Università di Pisa
Dipartimento di Informatica

Technical Report: TR-04-18

Computing All the Best Swap Edges Distributively

P. Flocchini L. Pagli G. Prencipe N. Santoro P. Widmayer
T. Zuva

November 1, 2004

ADDRESS: via F. Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Computing All the Best Swap Edges Distributively *

P. Flocchini[†] L. Pagli[‡] G. Prencipe[‡] N. Santoro[§] P. Widmayer[¶]
T. Zuva[‖]

November 1, 2004

Abstract

Recently great attention has been given to *point-of-failure swap rerouting*, an efficient technique for routing in presence of transient failures. According to this technique, a message follows the normal routing table information unless the next hop has failed; in this case, it is redirected towards a precomputed link, called *swap*; once this link has been crossed, normal routing is resumed. The amount of precomputed information required in addition to the routing table is rather small: a single link per each destination. Several efficient serial algorithms have been presented to compute this information; none of them can unfortunately be efficiently implemented in a distributed environment. In this paper we present protocols, based on a new strategy, that allow the efficient computation of all the optimal swap edges under several optimization criteria.

1 Introduction

In systems using shortest-path routing tables, a single link failure is enough to interrupt the message transmission by disconnecting one or more shortest-path spanning trees. The on-line recomputation of an alternative path or of the entire new shortest path trees, rebuilding the routing tables accordingly, is rather expensive and causes long delays in the message's transmission [5, 10]. Hopefully, some of these costs will be reduced if the serial algorithms for dynamic graphs (e.g., those of [1]) could be somehow employed; to date, the difficulties of finding an efficient distributed implementation have not been overcome (e.g., see [9]).

An alternative approach is to precompute additional information and use it to augment the shortest-path routing tables so to make them operate when a failure occurs. Examples of this approach are techniques (e.g., see [4]) of pre-computing several edge-disjoint spanning trees for each destination. However, the alternative routes do not satisfy any optimization criterion (such as shortest path) even in the case when, at any time, only one link (not necessarily the same at all times) might be down.

A new strategy has been recently proposed [2, 5, 7, 8, 11]. It starts from the idea of precomputing, for each link in the tree, a single non-tree link (the *swap edge*) able to reconnect the network should the first fail. The strategy, called *point-of-failure swap rerouting* is simple: normal routing information will be used to route a message to its destination. If, however, the next hop is down, the message is first rerouted towards the swap edge; once this is crossed, normal routing will resume. Experimental results [11] show that the tree obtained from the swap edge is very close to the new shortest-path spanning tree computed from scratch.

Clearly, some swap edges are preferable to others. In [8], four main objective functions were defined, giving rise to four different problems. These functions have the goal to find a new tree that

*Research partially supported by "Progetto ALINWEB", MIUR, Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale, NSERC Canada, and the Swiss BBW 03.0378-1 for EC contract 001907 (DELIS).

[†]University of Ottawa, Canada, flocchin@site.uottawa.ca
[‡]Università di Pisa, Italy, {pagli, prencipe}@di.unipi.it
[§]Carleton University, Canada, santoro@scs.carleton.ca
[¶]ETH, Zurich, Switzerland, widmayer@inf.ethz.ch
[‖]University of Botswana,Gaborone, zuvat@mopipi.ub.bw

minimizes, respectively, the distance between the point of failure to the root (F_{dist}); the sum of distances (F_{sum}), the largest increment in the distance (F_{incr}), and the largest distance (F_{max}) of all nodes below the point of failure to the root.

In [8] they showed that these problems can be solved sequentially with different complexities: F_{dist} and F_{incr} in $O(m \cdot \alpha(m,n))$, F_{sum} in $O(n^2)$, and F_{max} in $O(n\sqrt{m})$, where $\alpha(m,n)$ is the functional inverse of Ackermann's function. These bounds are achieved using Tarjan's sophisticated technique of *transmuters* [12]. Unfortunately, there is currently no efficient distributed implementation of this sequential technique. From a distributed point of view, only the first of those problems, F_{dist}, has been investigated and solved. A simple but non-optimal solution has been developed in [5]. An efficient optimal solution has been recently proposed [3]. No efficient distributed solution exists to date for the problems F_{sum}, F_{incr}, and F_{max}. These problems appear to be rather important, since they minimize the average, the additional and the maximum delivery time of a message issued at any node. In this paper, we will be able to solve efficiently all three problems.

We propose two general distributed strategies, each solving the three problems with simple modifications. The first scheme uses $O(n_r^*)$ short messages, where n_r^* is the size of the transitive closure of $T_r \setminus \{r\}$; note that $0 \leq n_r^* \leq (n-1)(n-2)/2$. In the second scheme the number of messages decreases to $O(n)$ if long (i.e., $O(n)$ bits) messages are allowed. Both schemes use an overall complexity of $O(n_r^*)$.

All proofs are reported in the Appendix.

2 Terminology and Problems

Let $G = (V, E)$ be a 2-connected 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 *length* $w(e)$ is associated to each edge e. We say that the length of a path is the sum of the lengths of its edges, and the *distance* $d(x, y)$ between two vertices x and y is the length of a shortest path between them. Let $T = (V, E(T))$ be a spanning tree of graph G rooted in r. Let $T_q = (V(T_q), E(T_q))$ denote the subtree of T rooted in q.

Consider an edge $e = (x, y) \in E(T)$ with y closer to r; if such an edge is removed, the tree is disconnected in two subtrees: T_x and $T \setminus T_x$. A *swap* edge for $e = (x, y)$ is any edge $e' = (u, v) \in E \setminus \{e\}$ that connects the two subtrees and forms a new tree $T_{e/e'}$, called swap tree.

Let $\mathcal{S}_e$ be the set of all possible swap trees with respect to e. Depending on the goal of the swapping algorithm, some swap edges are preferable to others. Given an objective function F over $\mathcal{S}_e$, an *optimal* or *best* swap edge for a link e is a swap edge e' such that $F(T_{e/e'})$ is minimum.

Let $d_T(u, v)$ (shortly $d(u, v)$) denote the distance between nodes u and v in T, and let $d_{T_{e/e'}}(u, v)$ (shortly $d_{e/e'}(u, v)$) denote their distance in $T_{e/e'}$. Given a subtree T_w of T, we denote by $W(T_w)$ $= \sum_{t \in V(T_w)} d(t, w)$ the *weight* of T_w, and by $n(T_w)$ the number of nodes in T_w.

Given a rooted tree S, let $C(x, S)$ denote the set of children of node x in tree S, let $p(x, S)$ be the parent of node x in S, and $A(x, S)$ denote the ancestors of x in S. When $S = T$ we will simply write $C(x)$, $p(x)$ and $A(x)$. We consider the main problems studied in [8]:

1) F_{sum}**-problem:** $\min_{T_{e/e'} \in \mathcal{S}_e}\{F_{sum}(T_{e/e'})\}$, where $F_{sum}(T_{e/e'}) =$ $\sum_{t \in V(T_x)} d_{e/e'}(t, r)$. Choose one of the swap edges e' that minimizes the sum of the distances $F_{sum}(T_{e/e'})$ from all nodes in T_x to r.

2) F_{incr}**-problem:** $\min_{T_{e/e'} \in \mathcal{S}_e}\{F_{incr}(T_{e/e'})\}$ where $F_{incr}(T_{e/e'}) =$ $\max_{t \in V(T_x)}(d_{e/e'}(t, r) - d(t, r))$. Choose the swap edge that minimizes the maximum increment of the distance from r to any node in T_x.

3) F_{max}**-problem:** $\min_{T_{e/e'} \in \mathcal{S}_e}\{F_{max}(T_{e/e'})\}$ where $F_{max}(T_{e/e'}) =$ $\max_{t \in V(T_x)} d_{e/e'}(t, r)$. Choose the swap edge that minimizes the maximum distance from the nodes in T_x to r.

As an example, consider the 2-connected weighted graph and its shortest-path spanning-tree rooted in A shown in Figure 1.(a). The best swap edge for link (D, B) is (E, C) when considering F_{sum} or F_{incr}; (F, B) (D, C) and (E, C) are best swap edges if using F_{max}.

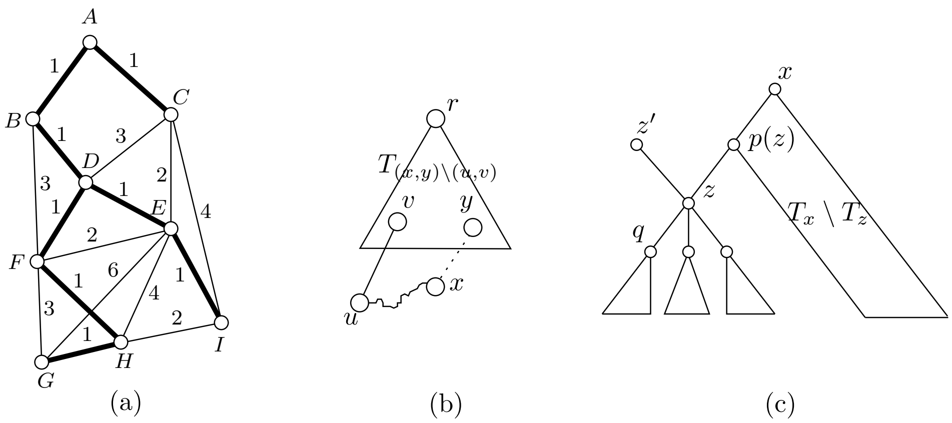

Figure 1: (a) An example: the thick line represents the starting SPT, rooted in A. (b) Structure of the subtree T_x with respect to the swap edge (z, z'). (c) Properties 2–3.

3 Algorithmic Shell and Computational Tools

3.1 A Generic Algorithm

Consider the problem of computing the best swap edge for link $e = (x, p(x)) \in E(T)$, where $p(x)$ denotes the parent of x in T. We now present a generic distributed algorithm to perform this computation; the details of its modules depend on the objective function F and will be described later.

The algorithm is started by x; during its execution each node $z \in V(T_x)$ will determine the best, according to the objective function, local swap edge (z, z') for $(x, p(x))$. Among the local swap edges of all nodes, the swap edge yielding the global minimum cost will be then selected. More precisely, we define:

PROCEDURE BSE($F, (x, p(x)$)
- Node x determines, among its local swap edges for $(x, p(x))$, the one that minimizes F. As we will see, x is the only node that can do so without any additional information.
- After this, x sends to each child the *enabling information* it needs to compute the best among its local swap edges for $(x, p(x))$.
- Upon receiving the enabling information from its parent, a node computes the best among its local swap edge for $(x, p(x))$; it then sends enabling information to its children. This process terminates once the leaves of T_x are reached.
- The leaves then start a *minimum finding* process to determine, among the swap edges chosen by the nodes in T_x, the one that minimizes the objective function F.
- The optimal swap edge for $(x, p(x))$ is thus determined at node x.

This procedure finds the best swap edge for link $(x, p(x))$ (according to F). Thus, the generic algorithm to find all the best swap edges is

ALGORITHM BEST F-SWAP
1. PRE-PROCESSING(F)
2. $\forall x \neq r$: BSE $(F, (x, p(x)))$

where PRE-PROCESSING(F) is a preliminary process to be executed only if the nodes do not have the required initial information.

3.2 Identifying Swap Edges

Before proceeding with the instantiation of the generic algorithm for each of the objective functions, we describe a tool that allows a node to distinguish, among its incident edges, the ones that are swap edges for a given edge $(x, p(x))$.

Consider the following labeling of the nodes $\lambda : V \rightarrow \{1, \ldots, n\}^2$. Given T, for $x \in V$ let $\lambda(x) = (a, b)$, where a is the numbering of x in the *preorder* traversal of T; and b is the numbering of x in the *inverted preorder* traversal of T, i.e., when the order of the visit of the children is inverted. The pairs given by the labeling form a partial order $(\lambda, \geq)$ of dimension 2 (let $\lambda(z) = (z_1, z_2)$ and $\lambda(w) = (w_1, w_2)$, then $\lambda(z) \geq \lambda(w)$ if $z_1 \geq w_1$ and $z_2 \geq w_2$). The "dominance" relationship between these pairs completely characterizes the relationship "descendant" in the tree:

Property 1 *A node z is descendant of a node w in T* if and only if $\lambda(z) \geq \lambda(w)$.

In our algorithms, we assume that each node z knows its own pair $\lambda(z)$ as well the pairs of its neighbors. If not available, this information can be easily acquired by having each node exchange the information with its neighbors. Such a labeling will be given to the tree in a preprocessing phase. Based on Property 1, we can now see how the labeling can be used by a node u to recognize its incident swap edges for a given link $(x, p(x))$ (refer to Figure 1.(b)).

Property 2 *An edge $(u, v) \in E \setminus E(T)$ is a swap for $(x, p(x)) \in E(T)$* if and only if *only one of u and v is a descendant of x in T.*

Thus, node $u \in T_x$ will be able to tell whether its incident edge (u, v) is a swap edge for $(x, p(x))$ simply by comparing $\lambda(v)$ with $\lambda(x)$; if $\lambda(v) \geq \lambda(x)$, then (u, v) is not a swap edge for $(x, p(x))$.

4 The F_{sum}-problem.

In Problem F_{sum}, the *optimal swap edge* for link $e = (x, p(x))$ is one which minimizes the sum of the distances from all nodes in T_x to the root r, in the new spanning tree $T' = T_{e/e'}$. A swap edge (u, v) solving F_{sum} will also minimize the *average distance* of all the nodes belonging to T_x from the root r, since the size of T_x is the same for all the swap edges for x.

For solving the F_{sum}-problem (known also as *average stretch factor* [2]), we require each node z to possess the following a-priori information: its distance $d(z, r)$ from the root; the sum of the distances of all nodes in T_q to z for each of the children q of z; and the number of nodes $n(T_q)$ in T_q for each of its children q. If this information is not initially available, it can be easily acquired by the nodes in a pre-processing phase, composed of a simple convergecast in T, executed only once at the beginning of the algorithm, and described in the Appendix.

Let z be a node in T_x that needs to compute the cost of a candidate swap edge $e' = (z, z')$ for e. Let $T' = T_{e/e'}$. Let $sum(T_x, (z, z'))$ denote the sum of distances from all nodes of T_x to z', i.e., $sum(T_x, (z, z')) = W(T'_z) + n(T'_z) \cdot w(z, z')$.

Lemma 1 *The sum of the distances in T' from all nodes in T_x to r is:*

$$F_{sum}(T') = W(T'_z) + n(T'_z) \cdot w(z, z') + n(T_x) \cdot d(z', r).$$

Notice that the subtrees of node z in T and in T' are different; in particular, the children of z in T' consists of all the children of z in T plus the parent of z in T (see Figure 1.(c)). With this observation in mind, we can write: $W(T'_z) = W(T_z) + sum(T_x \setminus T_z, (p(z), z))$ and $n(T'_z) = n(T_z) + n(T_x \setminus T_z)$. It is then clear that x can locally compute the cost of its candidate swap edges (since it has no parent in T_x); on the other hand, any other node z in T_x requires some additional information.

To instantiate algorithm BSE for F_{sum} we have to specify what is the *enabling information* to be propagated. The enabling information that any node z has to send down to its child q is composed of: the sum $sum(T_x \setminus T_q, (z, q))$ of the distances from q to the nodes in the subtree $T_x \setminus T_q$; and the number $n(T_x \setminus T_q)$ of nodes in this subtree.

The algorithm for finding the best swap edge for $(x, p(x))$ according to F_{sum} is as follows:

BSE($F_{sum}, (x, p(x))$)

(* Algorithm for node z *)

1. *If* $z = x$
   - Compute cost of each local candidate swap edge:
     (for each $e' = (x, x')$, $F_{sum}(T_{e/e'}) = sum(T_x, (x, x')) + n(T_x) \cdot d(x', r)$)
   - select best candidate
   - for each child q: compute the enabling information $sum(T_x \setminus T_q, (x, q))$ and $n(T_x \setminus T_q)$ and send it to q. It will be shown that this information can be computed locally.
   - wait for the result of *minimum finding*; determine the best swap edge for $(x, p(x))$
2. *Else* $\{z \neq x\}$ – *Receiving enabling info* (s, n) *for* $(x, p(x))$.
   - Compute cost of each local candidate swap edge:
     (for each $e' = (z, z')$, $T_{e/e'}$: $= s + sum(T_z, (z, z')) + (n + n(T_z)) \cdot d(z', r)$).
     It will be shown that this information can be computed locally.
   - select best candidate
   - if I am a leaf: start *minimum finding*
   - if I am not a leaf
     - for each child q: compute the enabling information $sum(T_x \setminus T_q, (z, q))$, and $n(T_x \setminus T_q)$ and send it to q.
     - participate in *minimum finding* (wait for info from all children, select the best and send to parent)

Lemma 2 *Let $e = (x, p(x))$. Each node $z \in T_x$ can correctly compute:* 1) *the best local swap edge for e,* 2) *the value $sum(T_x \setminus T_q, (z, q))$ for each $q \in C(z)$,* 3) *the value $n(T_v \setminus T_q)$ for each $q \in C(z)$.*

5 The F_{max} and F_{incr} Problems

In Problem F_{max}, the *optimal swap edge* e' for link $e = (z, p(z))$ is any swap edge such that the longest distance of all the nodes in T_z from the root r is minimized in the new spanning tree $T_{e/e'}$; in F_{incr}, it is any swap edge such that the maximum increment in the distance from the nodes in T_z to the root r is minimized in the new spanning tree $T_{e/e'}$.

The algorithm for computing the best swap edges with respect to F_{max} and F_{incr} have the same structure as the one for F_{sum}. What differs is: (i) the information propagated in the preprocessing phase, and (ii) the "enabling information" to be sent to the children during the algorithm.

For solving the F_{max} and the F_{incr} problems we require each node z to possess the following information: its distance $d(z, r)$ from the root, and the maximum distance $mD(T_q, z)$ to z from a node in T_q for each $q \in C(z)$. If this information is not available, it can be computed in the preprocessing phase with a basic convergecast described in the Appendix.

Let z be a node in T_x that needs to compute the cost of a candidate swap edge $e' = (z, z')$ for $e = (x, p(x))$. Let $T' = T_{e/e'}$.

Lemma 3 *The maximum distance $F_{max}(T')$ and the maximum distance increment $F_{incr}(T')$ in T' from a node in T_x to r are:*

$$F_{max}(T') = \max_{q \in C(z, T')} \{mD(T_q, z) + w(z, z') + d(z', r)\}$$

$$F_{incr}(T') = \max_{q \in C(z, T')} \{mD(T_q, z) + w(z, z') + d(z', r)\} - d(z, r)$$

To instantiate the generic algorithm of Section 3 for the F_{max} and the F_{incr} objective functions we have now to specify what is the *enabling information* that needs to be propagated so that all the nodes can make their local choice. As it will be shown, in both cases the enabling information that a node z has to send down to its child q is composed of the maximum distance $mD(T_x \setminus T_q, q)$ of the nodes in the subtree $T_x \setminus T_q$ to q. The algorithm for node z is then the same as the one for F_{sum}, where the computation of the cost of the local candidate swap edges and the enabling information change as follows:

Changes: MAX Algorithm

1. *If* $z = x$, the cost of each local candidate swap edge is computed as follows: for each $e' = (z, z')$,
$F_{max}(T_{e/e'}) = \max_{q \in C(x)}\{mD(T_q, x) + w(x, x') + d(x', r)\}$
$F_{incr}(T_{e/e'}) = \max_{q \in C(x)}\{mD(T_q, x) + w(x, x') + d(x', r)\} - d(x, r)$.
2. Else $\{z \neq x\}$ – *Receiving enabling info* m *for* $(x, p(x))$, the cost of each local candidate swap edge is computed as follows:
$F_{max}(T') = \max\{m, \max_{q \in C(z)}\{mD(T_q, z) + w(z, z') + d(z', r)\}\}$
$F_{incr}(T') = \max\{m, \max_{q \in C(z)}\{mD(T_q, z) + w(z, z') + d(z', r)\} - d(z, r))\}$.
3. The enabling information to be sent is $mD(T_x \setminus T_q, q)$.

Lemma 4 *Given* $e = (x, p(x))$ *, each node* $z \in T_x$ *correctly computes:* 1) *the local best swap edges for e,* 2) *the value* $mD(T_q, z)$ *for each* $q \in C(z)$*.*

6 Correctness and Complexity

Lemma 5 *Algorithms* BSE(F_{sum}),BSE(F_{max})*, and* BSE(F_{incr})*, find the best swap edge for* $e = (x, p(x))$ *according to the corresponding objective function.*

Theorem 1 *Independently executing Algorithms* BSE(F_{sum}),BSE(F_{max})*, and* BSE(F_{incr}) *for each edge, the problems* $\{r, \sum\}$*,* $\{r, \delta\}$*, and* $\{r, \max\}$ *are solved.*

Let us now examine the complexity of the proposed algorithm. Let n^* be the number of edges of the transitive closure of $T_r \setminus \{r\}$.

Theorem 2 *The message complexity of the Algorithms is at most* $3n^*$*.*

Since each message contains only a constant number of units of information (i.e., node, edge, label, weight, distance), the overall information complexity is of the same order of magnitude, i.e., $O(n^*)$.

7 An O(n) Messages Algorithm

7.1 Algorithmic Shell

The idea is that each node x simultaneously computes the "best" swap edges, not only for $(x, p(x))$, but also for each $(a, p(a))$, where a is an ancestor of x in T. At an high level, the algorithm consists simply of a *broadcast* phase started by the children of the root, followed by a *convergecast* phase started by the leaves.

Best F-Swap-Long (BSL)

[Broadcast.]

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

[Convergecast.]

1. Each leaf z first computes the best local swap for $(z, p(z))$; then, for each a in the received list, it computes the best candidate swap for $(a, p(a))$; finally, sends the list of those edges to its parent (if different from r).
2. An internal node y waits until it receives the list of best swap edges from each of its children. Based on the received information and on its local swap edges, it computes its best swap edge for $(y, p(y))$; it then computes for each ancestor a the best candidate for $(a, p(a))$; finally, it sends the list of those edges to its parent (if different from r).

To show how this generic algorithmic structure can be used to solve the three studied problems, we need to specify how the convergecast part is done. The differences in three solutions are: (i) the computation of the best swap edge in the convergecast phase, and (ii) the additional information, of constant size, to be communicated to the ancestors together with the swap edge.

In the following, we will denote by $SL(x)$ the *Swap List* associated to node x; it is defined as a list of records *(edge, value, attributes)*, where *edge* indicates a swap edge for $(x, p(x))$; *value* the value of the objective function computed in the tree where $(x, p(x))$ has been substituted with *edge*; and *attributes* a list of parameters to be specified for the particular problem being solved. Moreover, let $ASL(x)$ be the swap list associated to the ancestors of x; it is a list of records *(edge, value, attributes, node)* indicating for each node $a \in A(x)$ (stored in the field *node*) the best candidate for $(a, p(a))$ (stored in *edge*), and the value of the objective function (*value*); *attributes* is as in $SL(x)$.

Let us describe in details the operations executed by node x. First of all x computes the best swap edge for $(x, p(x))$ by considering the set $InS(x)$ of all local swap edges for $(x, p(x))$ and the set of swap edges transmitted to it from its children (Algorithm MyBSE). Then for each ancestor a it computes, among the swap edges in T_x, the best candidate for $(a, p(a))$ (Algorithm MyABSE). Note that the swap edges x computes for its ancestors can be worse than the final swap edges computed by its ancestors when *they* execute Algorithm MyBSE.

MyBSE

(* Algorithm for node x, where $e = (x, p(x))$ is the link to be swapped *)

1. Determine which of x's incident edges are swap edges for $(x, p(x))$; i.e., x constructs the set $InS(x)$.
2. For each swap edge $e_i = (x, y_i) \in InS(x)$, compute the value of the objective function via e_i, and the value of the other attributes and insert them together with e_i in $SL(x)$.
3. If x is not a leaf, from each $ASL(x_j)$ received from $x_j \in C(x)$, extract $(e_j, value, attributes, x)$ (or NIL, if no such record exists), and insert $(e_j, value, attributes)$ in $SL(x)$ (or NIL).
4. Sort $SL(x)$ in non decreasing order of $value$. The minimal element of $SL(x)$ gives one of the best swap edges for x and the value which minimizes the objective function.

MyABSE

(* Algorithm for node x *) For each ancestor node $a \in A(x)$:

1. Select the swap edge $e_i \in SL(x)$ which is also a swap edge for $(a, p(a))$, if any, with the minimal value of $value$, and consider its record $(e_i, v_i, attributes, a)$.
2. For $x_j \in C(x)$,$1 \leq j \leq h$, let $(e_j, v_j, attributes, a)$ be the record from $ASL(x_j)$. Update the values of v_j and of the $attributes$ in relation to node x. Consider the set of the updated records $\{(e_j, v_j, attributes, a) \cup (e_i, v_i, attributes, a)\}$, $1 \leq j \leq h$, where $(e_i, v_i, attributes, a)$ is the record computed in Step 1. Select from this set the record $(\overline{e}, \overline{v}, attributes, a)$ with minimal *value*, if any, and insert it, in $ASL(x)$ (to be sent to x's parent); if no record can be selected, insert NIL in $ASL(x)$.

7.2 Identifying a Swap Edge

In order for a node to decide if one of its incident edge is a swap edge it is sufficient to check, during the convergecast phase, the information collected in the broadcast phase.

Property 3 *The fact that an edge $(u, v) \in E \setminus E(T)$ with $u \in T_u$ and $v \in T \setminus T_u$ is a* swap *edge for $(x, p(x))$, with $x \in A(u)$, can be checked at node u, and no communication is needed.*

Property 3 derives from the fact that, after the broadcast phase, u knows all its ancestors (refer to Figure 1.(b)). Observe that if an edge is not a swap edge for $e = (x, p(x))$, it is not feasible for none of $a \in A(x)$.

8 The F_{sum} Problem with O(n) Messages

Problem F_{sum} is solved with minor modifications of the Convergecast Phase of Algorithm BSL.

To compute, each node z need some additional information: the distance $d_{T'}(z,r)$ in $T_{e/e'}$ for each considered swap edge e' for $(z,p(z))$; the weight $W(T_z)$ of the subtree T_z; the number of nodes $n(T_z)$ in such a subtree. The records of the list $SL(z)$ will thus have the form: $(edge, F_{sum}(T_z),$ $\{d_{T'}(z,r), W(T_z), n(T_z)\})$; the same three items (plus the field *node* indicating the ancestor) are stored in the records of $ASL(z)$.

The parameters $n(T_z)$ and $W(T_z)$ are easily computed inductively from the values sent to z by its children z_j, and from the weight of the edge (z_j, z). Namely: $n(T_z) = \sum_{z_i \in C(z)} n(T_{z_i}) + 1$; and $W(T_z) = \sum_{z_i \in C(z)} W(T_{z_i}) + \sum_{z_i \in C(z)} n(T_{z_i}) w(z, z_i)$. If z is a leaf $n(T_z) = 1$ and $W(T_z) = 0$.

Let us now show how to compute the new values of $F_{sum}(T_z)$, and of $d_{T'}(z,r)$ (Step 2 of MyBSE and of MyABSE).

Lemma 6 *Let $(z,y) \in InS(z)$. Then*

(i) $F_{sum} = W(T_z) + n(T_z) \cdot (w(z,y) + d_{T'}(y,r))$.

(ii) For each record $(e_i \neq NIL, F_{sum}(T_{z_i}), \{d_{T'}(z_i,r), W(T_{z_i}), n(T_{z_i})\}, z)$ received from child z_i, $d_{T'}(z,r) = w(z,z_i) + d_{T'}(z_i,r)$, and
$F_{sum}(T_z) = F_{sum}(T_{z_i}) + d_{T'}(z,r) + \sum_{j=1, j\neq i}^{h} (W(T_{z_j}) + n(T_{z_j})(w(z,z_j) + w(z,z_i) + d_{T'}(z_i,r)))$.

Consider in the example of Figure 1.(a) the computation of node D. Assume that nodes F and E have already computed the lists $ASL(F) = \{(F,B), 15, 4, 3, 3, D), NIL\}$ and $ASL(E) = \{(E,C), 7, 3, 1, 2, D), (E,C), 7, 3, 1, 2, B)\}$ and sent them to their parent D. D first computes the new values of $n(T_D) = 6$ and $W(T_D) = 9$. D has only one feasible swap edge (D,C), for which $d_{T'}(D,r) = 4$ and $F_{sum} = W(T_D) + n(T_D)4 = 33$. Then, among the values sent by its children, D considers FB for which $d_{T'}(D,r) = 5$ $F_{sum}(T_D) = F_{sum}(T_F) + d_{T'}(D,r) + W(T_E) + n(T_E)(w(E,D) + w(D,F) + d_{T'}(F,r)) = 33$. For the swap edge (E,C), $d_{T'}(D,r) = 4$ and $F_{sum}(T_D) = 29$, hence the best swap for D is E,C, that will be also selected for B.

The messages used in the convergecast phase are now longer with respect to the messages used in the approach of Section 4, but still of constant size. We finally have:

Theorem 3 *Each node $z \neq r$:*

(i) correctly computes its best swap edge:

(ii) determines for each ancestor $a \neq r$ the best swap edge for a in T_z.

Theorem 4 *F_{sum} can be solved with the $O(n)$ message complexity and $O(n_r^*)$ data complexity.*

9 The F_{max} and F_{incr} Problems with O(n) Messages

We will show how F_{max} is solved by BSL. The value to be minimized is the maximal distance from the nodes in T_z to the root via a swap edge e_i. Similarly to F_{sum}, we need to compute inductively two values; namely, the distance from z to the root via e_i, $d_{T'}(z,r)$, and the maximal distance from the nodes in T_z to z, that is $mD(T_q, z)$, with $q \in C(z)$. The list $SL(z)$ is now composed of records of four elements; namely: $(edge, F_{max}(T_z), \{d_{T'}(z,r), mD(T_q,z)\})$; $ASL(z)$ contains the same information, plus the field *node*.

Let us now show how to compute the new values of the parameters along a new swap edge e_i (Step 2 of MyBSE) and how to compute the same values when a swap edge transmitted from a child is considered. The same operations are performed also in Step 2 of MyABSE. We have:

Lemma 7 *Let T_{z_k} be the subtree of T_z containing the node at the maximal distance from r. Moreover, let $mD_2(z) = max_{q\neq k}(mD(T_q,z)$ be the maximal distance of the nodes in T_{z_j} to z, with $z_j \in \{C(z) \setminus z_k\}$. For $(z,l) \in InS(z)$, we have*

(i) $F_{max}(T_z) = \max_{q \in C(z,T)}(mD(T_q, z) + w(z,l) + d_{T'}(l,r))$.

(ii) For each record $(e_s \neq NIL, F_{max}(T_{z_s}), \{d_{T'}(z_s, r), mD(z_s)\}, z)$ received from child z_s, $d_{T'}(z,r) = (w(z,z_s) + d_{T'}(z_s,r))$. Moreover, if $s = k$, then $F_{max}(T_z) = \max(F_{max}(T_{z_s}), mD_2s(z) + d_{T'}(z,r))$; otherwise, $F_{max}(T_z) = \max(F_{max}(T_{z_s}), mD(z) + d_{T'}(z,r))$.

Thus, it follows that:

Theorem 5 *Each node $x \neq r$:*

(i) correctly computes the best swap edge for $(x, p(x))$ according to F_{max};

(ii) determines for each ancestor $a \neq r$ the best swap edge for v in T_u.

F_{incr} can be solved with a simple extension of the solution of F_{max}.

Theorem 6 *Problems F_{max} and F_{incr} can be solved with $O(n)$ messages and an overall $O(n_r^*)$ information complexity.*

References

[1] D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms. *CRC Handbook of Algorithms and Theory, CRC Press, 1997.*

[2] A. Di Salvo and G. Proietti. Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. *Proc. of 10th Colloquium on Structural Information and Communication Complexity* (SIROCCO 2004) 2004.

[3] P. Flocchini, T. Mesa, L. Pagli, G. Prencipe, and N. Santoro. Efficient protocols for computing optimal swap edges. *In Proc. of 3rd IFIP International Conference on Theoretical Computer Science* (TCS 2004), 2004, to appear.

[4] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79:43-59, 1988.

[5] H. Ito, K. Iwama, Y. Okabe, and T. Yoshihiro. Polynomial-time computable backup tables for shortest-path routing. *Proc. of 10th Colloquium on Structural Information and Communication Complexity* (SIROCCO 2003), 163–177, 2003.

[6] H. Mohanty and G.P.Bhattacharjee. A distributed algorithm for edge-disjoint path problem *Proc. of 6th Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), 44-361, 1986.

[7] 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):1–23, 1997.

[8] E. Nardelli, G. Proietti, and P. Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algoritmica*, 35:56–74, 2003.

[9] P. Narvaez, K.Y. Siu, and H.Y. Teng. New dynamic algorithms for shortest path tree computation *IEEE Transactions on Networking*, 8:735–746, 2000.

[10] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach, 3rd Edition*. Morgan Kaufmann, 2003.

[11] G. Proietti. Dynamic maintenance versus swapping: An experimental study on shortest paths trees. *Proc. 3rd Workshop on Algorithm Engineering* (WAE 2000), 207–217 2000

[12] R. E.Tarjan. Application of path compression on balanced trees. *Journal of ACM*, 26:690–715, 1979.

Appendix

A Pre-processing Phase for the F_{sum}-problem.

For solving the F_{sum}-problem we require each node z to possess the following information:

- its distance $d(z,r)$ from the root,
- the sum of the distances $W(T_z)$ of all nodes in T_q to z for each of the children q of z,
- the number of nodes $n(T_q)$ in T_q for each of its children q.

If the information is not initially available, it can be easily acquired by the nodes in a pre-processing phase, composed by the following simple convergecast procedure in T, executed only once at the beginning of the algorithm.

PRE-PROCESSING(F_{sum})

1. The root r sends down a message to each child q containing a `request-for-sum` and a value $k = w(r,q)$.
2. the message is propagated down to the leaves (adding to k the weight of each traversed edge so that each node z knows its distance $d(z,r)$ to the root).
3. when a leaf l receives the message it starts a convergecast up to the root to propagate the requested information.
4. a leaf l with parent $p(l)$ sends up $sum(T_l,(l,p(l))) = w(l,p(l))$ and $n(T_l) = 1$
5. an internal node z receiving from each of its children q, the values $W(T_q)$ and $n(T_q)$, will compute:

$$\begin{aligned} sum(T_z,(z,p(z))) &= W(T_z) + \left(\sum_{q\in C(z)} n(T_q) + 1\right) \cdot w(z,p(z)) \\ n(T_z) &= \sum_{q\in C(z)} n(T_q) + 1 \end{aligned}$$

and will send up the information $[sum(T_z,(z,p(z))), n(T_z)]$.

The correctness of the pre-processing procedure is proven by the following Lemma :

Lemma 8 *Let z be a node in T.*

- *The total number of nodes in T_z is:* $n(T_z) = \sum_{q\in C(z)} n(T_q) + 1$.
- *The sum of the distances from all nodes in T_z to $p(z)$ is:*

$$sum(T_z,(z,p(z))) = W(T_z) + n(T_z) \cdot w(z,p(z))$$

Proof. Part 1. is obvious. Let us consider Part 2. By definition, $sum(T_z,(z,p(z))) = \sum_{u\in V(T_z)} d(u,p(z))$. Thus,

$$\begin{aligned} sum(T_z,(z,p(z))) &= \sum_{u\in V(T_z)} d(u,z) + \sum_{u\in V(T_z)} w(z,p(z)) \\ &= W(T_z) + n(T_z) \cdot w(z,p(z)). \end{aligned}$$

■

Once all the information are available to the nodes, each node will exchange its local information with the neighbours in G.

The number of messages exchanged during the preprocessing phase is then: $O(|E|)$.

B Preprocessing for the F_{max} and F_{incr} Problems

For solving the F_{max} and the F_{incr} problems we require each node z to possess the following information:

- its distance from the root $d(z,r)$,
- the maximum distance $max(T_q, z)$ to z from a node in T_q for each $q \in C(z)$

This will be accomplished with a basic convergecast like in the previous section. In this case, lines 4. and 5. of protocol Pre-processing change as follows:

In the Pre-processing

4. a leaf l with parent $p(l)$ sends up $max(T_l, p(l)) = w(l, p(l))$
5. an internal node z receiving from each of its children q, the values $max(T_q, z)$ will compute

$$max(T_z, p(z)) = \max\{max(T_q, z)\} + w(z, p(z))$$

and will send up the information $max(T_z, p(z))$.

C Proofs of Lemmas and Theorems

Proof. of Lemma 1
By definition we know that $F_{sum}(T') = \sum_{t\in V(T_x)} d_{e/e'}(t,r) = \sum_{t\in T_x}[d_{e/e'}(t,z') + d(z',r)] = \sum_{t\in T_x} d_{e/e'}(t,z') + \sum_{t\in T_x} d(z',r)$, which is equal to $sum(T'_z,(z,z')) + n(T_x)\cdot d(z',r)$.
Substituting $sum(T'_z,(z,z'))$ and observing that, using the same reasoning as the one in the proof of Lemma 8, $n(T'_z) = \sum_{q\in C(z,T')} n(T_q) + 1$, the Lemma follows. ■

Proof. of Lemma 2
First observe that, by Lemma 8, after the preprocessing phase, a node z has available: the labeling $\lambda(y)$ of each of its neighbours y; the distance $d(y,r)$ to r from each of its neighbours y; the sum of the distances $sum(T_q,(q,z))$ of all nodes in T_q to itself and the number of nodes $n(T_q)$ in T_q for each of its children q. The proof is by induction on the number of nodes in the path from z to of x.

Basis. $z = x$; i.e., the link to be swapped is $(z, p(z))$. By Lemma 1 we know that, for each swap edge (x,x'), $\sum_{t\in V(T_x)} d_{e/e'}(t,r) = sum(T_x,(x,x')) + n(T_x)\cdot d(x',r)$. Since x is the root of T_x, all the needed information is available at x after the preprocessing phase. Thus, x can locally compute all the swap edges and choose the minimum. Moreover x can compute, by using local information only, $sum(T_x \setminus T_q,(x,q))$ and $n(T_x \setminus T_q)$ for each $q \in C(x)$.

Induction step. Let it be true for a node and consider its child z in T. By Lemma 1 we know that, for each swap edge (z,z'), $\sum_{t\in V(T_x)} d_{e/e'}(t,r) = sum(T_x,(z,z')) + n(T_x)\cdot d(z',r)$. Moreover, $sum(T_x,(z,z'))sum(T'_z,(z,z')) = \sum_{q\in C(z,T')} sum(T'_q,(q,z)) + (\sum_{q\in C(z,T')} n(T_q)+1)\cdot w(z,z')$.

Notice that the children of z in T' consists of all the children of z in T plus the parent of z in T (i.e., $C(z,T') = C(z)\cup\{(z,p(z))\}$. The values of $sum(T'_q,(q,z))$, and $n(T'_q)$ for $q \in C(z)$ have been computed in the preprocessing phase and are locally available. Since, by induction hypothesis, $p(z)$ has computed the locally best swap edge and the values of $s(T_x \setminus T_z,(p(z),z))$ and $n(T_x \setminus T_z)$, and since it has sent to z these information, z can now correctly compute the cost of all its local swap edge and choose the minimum. Moreover, it can now compute $s(T_x \setminus T_q,(z,q))$ and $n(T_x \setminus T_q)$ for each of its children $q \in C(z)$.

■

Proof. of Lemma 4
The values $w(u, u')$ and $d(u', r))$ are locally available because they have been computed in the preprocessing phase. We know that $C(u, T') = C(u) \cup \{(u, p(u))\})$. If $q \in C(u)$, then $max(T_q, u)$ is locally available because it has also been computed in the preprocessing phase. On the other hand, if $q = p(u)$, $max(T_q, u)$ has to be computed during the algorithm. By definition, this is the *enabling information* sent to u by $p(u)$. ■

Proof. of Lemma 5
By Lemmas 2, and 4 respectively, every node correctly computes its local best swap edge for e. By the correctness of the minimum finding, the global best swap edge will be communicated to x. ■

Proof. of Theorem 2
The preprocessing phase is executed only once and its complexity is $O(|E|)$. During the swap algorithm for $(x, p(x))$ the number of messages exchanged is $2|V(T_x)|$, thus, in total we have: $\sum_x 2|V(T_x)| = 2n^*$. ■

Proof. of Lemma 6
Assume that the children of z have already terminated their computation and transmitted their lists to z. Case (i) follows by Lemma 1. The scenario of Case (ii) is better understood looking at Figure 2. If a swap edge e_i belonging to T_{z_i} is considered, all the nodes in T_{z_i} maintain their distance from the root, hence they contribute to $F_{sum}(T_z)$ only for $F_{sum}(T_{z_i})$. Node z contributes for $d_{T'}(z, r)$. All the other nodes in $T_{z_j}, 1 \leq j \leq h, j \neq i$, to get the root, follow a path through edges (z_j, z), (z, z_i) and finally through the swap edges e_i. ■

Proof. of Theorem 3
First observe that, as result of the broadcast, every node receives the label of its ancestors (except r) and it can determine which edges are swap edges for itself and its ancestors (Property 2 and 3). 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, one component contains only z, while the other contains all the other nodes. In other words, the only possible swap edges are incident on z. Thus, z can correctly compute its best swap edge by computing the value of the distance as stated in point (i) of Lemma 6, thus proving (i). It can also immediately determine the swap edges with respect to all of its ancestors and compute for them the value of the parameters as stated in point (ii) of Lemma 6, and select, for each ancestor, the best candidate.

Induction step. Let the theorem hold for all nodes z with $0 \leq h(z) \leq k - 1$; we will now show that it holds for z with $h(z) = k$. By inductive hypothesis, it receives from each child y the best candidate for each ancestor of $y \in C(z)$, including z itself. Hence, based on these lists and on the locally available set $InS(z)$, z can correctly determine its optimal swap edge, as well as its best feasible swap edge for each of its ancestors.

■

Proof. of Theorem 4
The theorem follows immediately from Properties 2 and 3, and from the fact that, by Lemma 6, the messages still have constant size. ■

Proof. of Lemma 7
Assume that the children of z have already terminated their computation and transmitted their lists to z. From these values z can compute the maximum distance of a node in T_z, and Case (i) follows immediately. For Case (ii), if the swap edge e_s does not belongs to T_{x_k}, the maximal distance is given by the maximal value among $F_{max}(T_{z_s})$ and $(mD(z) + d_{T'}(z, r))$. Otherwise, all the nodes in T_{x_k} maintain their distance from the root; for all the other nodes (in $T_j, 1 \leq j \leq h$, $j \neq k$), called *far* nodes, to get to the root the path goes through edges (z_j, z), (z, z_k), and finally through the swap edge z_s. Hence, in this case, to compute the distance of the far nodes we have to consider the node at the maximal distance not belonging to T_{x_k}, whose distance is $mD_2(z)$. ■

Proof. of Theorem 6
It follows immediately from Properties 2 and 3, and from Lemma 7. ■

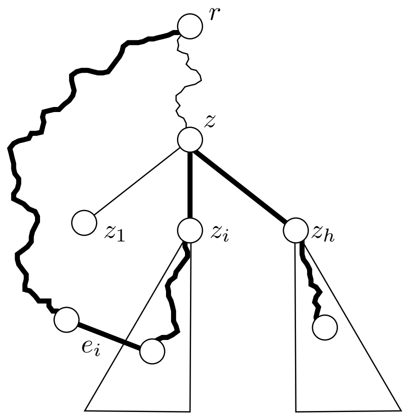

Figure 2: Case (ii) in Lemma 6: the computation of $F_{sum}(T_z)$ via the swap edge e_i. The thick line represents the path to the root via e_i.