Brief Annoucement: Distributed Swap Edges Computation for Minimum Routing Cost Spanning Trees

Linda Pagli and Giuseppe Prencipe

Dipartimento di Informatica, Università di Pisa
{pagli,prencipe}@di.unipi.it

1 Introduction

Given a weighted graph $G(V_G, E_G)$ representing a communication network, with n nodes and m edges where the weights are positive integers, its *Spanning Tree* is typically used to route messages. In [1] the *routing cost* of a spanning tree is defined as the sum of the distances over all pairs of vertices of this tree. Hence, the most suitable spanning tree for the routing problem is the one minimizing the routing cost: the *Minimum Routing Cost Spanning Tree* (MRCST).

Since trees have the minimal possible number of edges, a single edge failure is enough to interrupt the communication. For this reason, it is very important that it is fault-tolerant at least for the failure of one edge. Two approaches can be followed after the failure of an edge e: one can compute from scratch a new *optimal* spanning tree for $G - e$; or re-connect the two disconnected component of the spanning tree with *the best possible edge*. For temporary network failures, this second approach is clearly preferable to the first one: in fact, it is more efficient to use a swap edge for the duration of the failure, and to quickly revert to the original tree after the failure has been fixed; in this way, the original routing tables do not need major changes. In addition, we also note that finding the spanning tree with minimum routing cost is known to be *NP-hard* [3].

For these reasons, the "swap-edge" approach has been followed to solve similar problems in sequential and distributed fashion; see for instance [2,4,6]. In our case, the *best possible edge* to select is the one that re-connects the two components disconnected by the fault of edge e, and leading to the new spanning tree for $G - e$ having the minimum routing cost. This edge can be pre-computed for any possible edge fault and stored to recover the routing process in case of fault.

The same problem has been solved for the sequential model in [6] in $O(nm)$ time complexity. To compute the cost of the spanning tree, the authors make use of the concept of the "routing load" of one edge, expressing the load of the different routes passing through this edge. The total cost is then computed by multiplying the routing load by the edge weight. We use here a different technique that computes the cost of a tree from the cost of its subtrees. Different is also the information we keep at each tree node in order to rebuild, in a distributed fashion, the tree after an edge fault. Our approach appears more suitable for the distributed solution, because allows us to treat contemporarely any possible edge fault.

T. Abdelzaher, M. Raynal, and N. Santoro (Eds.): OPODIS 2009, LNCS 5923, pp. 365–371, 2009.

In this paper, we will present the first, to our knowledge, distributed algorithm to compute the best swap edge for a minimum routing cost spanning tree, that works with message complexity $O(m \cdot n)$. Our algorithms works for any spanning tree of a 2-connected graph; hence, for its MRCST.

2 Definitions, Basic Properties, and Algorithm

Notation. Let $G = (V_G, E_G)$ be a simple undirected graph, with $n = |V|$ vertices and $m = |E|$ edges. A *subgraph* $G' = (V', E')$ of G is any graph where $V' \subseteq V_G$ and $E' \subseteq E_G$. If $V' \equiv V_G$, G' is a *spanning* subgraph. A *path* $P = (V_p, E_p)$ is a subgraph of G, such that $V_p = \{v_1, \ldots, v_s\} | v_i \neq v_j$, for $i \neq j$, and $(v_i, v_{i+1}) \in E_p$, for $1 \leq i \leq s-1$. A spanning tree T of a given graph G is a *spanning* subgraph of G that is also a tree. Given a non-rooted tree T, we define its degree-1 nodes as the *leaves* of T; the non-leaves nodes in T will be referred to as the *internal* nodes of T.

A non negative real value called *weight* (or *length*) is associated to each edge e in G. Given a path P, the length of the path is the sum of the lengths of its edges. The *distance* $d_{G'}(x, y)$ (or simply $d(x, y)$) between two vertices x and y in a connected subgraph G' of G, is the length of the shortest path from x to y in G'.

Given a tree T, let u be a node in T; we will denote by T_u the version of T rooted in u. Furthermore, let $e = (u, v)$ be an edge in T; we will denote by $T_{u \setminus v}$ and $T_{v \setminus u}$ the two trees, rooted in u and v respectively, obtained by removing the edge (u, v) in T.

We will call the *weight* of T the sum of the distances of all paths between all pairs of nodes in a tree T, and denote it by $\mathcal{W}(T)$. That is,

$$\mathcal{W}(T) = \Sigma_{(i,j) \in V_T \times V_T} d_T(i, j)^1. \tag{1}$$

Furthermore, we define by $\mathcal{D}(u, T)$ the sum of the distances between u and all other nodes in T, and by $n(T)$ the number of nodes in T. Finally, let u' be a node in $T_{u \setminus v}$. We will denote by $\mathcal{D}_v(u', T)$ the sum of the distances between u' and each of the nodes in $T_{v \setminus u}$.

The Problem and the Model. Let T be any spanning tree of a given 2-connected graph $G(V_G, E_G)$. The fault (i.e., removal) of any edge e of T disconnects the tree into two subtrees; since G is 2-connected, there will exist at least one edge $f \in E_G \setminus E_T$, called the *swap edge for* e, able to re-connect T. Let us denote by $T'_{e|f}$ the spanning tree of G obtained by removing e and inserting f in T. We have:

Definition 1. *The* best swap edge *for a failing edge $e \in T$ is any swap edge f for e such that $\mathcal{W}(T'_{e|f})$ is mimimum.*

[1] Note that the weight of an edge (i, j) is considered only once in the sum.

By extending to non-rooted trees the definition of swap edge in [2], we have:

Property 1 (Swap Edge Property). An edge $f = (u, v) \in E_G \setminus E_T$ is a swap for $e = (x, y) \in E_T$ if and only if only one of u and v are in $T_{x \setminus y}$.

This test is easily made by having the tree labeled with a pre-order and inverse pre-order labeling (details on this can be found in [2]).

The problem addressed in this paper consists in computing, in a distributed way, the *best swap edge* for any possible fault of one of the edges of T, and will be denoted as the BSRT problem. In particular, the algorithm we will present works also for the *Minimum Routing Cost Spanning Tree* of G; that is, the spanning tree T of G such that $\mathcal{W}(T)$ is minimum.

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 distinct 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 sequence of bits called messages[2]. 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.

Lemma 1. *Let T be a weighted tree, $e = (u, v)$ be an edge of T, and u' be a node in $T_{u \setminus v}$. Then, we have*

$$\mathcal{D}_v(u', T) = d(u', v) \cdot n(T_{v \setminus u}) + \mathcal{D}(v, T_{v \setminus u}), \tag{2}$$

$$\mathcal{D}(u', T) = \mathcal{D}(u', T_{u \setminus v}) + (d(u', u) + d(u, v)) \cdot n(T_{v \setminus u}) + \mathcal{D}(v, T_{v \setminus u}), \tag{3}$$

$$\mathcal{W}(T) = \mathcal{W}(T_{u \setminus v}) + \mathcal{W}(T_{v \setminus u}) + n(T_{v \setminus u}) \cdot \mathcal{D}(u, T_{u \setminus v}) + n(T_{u \setminus v}) \cdot \mathcal{D}_v(u, T). \tag{4}$$

Furthermore, it is easy to see that $\mathcal{W}(T) = \mathcal{W}(T_u)$, for all $u \in T$.

3 The Algorithm

Our distributed algorithm to solve the BSRT problem is articulated in 3 main phases. Let T be any spanning tree of a 2-connected weighted graph G.

Saturation phase. In this phase, using the well-known *saturation technique* [5], each node u in T will compute locally the following values: $\mathcal{W}(T)$, $\mathcal{D}(u, T)$, $n(T)$, and $\mathcal{W}(T_{u \setminus u_j})$, $\mathcal{D}(u, T_{u \setminus u_j})$, and $n(T_{u \setminus u_j})$ for each neighbor u_j of u in T. In this phase, also, a node r in T is elected as the root of the tree; we will denote by T_r the rooted version of T.

[2] Since our algorithm transmits sum of weights in a single message, the messages have length bounded by $O(W + \log n)$ bits, where W is the number of bits necessary to describe the maximum value of $\mathcal{W}(T)$, among all possible spanning trees T of G.

Swap phase. This phase starts the core of the algorithm: each node $u \in T_r$ computes the set of all candidate swap edges for the failure of edge $(u, parent(u))$, with $parent(u)$ the parent of u in T_r. We will denote this set for u by $\mathcal{S}(u)$.

Best Swap phase. Finally, each node u will locally determine which one, among the candidate swap edges in $\mathcal{S}(u)$, is the best one, as defined in Definition 1.

3.1 Saturation Phase

Step I: Collecting The computation starts from degree-1 nodes in T. Let u be a leaf in T, and u_1 its only neighbor: u sends to u_1 the following values: $\mathcal{D}(u, T_{u\setminus u_1}) = 0$, $\mathcal{W}(T_{u\setminus u_1}) = 0$, and $n(T_{u\setminus u_1}) = 1$.

Let u be an internal node of T, with neighbors $u_1, u_2, \ldots, u_i$: it waits until it receives the values $\mathcal{D}(u_j, T_{u_j\setminus u})$, $\mathcal{W}(T_{u_j\setminus u})$, and $n(T_{u_j\setminus u})$ from $i-1$ neighbors. Then, it computes $\mathcal{D}(u, T_{u\setminus u_i})$, $\mathcal{W}(T_{u\setminus u_i})$, and $n(T_{u\setminus u_i})$, as follows.

First, $n(T_{u\setminus u_i}) = \Sigma_{j=1}^{i-1} n(T_{u_j\setminus u}) + 1$. Then, by definition, $\mathcal{D}(u, T_{u\setminus u_i}) = \Sigma_{j=1}^{i-1} \mathcal{D}_{u_j}(u, T)$; therefore, by Equation (2), it follows that $\mathcal{D}(u, T_{u\setminus u_i}) = \Sigma_{j=1}^{i-1}(d(u, u_j) \cdot n(T_{u_j\setminus u}) + \mathcal{D}(u_j, T_{u_j\setminus u}))$. Finally, we have

$$\mathcal{W}(T_{u\setminus u_i}) = \Sigma_{j=1}^{i-1} \mathcal{W}(T_{u_j\setminus u}) + \mathcal{D}(u, T_{u\setminus u_i}) + \\ + \Sigma_{j=1}^{i-2} \Sigma_{k=j+1}^{i-1} (n(T_{u_k\setminus u}) \cdot \mathcal{D}_{u_j}(u, T) + n(T_{u_j\setminus u}) \cdot \mathcal{D}_{u_k}(u, T)).$$

We note that, node u has locally all the necessary information to compute $\mathcal{D}(u, T_{u\setminus u_i})$ and $\mathcal{W}(T_{u\setminus u_i})$. As its final action, node u sends to node u_i, referred to as the *recipient* of u, the values of $n(T_{u\setminus u_i})$, $\mathcal{D}(u, T_{u\setminus u_i})$, and $\mathcal{W}(T_{u\setminus u_i})$ just computed.

Step II: Exchanging After finite time, due to the saturation process, there will be two nodes, x and y, that will exchange on edge $s = (x, y)$ their $\mathcal{D}()$, $\mathcal{W}()$, and $n()$ values. In particular, at this time, both x and y have (locally) the following information: $\mathcal{D}(x, T_{x\setminus y})$, $\mathcal{W}(T_{x\setminus y})$, and $n(T_{x\setminus y})$; and $\mathcal{D}(y, T_{y\setminus x})$, $\mathcal{W}(T_{y\setminus x})$, and $n(T_{y\setminus x})$: s is called the *saturated edge*.

Lemma 2. *The total weight of T is*

$$\mathcal{W}(T) = \mathcal{W}(T_{x\setminus y}) + \mathcal{W}(T_{y\setminus x}) + n(T_{y\setminus x}) \cdot \mathcal{D}(x, T_{x\setminus y}) + n(T_{x\setminus y}) \cdot \mathcal{D}_y(x, T). \quad (5)$$

Furthermore, after the second step of the Saturation process, node x has locally all the information necessary to compute $\mathcal{W}(T)$, $n(T)$, and $\mathcal{D}(x, T)$.

Step III: Broadcasting In the third step of the Saturation phase, the values computed by the *saturated* nodes x and y are opportunely broadcast to all other nodes in T; also, during this phase, all nodes compute other values that will be used in the evaluation of the candidate swap edges.

In particular, the Broadcasting step starts in x and in y towards the nodes in $T_{x\setminus y}$ and $T_{y\setminus x}$, respectively. In the following we will describe the broadcast in $T_{x\setminus y}$; the argument is symmetric for $T_{y\setminus x}$.

Node x sends to all its neighbors but y the following values: $\mathcal{W}(T)$, $n(T)$, $\mathcal{D}(x,T)$, all available to x after previous step, by Lemma 2; we will refer to these values as the *broadcast set of* x. An internal node v in $T_{x\setminus y}$ waits until it receives the broadcast set from node k, with k the father of v in $T_{x\setminus y}$; note also that k is the recipient of v as defined in the Collecting step. At this time, it will compute $\mathcal{D}(v,T)$, and will forward its broadcast set to all of its neighbors but k: these nodes are the children of v in $T_{x\setminus y}$. We have:

Theorem 1. *After the saturation process, each node $u \in T$ has locally the following values: $\mathcal{W}(T)$, $\mathcal{D}(u,T)$, $n(T)$, and $\mathcal{W}(T_{u\setminus u_j})$, $\mathcal{D}(u,T_{u\setminus u_j})$, and $n(T_{u\setminus u_j})$ for each neighbor u_j of u in T. All these values are computed within the complexity of the saturation algorithm, that is in linear message complexity.*

Property 2. Let T be any spanning tree over a 2-connected graph G. If f is a swap edge for $e = (u,v) \in T$ in the rooted tree T_r, for any $r \in T$, then f is also a swap edge for e in T_q, for any other node q in T.

Hence, since the presence of a root node does not affect the property of an edge to be a swap edge, in the Exchanging phase one of the two saturated nodes is also elected as leader (e.g., the one with the smallest ID): this node is selected as the root of T. This info is clearly broadcasted to all other nodes in the tree during the Broadcasting phase. In the following, we will refer to the rooted version of T as T_r, with r the root of the tree. Also, we assume that a node u knows the weight of all its incident links, and can distinguish those that are part of T_r from those that are not; of those that are part of T_r, u can distinguish the one that leads to its parent from those leading to its children. Furthermore, we assume that each node u knows its distance from r and the distances of its neighbors from r. Finally, we assume that each node knows its pre-order and inverse pre-order labeling in T_r, as described in [2], as well as the labels of its neighbors: this labeling is necessary to verify the swap edge property introduced in Property 1. If not available, this information can be easily and efficiently acquired after the Saturation phase with a search of the tree.

3.2 Swap Phase

At the end of this phase, each node u will know the set of candidate swap edges for the possible failure of edge $(u, parent(u))$, that we will refer to as the *swap set* of u, and denote by $\mathcal{S}(u)$. The swap set of u is a set of 6-tuples $\langle (u',v'), d(u',v'), \mathcal{D}(u',T), \mathcal{D}(v',T),\ d(u',r), d(v',r)\rangle$, where: $f = (u',v')$ is a swap edge for $e = (u, parent(u))$; $\mathcal{D}(u',T)$ and $\mathcal{D}(v',T)$ are as computed in the Saturation phase; and $d(u',v')$, $d(u',r)$ and $d(v',r)$ are the weights of edge (u',v'), and the distance of u' and v' from r in T_r, respectively. Note that, by Property 1, exactly one node between u' and v' must be descendant of u in T_r; here, without loss of generality, we assume that u' is such a node.

This phase proceeds in a leaves-to-root fashion. The computation starts in the leaves of T_r: given a leaf u, any edge in G that is incident to u, with the obvious exception of $(u, parent(u))$, is clearly a candidate swap edge for u; hence, it is

in $\mathcal{S}(u)$. This set is sent to $parent(u)$. An internal node $u \in T_r$ waits until it receives the candidate swap edges computed by *all* of its children. For each tuple $t = \langle (u', v'), \cdot, \cdot, \cdot, \cdot, \cdot \rangle$ received from the children, it check whether (u', v') is a swap edge for $e = (u, parent(u))$: in this case, this tuple is inserted in $\mathcal{S}(u)$. Then, u checks whether some of its incident edges in $G - e$ are swap edges for $e = (u, parent(u))$: in this case, these edges are placed in $\mathcal{S}(u)$ as well. The check is again made by testing Property 1. Therefore, we can state the following:

Lemma 3. *After the Swap phase, each node u in T_r correctly computes $\mathcal{S}(u)$, containing information on all the candidate swap edges for $(u, parent(u))$.*

Since in the worst case a node u can have $O(m)$ swap edges for $(u, parent(u))$, we can state that

Lemma 4. *The Swap phase takes $O(nm)$ message complexity in the worst case.*

3.3 Best Swap Phase

In the Best Swap Phase, the algorithm selects, for each node u, the best swap edge for $e = (u, parent(u))$ among those in $\mathcal{S}(u)$. In other words, it selects the candidate swap edge $f = (u_i, v_i)$, with $\langle f, \cdot, \cdot, \cdot, \cdot, \cdot \rangle \cdot \in \mathcal{S}(u)$, such that $\mathcal{W}(T'_{e|f})$ is minimum, where $T'_{e|f}$ is the tree obtained from T by deleting e and adding f.

Lemma 5. *For the failing edge $e = (u, parent(u)) \in T_r$, the value of $\mathcal{W}(T'_{e|f})$ for the swap edge $f = (u_i, v_i)$ for e, with $u_i \in T_{u \setminus v}$ and $v_i \in T_{v \setminus u}$, is given by:*

$$\mathcal{W}(T'_{e|f}) = \mathcal{W}(T_{u \setminus v}) + \mathcal{W}(T_{v \setminus u}) + n(T_{v \setminus u}) \cdot \mathcal{D}(u_i, T_{u \setminus v}) + \\ + n(T_{u \setminus v}) \cdot (d(u_i, v_i) \cdot n(T_{v \setminus u}) + \mathcal{D}(v_i, T_{v \setminus u})). \tag{6}$$

Also, after the Swap phase, node u can locally select the best swap edge for e.

Hence, after the Best Swap Edge phase, for each edge $e = (u, v) \in T_r$, with $v = parent(u)$, the best swap edge f^* for e is computed.

Theorem 2. *Given a 2-connected graph $G = (V_G, E_G)$ and any spanning tree T for G, the BSRT problem can be solved in $O(m \cdot n)$ message complexity in the worst case.*

Corollary 1. *Given a 2-connected graph $G = (V_G, E_G)$ and its Minimum Routing Cost Spanning Tree, the BSRT problem can be solved in $O(m \cdot n)$ message complexity in the worst case.*

References

1. Liebchen, C., Wunsch, G.: The zoo of the tree spanner problems. Discrete Applied Mathematics 156, 569–587 (2008)
2. Flocchini, P., Mesa Enriques, A., Pagli, L., Prencipe, G., Santoro, N.: Point-of-failure shortest-path rerouting: computing the optimal swap edges distributively. IEICE Transactions on Information and Systems 2(E89-D), 700–708 (2006)

3. Johnson, D.S., Lenstra, J.K., Kan, A.R.: The complexity of the network design problem. Networks 8, 279–285 (1978)
4. Nardelli, E., Proietti, G., Widmayer, P.: Swapping a failing edge of a single source shortest-paths tree is good and fast. Algoritmica 35, 56–74 (2003)
5. Santoro, N.: Design and Analysis of Distributed Algorithms. Wiley, Chichester (2007)
6. Wu, B.Y., Hsiao, C.Y., Chao, K.M.: The swap edges of a multiple-sources routing tree. Algorithmica 50(3), 299–311 (2008)