Linear Time Distributed Swap Edge Algorithms*

A. K. Datta[1], L. L. Larmore[1], L. Pagli[2], and G. Prencipe[2]

[1] Dept. of Comp. Sc., Univ. of Nevada, {ajoy.datta,lawrence.larmore}@unlv.edu
[2] Dipartimento di Informatica, Università di Pisa, {pagli,prencipe}@di.unipi.it

Abstract. In this paper, we consider the *all best swap edges problem* in a distributed environment. We are given a 2-edge connected positively weighted network X, where all communication is routed through a rooted spanning tree T of X. If one tree edge $e = \{x, y\}$ fails, the communication network will be disconnected. However, since X is 2-edge connected, communication can be restored by replacing e by non-tree edge e', called a *swap edge* of e, whose ends lie in different components of $T - e$. Of all possible swap edges of e, we would like to choose the best, as defined by the application. The *all best swap edges problem* is to identify the best swap edge for every tree edge, so that in case of any edge failure, the best swap edge can be activated quickly. There are solutions to this problem for a number of cases in the literature. A major concern for all these solutions is to minimize the number of messages. However, especially in fault-transient environments, time is a crucial factor. In this paper we present a novel technique that addresses this problem from a time perspective; in fact, we present a distributed solution that works in linear time with respect to the height h of T for a number of different criteria, while retaining the optimal number of messages. To the best of our knowledge, all previous solutions solve the problem in $O(h^2)$ time in the cases we consider.

1 Introduction and Preliminaries

For a communication network, low cost and high reliability can be conflicting goals. For example, a *spanning tree* of a network could have minimum cost, but will not survive even a single link failure. We consider the problem of restoring connectivity when one link of a spanning tree fails.

One recent technique, particularly efficient in case of transient faults, consists in pre-computing a *replacement spanning tree* for each possible link or node failure, by computing the best replacement edge (or edges) which reconnects the tree. A number of studies have been done for this problem, both for the sequential [1–5] and distributed [6–9] models of computation, for different types of spanning trees and failures.

In this paper, we consider the *all best swap edges problem* in the distributed setting. We are given a positively weighted 2-edge connected network X of processes, where $w(x, y)$ denotes the weight of any edge $\{x, y\}$ of X, together with

* This work has been partially supported by MIUR of Italy under projects *ARS techno-media* and *AlgoDEEP* prot. 2008TFBWL4.

a spanning tree T of X, rooted at a process r. Suppose that all communication between processes is routed through T. If one tree edge $e = \{x, p(x)\}$ fails (where $p(x)$ denotes the parent of x in T) we say that x is the *point of failure.* Since X is 2-edge connected, communication can be restored by replacing e by an edge e' of X whose ends lie in different components of $T - e$. We call such an edge e' a *swap edge* of x (or a swap edge of e), and we define $\mathit{SwapEdges}(x)$ (or $\mathit{SwapEdges}(e)$) to be the set of all swap edges of x (refer to the example depicted in Figure 1.(b) and (c)). Of all possible swap edges of x, we would like to choose the best, as defined by the application. The *all best swap edges problem* is to identify the best swap edge for every tree edge, so that in case of any edge failure, the best replacement edge can be activated quickly.

Notation. Given T a spanning tree of X, we refer to an edge of T as a *tree edge*, and any other edge of X as a *cross edge* (see also Figure 1.(a)).

If $x \neq r$ is a process, we denote the set of children of x by $\mathit{Chldrn}(x)$, and the subtree of T rooted at x by T_x; the *level* of a process x is defined to be the *hop-distance* from x to r. We write $x \leq y$ or $y \geq x$ to indicate that x is an ancestor of y, *i.e.,* $y \in T_x$, and $x < y$ or $y > x$ if x is a proper ancestor of y.

If S is any subgraph of X, we let $\mathit{path}_S(x, y)$ denote the shortest (least weight) path through S from x to y, and let $W_S(x, y)$ denote the weighted length of $\mathit{path}_S(x, y)$. (We write simply $\mathit{path}(x, y)$ and $W(x, y)$ if S is understood.)

We will denote by T^* the *augmented tree*, whose nodes consist of all processes of T, together with a node for each directed cross edge of T, which we call an *augmentation node* of T^*. (See Figure 1.(d).) In particular, if $\{y, y'\}$ is a cross edge in T, we will denote by $[y, y']$ and $[y', y]$ its corresponding nodes in T^*; the parent of $[y, y']$ is y. For any process x, define T^*_x to be the subtree of the augmented tree rooted at x; in particular, T^*_x consists of T_x together with all the augmentation nodes $[y, y']$ such that $y \in T_x$.

Related Work and Our Contribution. In [8, 1], several different criteria for defining the "best" swap edge for a tree edge e have been considered. In each case, the best swap edge for e is that swap edge e' for which some penalty function F is minimized.

We consider three penalty functions in this paper. In each case, let $T' = T - e + e'$ be the spanning tree of X obtained by deleting e and adding e', where $e = \{x, p(x)\}$ is a tree edge, $y \in T_x$, and $e' = \{y, y'\}$ a swap edge for e.

1. $F_{\text{wght}}(x, y, y') = w(e')$, the weight of the swap edge. Note that if T is a minimum spanning tree of X and e' is that swap edge for e such that $w(e')$ is minimum, then $T' = T - e + e'$ is a minimum spanning tree of $X - e$.
2. $F_{\text{dist}}(x, y, y') = W_{T'}(r, x)$, the distance from the root to the point of failure in T'.
3. $F_{\max}(x, y, y') = \max\{W_{T'}(r, u) : u \in T_x\}$, the maximum distance, in T', from the root to any process in T_x.

If F is any of the above penalty functions, we define $F(x, y, y') = \infty$ for any $\{y, y'\}$ which is not a swap edge of x. The output of the problem is then $M_F(x) = \min\{F(x, y, y') : (y, y') \in T^*_x\}$.

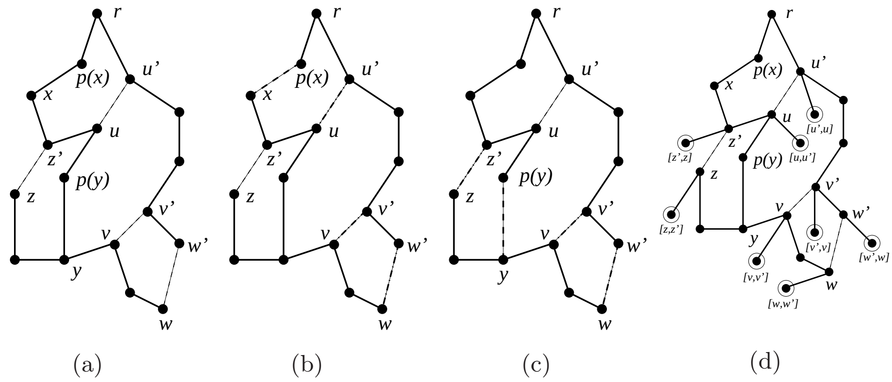


Fig. 1. (a) An example of a network X and its spanning tree T: Tree edges are bold, cross edges are dotted. (b) Failure at x. $\{u, u'\}$, $\{v, v'\}$, and $\{w, w'\}$ are the swap edges of x. (c) Failure at y. $\{v, v'\}$, $\{w, w'\}$, and $\{z, z'\}$ are the swap edges of y. (d) The augmented tree of T; the augmentation nodes are double circled.

In [6], Flocchini *et al.* give an algorithm for solving the F_{dist} version of all best swap edge problem. In [8], Flocchini *et al.* give a general algorithm for the all best swap edges problem, and then give specific versions of the technique for the $F_{\max}$ version. In [7], the F_{wght} version is solved both for the failure of a link and for the failure of a node and all its incident links.

All the above mentioned distributed solutions have the same general form, and have message complexity $O(n^*)$, with n^* the number of edges of the transitive closure of $T_r \setminus \{r\}$. The time complexity of each is $O(h^2)$, where h is the height of T. In particular, each of those solutions consists of two waves for each level ℓ of T, with $1 \leq \ell \leq h$: A broadcast wave followed by a convergecast wave (refer to the schema depicted in Figure 2.(a)). In particular, the general schema of previous solutions consists of two nested loops, where the outer loop is indexed by ℓ, and for each ℓ, the inner loop computes $M_F(x)$ for all x at level ℓ using two waves; a top-down wave that computes $F(x, y, y')$ for all $(y, y') \in T_x^*$, and a bottom-up wave that computes $M_F(x)$. Each wave takes $O(h)$ time in the worst case, hence the overall strategy leads to a final cost in time of $O(h^2)$. This is mainly due to the fact that the waves needs to be executed one after the other. In this paper we present a novel technique that finds a solution in linear time, for each of the penalty functions listed above. In particular, our strategy distributes the information and the computation among processes so that the waves can be pipelined, as shown in the general schema depicted in Figure 2.(b). This reduces the final time of the execution to $O(h)$, using the same number of messages as the previous solutions.

As a final remark, we note that in [1], Gfeller *et al.* study the problem of finding the optimal swap edges of a minimum spanning tree having minimum diameter. In particular, they provide a distributed algorithm that already works

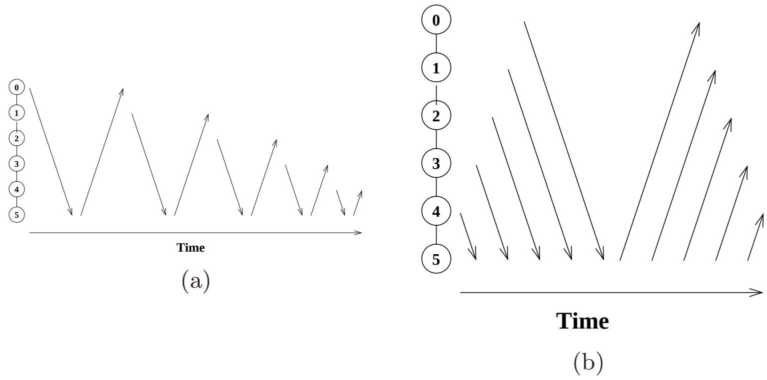


Fig. 2. Comparison between (a) the quadratic time paradigm, used in previous solutions to the best swap edge problem, and (b) the linear time paradigm introduced in this paper. Both paradigms have the same number and length of waves, but (b) uses pipelining to guarantee linear time complexity.

in linear time. The general technique that we present in this paper can be also adapted to this case.

The paper is organized as follows. The overall structure of our paradigm is given in Section 2. The various phases are described in Sections 3, 4, and 5. We conclude in Section 6. Due to space constraints, some of the proofs are given in the appendix.

2 The Linear Time Solution

In this section, we present the strategy that allows to devise $O(h)$-time distributed algorithms to solve the five versions of the all best swap edges problem introduced in the previous section. We call these algorithms $\text{LINEAR}_{\text{dist}}$, $\text{LINEAR}_{\text{wght}}$, $\text{LINEAR}_{\text{max}}$, respectively. Each can be considered to be a version of a general algorithm, which we call LINEAR, whose structure is given as Algorithm 1. LINEAR is structured in phases; the actual number of phases depends on the specific version of the problem. However, in all cases, the number of phases is at least three: a *preprocessing* phase, a *ranking* phase, and an *optimization* phase. In the last optimization phase, a piece of information, denoted by $up\_package(y, \ell)$, is computed in a convergecast wave. The content of this package is different for each of the versions of the problem, and will be detailed in Section 5.

Each of the phases of LINEAR uses at most $O(\delta_x)$ space for each process x, where δ_x is the degree of x. The space complexity of LINEAR is thus $O(\delta_x)$ for each x.

Our linear time algorithms make use of the concept of *critical level.* Informally, a critical level function is a function that can be computed top-down,

Algorithm 1 LINEAR

1: Preprocessing Phase
2: Ranking Phase
3: **If** $\text{LINEAR}_{\max}$ **Then** Additional Critical Level Phase(s)
4: Optimization Phase

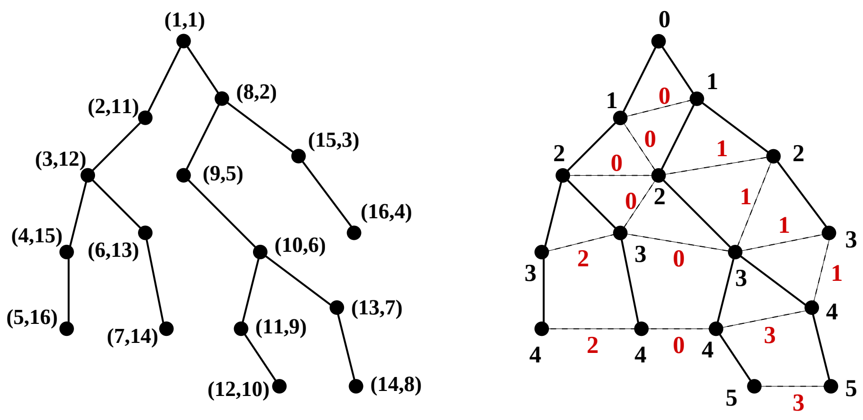
which enables another function – whose computation would otherwise require independent top-down followed by bottom-up waves for all processes – to be computed in a single bottom-up wave for each process, thus allowing the waves to be pipelined. In particular, for each of the versions of the best swap edge problem we consider, one or more critical levels are computed, depending on the specific penalty function. Due to space constraints, all the proofs will be omitted.

The Role of Critical Levels. A *critical level* function is a function Λ on the augmentation nodes of T^* such that $0 \le \Lambda(y, y') \le y.level$, and which aids in the computation of $F(x, y, y')$ for any $x \le y$. More specifically, the computation of $F(x, y, y')$ contains a branch which depends on the comparison between $x.level$ and $\Lambda(y, y')$. For example, the function *rank*, defined in Section 4, has the property that $F(x, y, y') = \infty$ if and only if $rank(y, y') \ge x.level$, where F is any one of the penalty functions defined above.

3 Preprocessing Phase

In the preprocessing phase, which takes $O(h)$ time, each process x computes and retains a set of variables, some of which are the same as in [6, 8]. All the variables listed below are needed for $\text{LINEAR}_{\max}$, but only *level*, *index*, and *depth* are needed for $\text{LINEAR}_{\text{wght}}$ and $\text{LINEAR}_{\text{dist}}$.

1. $x.level$, the *level* of x, which is the hop-distance from r to x.
2. $x.index = (x.pre\_index, x.post\_index)$, the *index* of x, where $x.pre\_index$ is the index of x in the pre-order visit of T, and $x.post\_index$ is the index of x in the reverse postorder of T (see Figure 3(a)).
3. $x.depth = W(r, x)$, the *depth* of x.
4. $x.height = \max\{W(x, u) : u \in T_x\}$, the *height* of x.
5. $x.best\_child$, the *best child* of x, defined to be the process $y \in Chldrn(x)$ such that $w(x, y) + y.height > w(x, z) + z.height$ for any other child z of x. Note that, since we use a strict inequality in this definition, a process can have at most one best child. If $Chldrn(x) = \emptyset$, or if there is more than one choice of y for which $w(x, y) + y.height$ is maximum, $best\_child(x)$ is undefined.
6. $x.eta$, for $x \ne r$, the largest weight of any path in $T_{p(x)} - T_x$ from $p(x)$; that is, $x.eta = \max\{w(p(x), y) + y.height : y \ne x \text{ and } y \in Chldrn(p(x))\}$. If x is the only child of its parent, then $x.eta$ defaults to 0.
7. $x.secondary\_height$, the length of the longest path which does not contain $x.best\_child$ from x to any leaf of T_x. In the case that $x.best\_child$ is undefined, let $x.secondary\_height = x.height$.



(a) (b)

Fig. 3. (a) Processes are labeled with their indices. A process x is an ancestor of y if and only if $x.index \le y.index$. (b) Levels of processes and ranks of cross edges.

Note that all of the above variables can be computed with a constant number of broadcast and convergecast waves, in $O(h)$ total time.

4 Ranking Phase

The ranking phase is the same for all versions of the best swap edge problem. In this phase, we compute the *rank* of every cross edge $\{y, y'\}$, defined to be the level of the nearest common ancestor of y and y' in T. This value is stored by both y and y'. Ranks are used to distinguish swap edges of x from other cross edges in T_x^*.

Remark 1.
(a) A process x is an ancestor of y if and only if $x.index \le y.index$.
(b) If $[y, y'] \in T_x^*$, then $\{y, y'\} \in SwapEdges(x)$ if and only if $x.index \not\le y'.index$.

From the previous remark, it follows that:

Remark 2. Let $x \ne r$ be a process and $e' = \{z, z'\}$ a cross edge, where $z \in T_x$. Then, e' is a swap edge for x if and only if $rank(z, z') < x.level$.

The ranking phase is given as Algorithm 2. In particular, there is a main loop that cycles over the levels of the tree in increasing order. The phase consists of a top-down *wave* for each $0 \le \ell \le h$, denoted by Wave ℓ. For each ℓ, the inner loop computes, for each process y whose level is greater than or equal to ℓ, the value $ancestor\_index(y, \ell)$, which is $x.index$ where x is the ancestor of y at level

Algorithm 2 Ranking Phase: Rank of every Cross Edge of T is Computed

For $0 \le \ell \le h$ in increasing order **Do** %*Wave* ℓ%
  For all y such that $y.level \ge \ell$ in top-down order **Do**
    If $y.level = \ell$ **Then** $ancestor\_index(y, \ell) \leftarrow y.index$
    Else $ancestor\_index(y, \ell) \leftarrow ancestor\_index(p(y), \ell)$
    For all cross edges $\{y, y'\}$ **Do**
      If $y'.index \not\geq ancestor\_index(y, \ell)$ **Then** $rank(y, y') \leftarrow \ell$

ℓ. Then, for each $[y, y'] \in T_x^*$, the value ℓ is assigned to $rank(y, y')$ if $y' \notin T_x$, *i.e.,* $y'.index \not\geq ancestor\_index(y.\ell)$ (refer to Remark 1).

The inner loop is executed as a top-down wave; hence the waves can be pipelined, so that the total time of the ranking phase is $O(h)$.

Lemma 1. *If* $rank(y, y') = \ell$*, then, for all* $\ell' \le \ell$*, the computed value of* $rank(y, y')$ *will be set to* ℓ' *during Wave* ℓ' *of Algorithm 2 and thus the final computed value of* $rank(y, y')$ *will be* ℓ*.*

5 Optimization Phase

The optimization phase is implemented as a bottom-up wave for each level ℓ. (Refer to Algorithm 1.) In this phase, all best swap edges are computed. In particular, the phase consists of an outer loop, indexed by decreasing values of $1 \le \ell \le h$, where each iteration consists of an inner loop which computes $M_F(x)$ for all x at level ℓ. For each x such that $x.level = \ell$, the inner loop consists of a convergecast wave, which computes a set of variables we call $up\_package(y, \ell)$ for each $y \in T_x$; each process y is able to compute $up\_package(y, \ell)$ by using the information computed and stored at y during the earlier phases, as well as the contents of $up\_package(z, \ell)$ received from all $z \in Chldrn(y)$. The final value of $M_F(x)$ is then computed using $up\_package(x, \ell)$. To save space, each up-package is deleted as soon as it is no longer needed. The convergecast waves can be pipelined, and thus the entire optimization phase can be executed in $O(h)$ time.

The specific content of $up\_package(y, \ell)$ depends on the specific version of LINEAR that is solved.

5.1 LINEAR$_{\mathbf{wght}}$ and LINEAR$_{\mathbf{dist}}$

For each $\ell \ge 1$ and each $y \in T$ at level $\ge \ell$, let x be the unique ancestor of y at level ℓ, and let $e = \{x, p(x)\}$. We define $Swap\_N(y, \ell)$ to be the set of all neighbors y' of y such that $\{y, y'\}$ is a swap edge for e. In order to compute this set, the test established by Remark 2 is used.

For both LINEAR$_{\text{wght}}$ and LINEAR$_{\text{dist}}$, $up\_package(y, \ell)$ consists of just the value $sbtree\_min(y, \ell)$, defined as follows. If x is the unique ancestor of y at level ℓ, then

1. In $\text{LINEAR}_{\text{wght}}$:

$$sbtree\_min(y, \ell) = \min\left\{w(z, z') : (z, z') \in T_y^* \cap SwapEdges(x)\right\}$$

2. In $\text{LINEAR}_{\text{dist}}$:

$$sbtree\_min(y, \ell) = \min\left\{W(x, z) + w(z, z') + z'.depth : (z, z') \in T_y^* \cap SwapEdges(x)\right\}$$

At the end of the iteration for ℓ, the value of $M_F(x)$ is set to $sbtree\_min(x, \ell)$ for all x at level ℓ.

The pseudo-code of the optimization phase, for the functions F_{wght} and F_{dist} is given as Algorithm 3 and 4, respectively. In both cases, the waves of the optimization phase are pipelined, permitting the total time complexity of the phase to be $O(h)$.

Concerning the number of messages of both $\text{LINEAR}_{\text{wght}}$ and $\text{LINEAR}_{\text{dist}}$, note that the information sent along the tree either in the ranking phase or in the optimization phase, is composed of messages of constant size. In the ranking phase, the information consists of node indices, and in the optimization phase of "subtree minimum" values. From Figure 2, where the flow of information is shown both for the old (quadratic time) approach and the new (linear time) paradigm, it can be easily observed that the total edge distance traveled by the messages along the tree is exactly the same. Therefore the communication complexity, corresponding to the transitive closure of the tree edges, is $O(n^*)$ (limited by $O(n^2)$) in both cases.

Algorithm 3 Algorithm $\text{LINEAR}_{\text{wght}}$

```
Preprocessing phase
Ranking phase
For all 1 ≤ ℓ ≤ h Do %Optimization Phase%
    For all y such that y.level ≥ ℓ in bottom-up order Do
        Swap_N(y, ℓ) ← {y' : {y', y} is a cross edge and rank(y, y') < ℓ}
        sbtree_min(y, ℓ) ← min { w(y, y') : y' ∈ Swap_N(y, ℓ)
                                { min {sbtree_min(z, ℓ) : z ∈ Chldrn(y)}
    For all x such that x.level = ℓ Do
        M_F(x) = sbtree_min(x, ℓ)
```

5.2 $\text{LINEAR}_{\textbf{max}}$

If $S \subseteq X$ is connected and $x \in S$, define $radius(S, x) = \max\{W_S(x, s) : s \in S\}$, the *radius of* S *based at* s. Note that, we can write $F_{\max}(x, y, y') = \max\{W_{T'}(r, u) : u \in T_x\} = radius(T_x, y) + w(y, y') + y'.depth$ if $\{y, y'\} \in SwapEdges(x)$. Thus, in the case of $\text{LINEAR}_{\max}$ we face with the problem of computing $radius(T_x, y)$: This computation is handled by an additional phase before the actual optimization phase. In this phase, we compute a variable called $critical\_level(y)$, for all $y \in T_x$.

Algorithm 4 Algorithm LINEAR$_{\text{dist}}$

Preprocessing phase
Ranking phase
For all $1 \leq \ell \leq h$ **Do** %*Optimization Phase*%
  For all y such that $y.level \geq \ell$ in bottom-up order **Do**
    $Swap\_N(y, \ell) \leftarrow \{y' : \{y', y\} \text{ is a cross edge and } rank(y, y') < \ell\}$
    $sbtree\_min(y, \ell) \leftarrow \min \begin{cases} w(y, y') + depth(y') : y' \in Swap\_N(y, \ell) \\ \min \{w(y, z) + sbtree\_min(z, \ell) : z \in Chldrn(y)\} \end{cases}$
  For all x such that $x.level = \ell$ **Do**
    $M_F(x) = sbtree\_min(x, \ell)$

Additional Critical Level Phase. For $y \in T_x$, define $\mu(y, x)$ to be the weight of the longest path in T_x from y to any node of $T_x - T_y$. We let $\mu(x, x) = 0$ by default. It follows from these definitions that

$$radius(T_x, y) = \max \begin{cases} y.height \\ \mu(y, x) \end{cases} \tag{1}$$

Since we want LINEAR to use only constant space per process, y can hold only $O(\delta_y)$ values; hence, it could not be possible for y to store all the values $\{\mu(y, x) : x \leq y\}$. We tackle this problem by executing in LINEAR$_{\max}$ an extra phase before the optimization phase (called *critical level phase* in Algorithm 1). In particular, as the convergecast wave moves up the tree, we compute the *critical level* of y, that determines not the actual value of $radius(T_x, y)$, but rather which of the two choices given in Equation (1) is larger, together with enough additional information to calculate the actual value of $M_F(x)$ when the wave reaches x.

We now explain critical levels in greater detail. Let

$$critical\_level(y) = \min \{x.level : y \in T_x \text{ and } radius(T_x, y) = y.height\}$$

Note that

$$critical\_level(y) = \min \{x.level : y \in T_x \text{ and } \mu(y, x) \leq y.height\}$$

Lemma 2. *For any processes $x' \leq x \leq y$, $\mu(y, x') \geq \mu(y, x)$.*

Corollary 1. *If $y \in T_x$, then $radius(T_x, y) = y.height$ if and only if $x.level \geq critical\_level(y)$.*

Critical levels are calculated by Algorithm 5. Recall, from Section 3, that $y.eta$, for $y \neq r$, is the largest weight of any path in $T_{p(y)} - T_y$ from $p(y)$; this value is computed during the preprocessing phase. Note that, once again, the waves of the inner loop of Algorithm 5 can be pipelined, so that the total time required for this phase is again $O(h)$.

Optimization Phase. Before introducing the optimization phase, we need to introduce the notion of *Spine*, which is strictly related to the notion of critical level. (Refer also to Figure 4.)

Algorithm 5 Critical Level Phase

For $0 \le \ell \le h$ in decreasing order **Do** %*Wave* ℓ%
  For all x such that $x.level = \ell$ concurrently **Do**
    $\mu(x,x) \leftarrow 0$
    For all $y \in T_x - x$ in top down order **Do**
      $\mu(y,x) \leftarrow \max \begin{cases} \mu(p(y),x) + w(y,p(y)) \\ y.eta \end{cases}$
      If $\mu(y,x) \le y.height$ **Then** $critical\_level(y) \leftarrow \ell$

Definition 1 (Spine). *Given any process x, we define the* Spine *of x:*

$$Spine(x) = \{y \in T_x : radius(T_x, y) = y.height\}.$$

We extend this definition to a specific level ℓ as follows: $Spine(\ell) = \bigcup\{Spine(x) : x.level = \ell\}$.

We will denote by $Others(x)$ the nodes in T_x that are not in $Spine(x)$ (*i.e.*, $Others(x) = T_x - Spine(x)$), and by $Others(\ell) = \bigcup\{Others(x) : x.level = \ell\}$. Furthermore, we define the *base process* of x, denoted by $base(x)$, as the process in $Spine(x)$ of greatest level; again, given a specific level ℓ, we let $Base(\ell) = \{base(x) : x.level = \ell\}$. We define the *tail process* of x as $tail(x) = best\_child(base(x))$ (note that $tail(x)$ might be not defined), and $Tail(\ell) = \{tail(x) : x.level = \ell\}$. Finally, we let $Fan(x) = T_{tail(x)}$ and $Fan(\ell) = \bigcup\{Fan(x) : x.level = \ell\}$; if $tail(x)$ is undefined, we let $Fan(x) = \emptyset$. We now give few properties of $Spine(x)$.

Lemma 3. *For any process x*
(a) *$x \in Spine(x)$.*
(b) *If $y \in Spine(x)$ and $y \neq x$, then $p(y) \in Spine(x)$ and $y = best\_child(p(y))$.*
(c) *$Spine(x)$ is a chain.*

For any $s \in S$, where S is connected, let $longest\_path(S, s)$ denote the simple path of weight $radius(S, s)$ in S starting at s. In the next lemma, we give a characterization of $longest\_path(T_x, y)$.

Lemma 4. *Let $y \in T_x$, and let u be the process of minimum level on $longest\_path(T_x, y)$. Then, the following properties hold:*
(a) *$u \in Spine(x)$.*
(b) *If $y \in Fan(x)$, then $longest\_path(T_x, y) = path(y, u) + secondary\_down\_path(u)$, where "+" denotes concatenation of paths.*
(c) *If $y \notin Fan(x)$, then $longest\_path(T_x, y) = path(y, u) + longest\_path(T_u, u)$,*

Let F_ℓ be the forest given by the union of all T_x, where $x.level = \ell$. Thus, $radius(F_\ell, y) = radius(T_x, y)$ if $x.level = \ell$ and $y \in T_x$. The critical level of a process y enables y to determine whether it lies in $Others(\ell)$ for any given ℓ, as shown by the following lemma.

Lemma 5. *$y \in Spine(\ell)$ if and only if $critical\_level(y) \le \ell \le y.level$.*

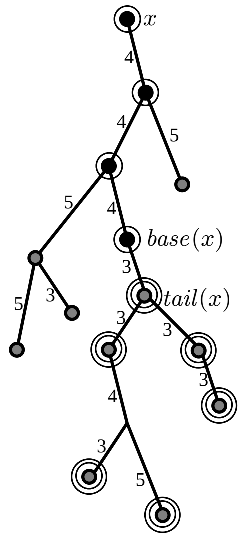


Fig. 4. Definition 1: Black (and single circled) nodes are in $Spine(x)$, while the light gray nodes are in $Others(x)$; $base(x)$, $tail(x)$ are also shown; nodes in $Fan(x)$ are double circled.

Corollary 2. *Given any y such that $\ell \leq y.level$, the following properties hold:*
(a) *$y \in Others(\ell)$ if and only if $critical\_level(y) > \ell$.*
(b) *$y \in Spine(\ell)$ if and only if $critical\_level(y) \leq \ell$.*
(c) *$y \in Base(\ell)$ if and only if $y \in Spine(\ell)$, and either $best\_child(y) \in Others(\ell)$, or $best\_child(y)$ is undefined.*
(d) *$y \in Tail(\ell)$ if and only if $p(y) \in Base(\ell)$ and $y = best\_child(p(y))$.*

Corollary 2 is used during the optimization phase of $\mathrm{LINEAR_{max}}$ to determine the content of $up\_package(y, \ell)$ (See Algorithms 1 and 6). In particular, the optimization phase proceeds bottom-up in the tree, with two nested loops. Let

$$local\_cost(y, \ell) = \min\{w(y, y') + depth(y') : y' \in Swap\_N(y, \ell)\},$$

where $Swap\_N(y, \ell)$ is as defined in Section 5.1.

If $y \in Others(\ell)$, then $radius(F_\ell, y)$ is not computed going down in the tree; hence, the only information that needs to be propagated (that is, the content of $up\_package(y, \ell)$) is

$$min\_up\_cost(y, \ell) = \min\{local\_cost(z, \ell) + W(y, z) : z \in T_y\},$$

i.e., the minimum value of $W(path(y, z)) + w(z, z') + depth(z')$ over all $z \in T_y$ such that $\{z, z'\} \in SwapEdges(x)$.

Algorithm 6 Algorithm $\text{LINEAR}_{\max}$

Preprocessing phase
Ranking phase
Critical Level Phase (Algorithm 5)
For all $1 \leq \ell \leq h$ **Do** %*Optimization Phase*%
  For all y such that $y.level \geq \ell$ in bottom-up order **Do**
    $Swap\_N(y,\ell) \leftarrow \{y' : \{y', y\}$ is a cross edge and $rank(y,y') < \ell\}$
    $local\_cost(y,\ell) = \min\{w(y,y') + depth(y') : y' \in Swap\_N(y,\ell)\}$
    If $y \in Others(\ell)$ **Then**
      $min\_up\_cost(y,\ell) \leftarrow \min \begin{cases} local\_cost(y,\ell) \\ \min\{min\_up\_cost(z,\ell) + w(y,z) : z \in Chldrn(y)\} \end{cases}$
    Else %$y \in Spine(\ell)$%
      $min\_normal\_cost(y,\ell) \leftarrow \min \begin{cases} local\_cost(y,\ell) \\ \min\{min\_up\_cost(z,\ell) + w(y,z) : z \in Normal\_Chldrn(y)\} \end{cases}$
      If $best\_child(y)$ is defined **Then**
        $z \leftarrow best\_child(y)$
        If $z \in Spine(\ell)$ **Then**
          $min\_fan\_cost(y,\ell) \leftarrow min\_fan\_cost(z,\ell) + w(z,y)$
        Else
          $min\_fan\_cost(y,\ell) \leftarrow min\_up\_cost(z,\ell) + w(z,y)$
        $sbtree\_min(y,\ell) \leftarrow \min \begin{cases} min\_normal\_cost(y,\ell) + y.height \\ min\_fan\_cost(y,\ell) + secondary\_height(y) \\ sbtree\_min(z,\ell) \end{cases}$
      Else %$y = base(x)$, *and* $tail(x)$ *undefined*%
        $min\_fan\_cost(y,\ell) \leftarrow \infty$
        $sbtree\_min(y,\ell) \leftarrow min\_normal\_cost(y,\ell) + y.height$
  For all x such that $x.level = \ell$ **Do**
    $M_F(x) = sbtree\_min(x,\ell)$

If $y \in Spine(\ell)$, first the value of $min\_normal\_cost(y,\ell)$ is computed:

$$min\_normal\_cost(y,\ell) = \min\left\{local\_cost(z,\ell) + W(y,z) : z \in T_y \text{ and } z \notin T_{best\_child(y)}\right\}.$$

Then, the algorithm branches according to whether $best\_child(y)$ is defined or not. If $z = best\_child(y)$ is defined, then $T_y \cap Fan(\ell) \neq \emptyset$; in this case, $min\_fan\_cost(y,\ell)$ is computed:

$$min\_fan\_cost(y,\ell) = \min\{local\_cost(z,\ell) + W(y,z) : z \in T_y \cap Fan(\ell)\},$$

i.e., the minimum value of $W(path(y,z)) + w(z,z') + depth(z')$ over all $\{z,z'\} \in SwapEdges(x)$ such that $z \in T_y \cap Fan(\ell)$ (note that the algorithm computes $min\_fan\_cost(y,\ell)$ differently, according to whether $z \in Spine(\ell)$ or not). Finally, the actual cost of the swap edge is computed: $sbtree\_min(y,\ell)$, which is the minimum value of $radius(T_y,z) + w(z,z') + depth(z')$ over all $\{z,z'\} \in SwapEdges(x)$ such that $z \in T_y$ (this is the value that is propagated in $up\_package(y,\ell)$). In particular, by Lemma 4, $sbtree\_min(y,\ell)$ is the minimum between the value of $sbtree\_min(z,\ell)$ obtained from z, $min\_normal\_cost(y,\ell) + y.height$, and $min\_fan\_cost(y,\ell) + secondary\_height(y)$.

If $z = best\_child(y)$ is not defined, then $T_y \cap Fan(\ell) = \emptyset$. In this case, $min\_fan\_cost(y, \ell)$ is set to ∞, and $sbtree\_min(y, \ell)$ is set to $min\_normal\_cost(y, \ell)$ $+y.height$.

When the ℓ^{th} wave terminates, it is possible to compute the best swap edge for x: $M_F(x) = sbtree\_min(x, \ell)$ for all x such that $x.level = \ell$. Again, as in the previous cases, the waves are executed in pipeline, and thus the overall time complexity is $O(h)$. Also, in this case, it is not difficult to see that the number of messages used by Algorithm 6 is the same as for the quadratic time versions, *i.e.,* $O(n^*)$.

Theorem 1. *The overall time complexity for* LINEAR *is* $O(h)$.

6 Summary

In this paper, we present a new technique that yields solutions to several versions of the all best swap edges problem in linear time; in particular, we give linear time solutions for $\text{LINEAR}_{\text{wght}}$, $\text{LINEAR}_{\text{dist}}$, We also note that the technique presented in this paper can be adapted to find swap edges that minimize the diameter of the spanning tree in linear time, using an approach similar to the one used by $\text{LINEAR}_{\text{max}}$.

References

1. Gfeller, B., Santoro, N., Widmayer, P.: A distributed algorithm for finding all best swap edges of a minimum diameter spanning tree. IEEE Trans. on Dependable and Secure Comp **8**(1) (2011) 1–12
2. Nardelli, E., Proietti, G., Widmayer, P.: Swapping a failing edge of a single source shortest paths tree is good and fast. Algorithmica **35**(1) (2003) 56–74
3. Nardelli, E., Proietti, G., Widmayer, P.: Nearly linear time minimum spanning tree maintenance for transient node failures. Algorithmica **40**(1) (2004) 119–132
4. Salvo, A.D., Proietti, G.: Swapping a failing edge of a shortest paths tree by minimizing the stretch factor. Theoretical Computer Science **383**(1) (2007) 23–33
5. Shantanu Das, Beat Gfeller, N.S., Widmayer, P.: Computing best swaps in optimal tree spanners. In: ISAAC. (2008) 716–727
6. Flocchini, P., Enriques, A.M., Pagli, L., Prencipe, G., Santoro, N.: Point-of-failure shortest-path rerouting: Computing the optimal swap edges distributively. IEICE Transactions **89-D**(2) (2006) 700–708
7. Flocchini, P., Pagli, A.M.E.L., Prencipe, G., Santoro, N.: Distributed minumum spanning tree maintenance for transient node failures. IEEE Trans. on Comp. **61**(3) (2012) 408–414
8. Flocchini, P., Pagli, L., Prencipe, G., Santoro, N., Widmayer, P.: Computing all the best swap edges distributively. J. Parallel Distrib. Comput. **68**(7) (2008) 976–983
9. Pagli, L., Prencipe, G.: Distributed swap edges computation for minimum routing cost spanning trees. In: OPODIS. (2009) 365–371