graph of $G - x$, $G_x = (V_x, E_x)$. From [18] we know that the MST of graph $G - x$ can be computed through the computation of the MST of the contracted graph; we use the technique of [18] to compute the MST of $G - x$ and then the replacement set for x. These operations are carried out in Routine `ComputeRepSet()`. In particular, each node in V_x is a representative node for the parent of x, $\overline{x_0}$, and for the children of x, $\{\overline{x_1}, \ldots, \overline{x_k}\}$. Then, the set E_x is computed from `RecMe`, as follows. Let $\overline{\mathcal{U}}_x(i)$ and $\overline{\mathcal{H}}_x(i)$ be the set of best upwards and horizontal edges for x received from the i-th children of x (clearly, both $\overline{\mathcal{U}}_x(i)$ and $\overline{\mathcal{H}}_x(i)$ are subsets of `RecMe`). If (a, b) is any edge in $\overline{\mathcal{H}}_x(i)$, then the edge (b, a) must be part of $\overline{\mathcal{H}}_x(j)$, for some $1 \le j \ne i \le k$ (see the example depicted in Figure 4). Then, there exist an edge between nodes $\overline{x_i}$ and $\overline{x_j}$ in G_x. If (a, b) is any edge in $\overline{\mathcal{U}}_x(i)$, there exist an edge between nodes $\overline{x_i}$ and $\overline{x_0}$ in G_x. After G_x has been locally computed, node x can compute its MST, by executing any optimal sequential algorithm (as the one in [18]). Note that the computation of G_x and its MST is entirely performed locally at x, with no exchange of additional messages.

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 Routine `BestEdgesComp()`.

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.

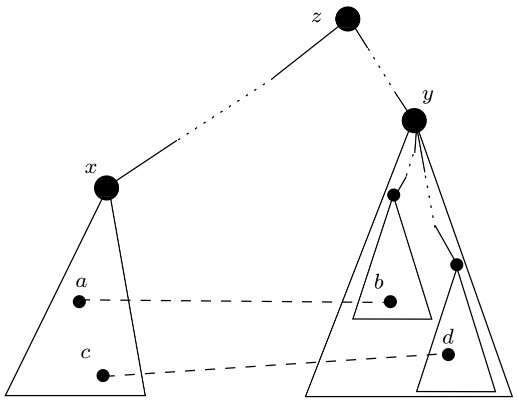

Fig. 5. Proof of Lemma 1.

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

Proof: The property 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 $x, y \in T$, $nca(x, y)$, is also used in Routine `BestEdgesComp()`. In order to be able to compute the $nca()$ function, we make use of the labeling of the nodes defined in the first phase of the algorithm; given any node $x \in T$, let $l(x)$ denote its label. Directly from [2], we have the following:

Theorem 1 ([2]). *Given any pair of nodes $a \in T$ and $b \in T$, if any node $x \in T$ locally knows the values of $l(a)$ and $l(b)$, then $nca(a, b)$ can be locally computed at x, and no additional communication is needed.*

The last open issue is how to detect the *analogy* between two horizontal edges; this is used in Routine `NoAnalog()` (refer to Figure 5).

Lemma 1. *Let (a, b) and (c, d) be two edges such that $a \in T_x$, $c \in T_x$, and $nca(a, b) = nca(c, d) = z$. These edges are* analogous *if $nca(b, d) = y$, $y \ne z$. Furthermore, if x locally has the values of $l(a)$, $l(b)$, $l(c)$, and $l(d)$, then x can locally check the analogy condition, and no communication is needed.*

Proof: Since (a, b) and (c, d) have the same nca, z, we have that either $nca(b, d) = z$, or $nca(b, d) = y$, $y \ne z$, with z an ancestor of y. The latter case means that nodes c and d belong to the same subtree of z, hence they are analogous. Finally, by Theorem 1 and since by hypothesis x locally has the values of $l(a)$, $l(b)$, $l(c)$, and $l(d)$, the analogy condition can be checked locally at x, and the lemma follows □

The proof of Lemma 1 can be followed also observing Figure 3: (x_1, x_2) and (c, d) are analogous, since $nca(x_1, x_2) = nca(c, d) = x$, and $nca(x_1, c) = x_1 \ne x$. In contrast, (x_1, x_2) and (e, f) are not analogous: in fact, $nca(x_2, f) = x$.

4.2 Analysis

We now prove the correctness of algorithm AllNodesReplacement(G,T). We have:

Theorem 2. *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 $a \in \mathcal{A}(z)$, with $a \neq parent(z)$ and $z \neq r$, the best* candidate *upward edges and the best horizontal edges for a.*

Proof: First observe that, as result of the broadcast phase, every node receives the label of all of its ancestors. The proof is by induction on the height $h(z)$ of the subtree T_z.

Basis. $h(z) = 0$; i.e., z is a leaf.

In this case, by Routine `BestEdgesComp()`, Property 1 and Theorem 1, z can locally determine, for all of its ancestors with the exception of the root, the best upwards edge (if any) and the best horizontal edges (if any), among its incident edges, for its parent, thus proving (i). Also, it can locally compute the best *candidate* upwards edge (if any) and the best horizontal edges (if any), among its incident edges, for all of its other ancestors, with the exception of r, thus proving (ii).

Induction step.

Let the theorem hold for all nodes z with $0 \leq h(z) \leq n-1$; we will now show that it holds for any z, with $h(z) = n$. By inductive hypothesis, node z receives from children z_i the best candidate upwards an the best horizontal edges for itself and for all the other ancestors; the collection of this information from all of its children forms the sets $\overline{\mathcal{U}}_z$ and $\overline{\mathcal{H}}$ (these two sets are used in Routine `ComputeRepSet()` to compute the MST of G_z).

By the note highlighted in Algorithm AllNodesReplacement(G,T), by Routine `BestEdgesComp()`, Property 1 and Theorem 1, node z can thus locally determine, based on the sets of edges received from its children, the best upwards edge and the best horizontal edges for its parent (case INTERNAL in Routine `BestEdgesComp()`), thus proving (i). Analogously, z can locally compute the best *candidate* upwards and horizontal edges for all the other ancestors, with the except of the root.

□

Theorem 3. *Algorithm* AllNodesReplacement(G,T) *correctly solves the* All Nodes Replacement *problem.*

Proof: We need to prove that, at the end of the execution of the algorithm, each node z correctly computes its replacement set $RepSet_z$: clearly, this needs to be proven only for internal nodes in T.

By Theorem 2, node z correctly computes $\overline{\mathcal{U}}_z$ and $\overline{\mathcal{H}}_z$, that are used in Routine `ComputeRepSet()` to compute the MST of G_z, T_{G_z}; by construction, each edge in T_{G_z} represents a non-tree edge that is in the replacement set of z, and the theorem follows. □

In the following theorem, we will establish the data complexity required by the algorithm.

Theorem 4. *The data complexity of algorithm* AllNodesReplacement(G,T) *is $O(n^2)$.*

Proof: In the broadcast phase, every node except the root receives the label of all its ancestors. In the convergecast phase, every node z sends a message containing information on $\sum_{j=1}^{s}(d_j)$ edges, where d_j is the degree of ancestor $a_j \in \mathcal{A}(z)$; in fact, one upwards edge and up to $d-1$ horizontal edges are sent from from z for each of its ancestors. Hence, z sends out information on at most $2n-1$ edges. Summing up over all n nodes in T, we have that the overall data complexity is $O(n^2)$. □

The message complexity of Theorem 4 may seem quite high; however we note that, as mentioned in Section 1.3, the optimal protocol to compute the MST already takes $O(m + n\log n)$ messages [13], and the proposed algorithm is better than the simple algorithm consisting in reconstructing the tree at each node.

Finally, note that the communication can be achieved with $O(n)$ messages if the system allowed the transmission of long messages (i.e., containig $O(n)$ data items); otherwise $O(n^2)$ short messages suffice.

5 Routing with the pre-computed data

Let us now address the problem of how the routing tables are organized and how the information stored there must be used in presence of failure of a node; let $x \in T$ be the node that fails.

The routing table at node x contains, for each destination r, the neighbor y in the shortest path from x to r (as determined in the MST of G, T_r). Let $x_1, \ldots, x_k$ be the children of x in T_r; as its final act, Algorithm AllNodesReplacement$((G,T))$ terminates leaving in the children of x the edges to activate in case of failure of x, that is the set $RepSet_{x_i}$. This information is stored as well in the routing table of x_i, and will be indicated as $RT[x_i,r].rep$ in the following.

Consider now a message M with destination r arriving to one of the children of x, x_i, where however x has just failed. The following steps are performed in this case (all the operations are relative to T_r):

1) the replacement set for x, $RepSet_{x_i}$, is retrieved by x_i from $RT[x_i,r].rep$;
2) x_i *activates* the edge in $RepSet_{x_i}$ whose one endpoint is x_i;
3) x_i starts a broadcast phase, sending RS_{x_i} down in the subtree T_{x_i};
    a) in this phase, the nodes reached by this message that discover to be incident to one edge $e \in RS_{x_i}$ *activate* this edge;

4) x_i sends M on the edge it activated;
   a) when a node descendant of x_i receives M, it forwards it along the edge it activated.
   b) when a non-descendant node of x_i receives M, it forwards it towards r according to the standard routing table information.

Note that a node in T that receives M realizes to be a *non-descendant* node of x_i from the fact that it does not have any edge *active*.

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.

6 Solving the AER Problem

The All Edges Replacement problem can be solved distributively by applying one of the algorithmic shells of [11], 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 data complexity is then the same as in [11] 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 $O(n^2)$. Also in this case, the communication can be achieved with the exchange of $O(n)$ long messages, if the system so allows; otherwise, $O(n^2)$ short messages suffice.

7 Concluding Remarks

The proposed protocols allow the nodes to find all the replacement minimum-cost spanning trees, one for each possible node or link failure. Considering the amount of information that must be computed, the total cost is surprisingly low, ($O(n^2)$ data complexity). Furthermore, the overall communication structure of the protocol is relatively simple: a broadcast followed by a convergecast.

An interesting open problem is to determine whether or not it is possible to improve the $O(n^2)$ upper bound on the data complexity of the problem.

References

[1] A. Di Salvo, G. Proietti. Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. *Theoretical Computer Science* 383(1): 23-33, 2007.

[2] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestor: a survey and a new algorithm for a distributed environment. *Theory of Computing System*, 37: 441–456, 2004.

[3] C. Cheng, I.A. Cimet, S.P.R. Kumar. A protocol to maintain a minimum spanning tree in a dynamic topology. *Comput. Commun. Rev.*, 18(4): 330–338, 1988.

[4] F. Chin and D. Houck. Algorithms for updating minimal spanning trees. *J. Comput. System Sci.*, 16(3): 333–344, 1978.

[5] S. Das, B. Gfeller, P. Widmayer. Computing best swaps in optimal tree spanners. *19th Int. Symposium on Algorithms and Computation (ISAAC)* 716-727, 2008.

[6] B. Das and M.C. Loui. Reconstructing a minimum spanning tree after deletion of any node. *Algorithmica*, 31: 530–547, 2001.

[7] B. Dixon, M. Rauch, R.E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Computing*, 21(6): 1184–1192, 1992.

[8] B. Dixon and R.E. Tarjan. Optimal parallel verification of minimum spanning trees in logarithmic time. *Algorithmica*, 17(1): 11-17, 1997.

[9] M. Faloutsos and M. Molle. A linear-time optimal-message distributed algorithm for minimum spanning trees. *Distributed Computing* 17 (2): 151-170, 2004.

[10] P. Flocchini, A. Mesa Enriques, L. Pagli, G. Prencipe, and N. Santoro. Point of failure shortest-path rerouting. *IEICE Trans. Inf. Syst.*, E89-D (2): 700–708, 2006.

[11] P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, and P. Widmayer. Computing all the best swap edges distributively. *Journal of Parallel and Distributed Computing*, 68(7): 976-983, 2008.

[12] C. Gaibisso, G. Proietti, and R.B. Tan. Optimal MST maintenance for transient deletion of every node in planar graphs. *Proc. of International Conference on Computing and Combinatorics* (COCOON'03), LNCS 2697, 404–414, 2003.

[13] R.G. Gallager, P.A. Humblet, and P.M. Spira, A distributed algorithm for minimum spanning tree. *ACM Trans. Prog. Lang. and Systems* 5 (1): 66–77, 1983.

[14] B. Gfeller. Faster swap edge computation in minimum diameter spanning trees. *16th Annual European Symposium on Algorithms* (ESA) 454-465, 2008.

[15] B. Gfeller, N. Santoro, P. Widmayer. A distributed algorithm for finding all best swap edges of a minimum diameter spanning tree. *IEEE Transactions on Dependable and Secure Computing*, to appear, 2011.

[16] D.B. Johnson and P. Metaxas. A parallel algorithm for computing minimum spanning trees. *J. Algorithms* 19 : 383-401, 1995.

[17] E. Nardelli, G. Proietti, P. Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica*, 35(1): 56-74, 2003.

[18] E. Nardelli, G. Proietti, and P. Widmayer. Nearly linear time minimum spanning tree maintenance for transient node failures. *Algoritmica*, 40:119–132, 2004.

[19] E. Nardelli, G. Proietti, and P. Widmayer. Finding all the best swaps of a minimum diameter spanning tree under transient edge failures. *Journal of Graph Algorithms and Applications*, 2(1):123, 1997.

[20] N. Santoro. *Design and Analysis of Distributed Algorithms*. Wiley, 2007.

[21] Y.H. Tsin. On handling vertex deletion in updating minimum spanning trees. *Information Processing Letters*, 27(4):167–168, 1988.