7.3.3 Global Description of the GHS Algorithm.

We will first describe how the algorithm operates in a global fashion, i.e., from the fragments point of view. We then describe the local algorithm that each node must execute in order to obtain this global operation of the fragments.

A computation of the GHS algorithm proceeds according to the following steps.

(1) A collection of fragments is maintained, such that the union of all fragments contains all nodes.
(2) Initially this collection contains each node as a one-node fragment.
(3) The nodes in a fragment cooperate to find the lowest-weight outgoing edge of the fragment.
(4) When the lowest-weight outgoing edge of a fragment is known, the fragment will be combined with another fragment by adding the outgoing edge, in cooperation with the other fragment.
(5) The algorithm terminates when only one fragment remains.

The efficient implementation of these steps requires the introduction of some notation and mechanisms.

(1) *Fragment name.* To determine the lowest-weight outgoing edge it must be possible to see whether an edge is an outgoing edge or leads to a node in the same fragment. To this end each fragment will have a name, which will be known to the processes in that fragment. Processes test whether an edge is internal or outgoing by a comparison of their fragment names.
(2) *Combining large and small fragments.* When two fragments are combined, the fragment name of the processes in at least one of the fragment changes, which requires an update to take place in every node of at least one of the two fragments. To keep this update efficient, the combining strategy is based on the idea that *the smaller of two fragments* combines into *the larger of the two* by adopting the fragment name of the larger fragment.
(3) *Fragment levels.* A little thinking reveals that the decision about which of two fragments is the larger should *not* be based on the number of nodes in the two fragments. This would make it necessary to update the fragment size in every process of *both* the larger *and* the smaller constituent fragment, thus spoiling the desirable property that an update is necessary only in the smaller. Instead, each

fragment is assigned a *level*, which is 0 for an initial one-node fragment. It is allowed that a fragment F_1 combines into a fragment F_2 with higher level, after which the new fragment $F_1 \cup F_2$ has the level of F_2. The new fragment also has the fragment name of F_2, so no updates are necessary for the nodes in F_2. It must also be possible for two fragments of the same level to combine; in this case the new fragment has a new name and its level is one higher than the level of the combining fragments. The new name of the fragment is the weight of the edge by which the two fragments are combined, and this edge is called the *core edge* of the new fragment. The two nodes connected by the core edge are called the *core nodes*.

Lemma 7.20 *If these combining rules are obeyed, the number of times a process changes its fragment name or level is at most $N \log N$.*

Proof. The level of a process never decreases, and only when it increases does the process change its fragment name. A fragment of level L contains at least 2^L processes, so the maximum level is $\log N$, which implies that each individual process increases its fragment level at most $\log N$ times. Hence, the overall total number of fragment name and level changes is bounded by $N \log N$. □

Summary of combining strategy. A fragment F with name FN and level L is denoted as $F = (FN, L)$; let e_F denote the lowest-weight outgoing edge of F.

Rule A. If e_F leads to a fragment $F' = (FN', L')$ with $L < L'$, F combines into F', after which the new fragment has name FN' and level L'. These new values are sent to all processes in F.

Rule B. If e_F leads to a fragment $F' = (FN', L')$ with $L = L'$ and $e_{F'} = e_F$, the two fragments combine into a new fragment with level $L+1$ and name $\omega(e_F)$. These new values are sent to all processes in F and F'.

Rule C. In all other cases (i.e., $L > L'$ or $L = L'$ and $e_{F'} \neq e_F$) fragment F must wait until rule A or B applies.

7.3.4 Detailed Description of the GHS Algorithm

Node and link status. Node p maintains the variables as indicated in Algorithm 7.10, including the channel status $stach_p[q]$ for each channel pq. This status is *branch* if the edge is known to be in the MST, *reject* if it is known not to be in the MST, and *basic* if the edge is still unused. The

```
var $state_p$                           : $(sleep, find, found)$ ;
    $stach_p[q]$                        : $(basic, branch, reject)$   for each $q \in Neigh_p$ ;
    $name_p$, $bestwt_p$                : real ;
    $level_p$                           : integer ;
    $testch_p$, $bestch_p$, $father_p$  : $Neigh_p$ ;
    $rec_p$                             : integer ;

(1) As the first action of each process, the algorithm must be initialized:
    begin let $pq$ be the channel of $p$ with smallest weight ;
            $stach_p[q] := branch$ ; $level_p := 0$ ;
            $state_p := found$ ; $rec_p := 0$ ;
            send ⟨ connect, 0 ⟩ to $q$
    end

(2) Upon receipt of ⟨ connect, $L$ ⟩ from $q$:
    begin if $L < level_p$ then (* Combine with Rule A *)
              begin $stach_p[q] := branch$ ;
                    send ⟨ initiate, $level_p, name_p, state_p$ ⟩ to $q$
              end
          else if $stach_p[q] = basic$
                  then (* Rule C *) process the message later
                  else (* Rule B *) send ⟨ initiate, $level_p + 1, \omega(pq), find$ ⟩ to $q$
    end

(3) Upon receipt of ⟨ initiate, $L, F, S$ ⟩ from $q$:
    begin $level_p := L$ ; $name_p := F$ ; $state_p := S$ ; $father_p := q$ ;
          $bestch_p := udef$ ; $bestwt_p := \infty$ ;
          forall $r \in Neigh_p : stach_p[r] = branch \wedge r \neq q$ do
                 send ⟨ initiate, $L, F, S$ ⟩ to $r$ ;
          if $state_p = find$ then begin $rec_p := 0$ ; $test$ end
    end
```

Algorithm 7.10 THE GALLAGER–HUMBLET–SPIRA ALGORITHM (PART 1).

communication in a fragment to determine the lowest-weight outgoing edge takes place via the *branch* edges in the fragment. For process p in the fragment, $father_p$ is the edge leading to the core edge of the fragment. The state of node p, $state_p$, is *find* if p is currently engaged in the fragment's search for the lowest-weight outgoing edge and *found* otherwise. The algorithm is given as Algorithm 7.10/7.11/7.12. Sometimes the processing of a message must be deferred until a local condition is satisfied. It is assumed that in this case the message is stored, and later retrieved and treated as if it had been received at that moment. If a process receives a message while it is still in the state *sleep*, the algorithm is initialized in that node (by executing action (1)) before the message is processed.

```
(4) procedure test:
    begin if $\exists q \in Neigh_p : stach_p[q] = basic$ then
              begin $testch_p := q$ with $stach_p[q] = basic$ and $\omega(pq)$ minimal ;
                    send ⟨ test, $level_p$, $name_p$ ⟩ to $testch_p$
              end
          else begin $testch_p := udef$ ; report end
    end

(5) Upon receipt of ⟨ test, $L$, $F$ ⟩ from $q$:
    begin if $L > level_p$ then      (* Answer must wait! *)
              process the message later
          else if $F = name_p$ then (* internal edge *)
                   begin if $stach_p[q] = basic$ then $stach_p[q] := reject$ ;
                         if $q \neq testch_p$
                            then send ⟨ reject ⟩ to $q$
                            else test
                   end
               else send ⟨ accept ⟩ to $q$
    end

(6) Upon receipt of ⟨ accept ⟩ from $q$:
    begin $testch_p := udef$ ;
          if $\omega(pq) < bestwt_p$
             then begin $bestwt_p := \omega(pq)$ ; $bestch_p := q$ end ;
          report
    end

(7) Upon receipt of ⟨ reject ⟩ from $q$:
    begin if $stach_p[q] = basic$ then $stach_p[q] := reject$ ;
          test
    end
```

Algorithm 7.11 THE GALLAGER–HUMBLET–SPIRA ALGORITHM (PART 2).

Finding the lowest-weight outgoing edge. The nodes in a fragment cooperate to find the lowest-weight outgoing edge of the fragment, and when the edge is found a ⟨ **connect**, L ⟩ message is sent through it; L is the level of the fragment. If the fragment consists of a single node, as is the case after the initialization of this node, the required edge is simply the lowest-weight adjacent edge of this node; see (1). A ⟨ **connect**, 0 ⟩ message is sent via this edge.

Next consider the case that a new fragment is formed by combining two fragments, connection being by edge $e = pq$. If the two combined fragments were of the same level, L, both p and q will have sent a ⟨ **connect**, L ⟩ message via e, and will have received a ⟨ **connect**, L ⟩ message in return while

```
(8) procedure report:
    begin if $rec_p = \#\{q : stach_p[q] = branch \wedge q \neq father_p\}$
                 and $testch_p = udef$ then
              begin $state_p := found$ ; send ⟨ report, $bestwt_p$ ⟩ to $father_p$ end
    end

(9) Upon receipt of ⟨ report, $\omega$ ⟩ from $q$:
    begin if $q \neq father_p$
          then (* reply for initiate message *)
               begin if $\omega < bestwt_p$ then
                        begin $bestwt_p := \omega$ ; $bestch_p := q$ end ;
                     $rec_p := rec_p + 1$ ; report
               end
          else (* pq is the core edge *)
               if $state_p = find$
                 then process this message later
                 else if $\omega > bestwt_p$
                      then changeroot
                      else if $\omega = bestwt_p = \infty$ then stop
    end

(10) procedure changeroot:
     begin if $stach_p[bestch_p] = branch$
           then send ⟨ changeroot ⟩ to $bestch_p$
           else begin send ⟨ connect, $level_p$ ⟩ to $bestch_p$ ;
                      $stach_p[bestch_p] := branch$
                end
     end

(11) Upon receipt of ⟨ changeroot ⟩:
     begin changeroot end
```

Algorithm 7.12 THE GALLAGER–HUMBLET–SPIRA ALGORITHM (PART 3).

the status of e is *branch*; see action (2). Edge pq becomes the core edge of the fragment, and p and q exchange an ⟨ **initiate**, $L+1, N, S$ ⟩ message, giving the new level and name of the fragment. The name is $\omega(pq)$ and the status *find* causes each process to start searching for the lowest-weight outgoing edge; see action (3). The message ⟨ **initiate**, $L+1, N, S$ ⟩ is flooded to each node in the new fragment. If the level of p was smaller than the level of q, p will have sent a ⟨ **connect**, L ⟩ message via e, and will have received an ⟨ **initiate**, L', N, S ⟩ message in return from q; see action (2). In this case, L' and N are the current fragment level and name of q, and the name and level of the nodes on q's side of the edge do not change. On p's side of the edge the initiate message is flooded to all the nodes (see action (3)), causing every process to update its fragment name and level. If q is currently searching for the lowest-weight outgoing edge ($S = \mathit{find}$) the processes in p's fragment join the search by calling *test*.

Each process in the fragment searches through its edges (if it has any, see (4), (5), (6), and (7)) to see if there is one leading out of the fragment, and if so, chooses the one of lowest weight. The lowest-weight outgoing edge is reported for each subtree using $\langle$ **report**, ω $\rangle$ messages; see (8). Node p counts the number of $\langle$ **report**, ω $\rangle$ messages it receives, using the variable rec_p, which is set to 0 when the search starts (see (3)) and incremented with each receipt of a $\langle$ **report**, ω $\rangle$ message; see (9). Each process sends a $\langle$ **report**, ω $\rangle$ message to its father when it has received such a message from each of its sons and has finished the local search for an outgoing edge.

The $\langle$ **report**, ω $\rangle$ messages are sent in the direction of the core edge by each process, and the messages of the two core nodes cross on the edge; both receive the message from their *father*; see (9). Each core node waits until it has sent a $\langle$ **report**, ω $\rangle$ message itself before it processes the message of the other process. When the two $\langle$ **report**, ω $\rangle$ messages of the core nodes have crossed, the core nodes know the weight of the lowest-weight outgoing edge. The algorithm terminates at this point if no outgoing edge was reported at all (both messages report the value ∞).

If an outgoing edge was reported, the best edge is found by following the *bestch* pointer in each node, starting from the core node on whose side the best edge was reported. A $\langle$ **connect**, L $\rangle$ message must be sent through this edge, and all *father* pointers in the fragment must point in this direction; this is done by sending a $\langle$ **changeroot** $\rangle$ message. The core node on whose side the lowest-weight outgoing edge is located sends a $\langle$ **changeroot** $\rangle$ message, which is sent via the tree to the lowest-weight outgoing edge; see (10) and (11). When the $\langle$ **changeroot** $\rangle$ message arrives at the node incident to the lowest-weight outgoing edge, this node sends a $\langle$ **connect**, L $\rangle$ message via the lowest-weight outgoing edge.

Testing the edges. To find its lowest-weight outgoing edge, node p inspects its *basic* edges one by one in increasing order of weight; see (4). The local search for an edge ends either when no edge remains (all edges are *reject* or *branch*), see (4), or when one edge is identified as outgoing; see (6). Because of the order in which p inspects the edges, if p identifies one edge as outgoing, this must be the lowest-weight edge outgoing from p.

To inspect edge pq, p sends a $\langle$ **test**, $level_p$, $name_p$ $\rangle$ message to q and waits for an answer, which can be a $\langle$ **reject** $\rangle$, $\langle$ **accept** $\rangle$, or $\langle$ **test**, L, F $\rangle$ message. A $\langle$ **reject** $\rangle$ message is sent by process q (see (5)) if q finds that p's fragment name, as in the test message, coincides with q's fragment name; node q also rejects the edge in this case. On receipt of the $\langle\,\textbf{reject}\,\rangle$ message p rejects edge pq and continues the local search; see (7). The $\langle\,\textbf{reject}\,\rangle$ message is omitted if the edge pq was just used by q also to send a $\langle\,\textbf{test}, L, F\,\rangle$ message; in this case q's $\langle\,\textbf{test}, L, F\,\rangle$ message serves as the reply to p's message; see (5). If the fragment name of q differs from p's, an $\langle\,\textbf{accept}\,\rangle$ message is sent. On receipt of this message p terminates its local search for outgoing edges with edge pq as the best local choice; see (6).

The processing of a $\langle\,\textbf{test}, L, F\,\rangle$ message by p is deferred if $L > level_p$. The reason is that p and q may actually belong to the same fragment, but the $\langle\,\textbf{initiate}, L, F, S\,\rangle$ message has not yet reached p. Node p could erroneously reply to q with an $\langle\,\textbf{accept}\,\rangle$ message.

Combining the fragments. After the lowest-weight outgoing edge of a fragment $F = (name,\ level)$ has been determined, a $\langle\,\textbf{connect}, level\,\rangle$ message is sent via this edge, and is received by a node belonging to a fragment $F' = (name',\ level')$. Call the process sending the $\langle\,\textbf{connect}, level\,\rangle$ message p and the process receiving it q. Node q has earlier sent an $\langle\,\textbf{accept}\,\rangle$ message to p in reply to a $\langle\,\textbf{test}, level, name\,\rangle$ message, because the search for the best outgoing edge in p's fragment has terminated. The waiting introduced before answering test messages (see (5)) implies that $level' \geq level$.

According to the combining rules discussed earlier, the $\langle\,\textbf{connect}, level\,\rangle$ is answered with an $\langle\,\textbf{initiate}, L, F, S\,\rangle$ message in two cases.

Case A: If $level' > level$, p's fragment is absorbed; the nodes in this fragment are informed about their new fragment name and level by a message $\langle\,\textbf{initiate}, level', name', S\,\rangle$, which is flooded to all nodes in fragment F. The entire absorbed fragment F becomes a subtree of q in the spanning tree of fragment F' and if q is currently engaged in a search for the best outgoing edge of fragment F', all processes in F must participate. This is why q includes its state (*find* or *found*) in the $\langle\,\textbf{initiate}, level', name', S\,\rangle$ message.

Case B: If the two fragments have the same level and the best outgoing edge of fragment F' is also pq, a new fragment is formed, of which the level is one higher and the name is the weight of edge pq; see (2). This case occurs if the two levels are equal and the connect message is received via a branch edge; observe that the status of an edge becomes *branch* if a connect message is sent through it.

If neither of these two cases occurs, fragment F must wait until either q

sends a $\langle$ **connect**, $L \rangle$ message or the level of q's fragment has increased sufficiently to make Case A applicable.

Correctness and complexity. From the detailed description of the algorithm it should be clear that the edge through which a fragment sends a $\langle$ **connect**, $L \rangle$ message is indeed the lowest-weight outgoing edge of the fragment. Together with Proposition 7.19 this implies that the MST is computed correctly if each fragment indeed sends such a message and joins the other fragment, in despite of the waiting induced by the algorithm. The most complex message contains one edge weight, one level (up to $\log N$) and a constant number of bits to indicate message type and node state.

Theorem 7.21 *The Gallager–Humblet–Spira algorithm (Algorithm 7.10/ 7.11/7.12) computes the minimal spanning tree, using at most* $5N \log N + 2|E|$ *messages.*

Proof. Deadlock potentially arises in situations where nodes or fragments must wait until some condition occurs in another node or fragment. The waiting introduced for $\langle$ **report**, $\omega \rangle$ messages on the core edge does not lead to a deadlock because each core node eventually receives reports from all sons (unless the fragment as a whole waits for another fragment), after which the message will be processed.

Consider the case where a message of fragment $F_1 = (level_1, name_1)$ arrives at a node of fragment $F_2 = (level_2, name_2)$. A $\langle$ **connect**, $level_1 \rangle$ message must wait if $level_1 \geq level_2$ and no $\langle$ **connect**, $level_2 \rangle$ message has been sent through the same edge by fragment F_2; see (2). A $\langle$ **test**, $level_1, name_1 \rangle$ message must wait if $level_1 > level_2$; see (5). In all cases where F_1 waits for F_2, one of the following holds.

(1) $level_1 > level_2$;
(2) $level_1 = level_2 \wedge \omega(e_{F_1}) > \omega(e_{F_2})$;
(3) $level_1 = level_2 \wedge \omega(e_{F_1}) = \omega(e_{F_2})$ and F_2 is still searching for its lowest-weight outgoing edge. (As e_{F_1} is an outgoing edge of F_2 it is not possible that $\omega(e_{F_2}) > \omega(e_{F_1})$.)

Thus no deadlock cycle can occur.

Each edge is rejected at most once and this requires two messages, which bounds the number of reject messages plus test messages leading to rejections to $2|E|$. At any level, a node receives at most one initiate and one accept message, and sends at most one report, one changeroot *or* connect message, and one test message not leading to a rejection. At level zero no accept messages are received and no report or test messages are sent. At the highest level each node only sends a report message and receives one initiate message. The total number of messages is therefore bounded by $2|E| + 5N \log N$. □

7.3.5 Discussion and Variants of the GHS Algorithm

The Gallager–Humblet–Spira algorithm is one of the most sophisticated wave algorithms, requiring only local knowledge and having optimal message complexity. The algorithm can easily be extended so that it elects a leader, using only two more messages. The algorithm terminates in two nodes, namely core nodes of the last fragment (spanning the entire network). Instead of executing **stop**, the core nodes exchange their identities and the smaller of them becomes leader.

A number of variations and related algorithms have been published. The GHS algorithm may require $\Omega(N^2)$ time if some nodes start the algorithm very late. If an additional wake-up procedure is used (taking at most $2|E|$ more messages) the time complexity of the algorithm is $5N \log N$; see Exercise 7.11. Awerbuch [Awe87] has shown that the time complexity of the algorithm can be improved to $O(N)$, while keeping the message complexity order optimal, i.e., $O(|E| + N \log N)$.

Afek *et al.* [ALSY90] have adapted the algorithm to compute a spanning forest with favorable properties, namely that the diameter of each tree and the number of trees are $O(\sqrt{N})$. Their algorithm distributively computes a clustering of the network as indicated in Lemma 4.47 and a spanning tree and a center for each cluster.

One may ask if the construction of arbitrary spanning trees can be done more efficiently than the construction of the minimal spanning trees, but Theorem 7.15 implies a lower bound of $\Omega(N \log N + |E|)$ on the construction of arbitrary spanning trees as well. Johansen *et al.* [JJN$^+$87] give an algorithm for computing an arbitrary spanning tree that uses $3N \log N + 2.|E| + O(N)$ messages, thus improving on the GHS algorithm by a constant factor if the network is sparse. Bar-Ilan and Zernik [BIZ89] presented an algorithm that computes *random* spanning trees, where each possible spanning tree is chosen with equal probability. The algorithm is randomized and uses an expected number of messages that is between $O(N \log N + |E|)$ and $O(N^3)$, depending on the topology of the network.

While the construction of arbitrary and of minimal spanning trees is of equal complexity in arbitrary networks, this is not true in cliques. Korach, Moran and Zaks [KMZ85] have shown that the construction of a minimal spanning tree in a weighted clique requires the exchange of $\Omega(N^2)$ messages.