17.8 Self-stabilizing algorithms for spanning-tree construction

In this section, we discuss a set of representative self-stabilizing algorithms for constructing spanning trees \footnote{[43]}. 

17.8.1 Dolev, Israeli, and Moran algorithm

Dolev \textit{et al.} \cite{40} developed a self-stabilizing BFS spanning-tree construction algorithm for semi-uniform systems with a central demon under read/write atomicity. In the algorithm, every node maintains two variables: (i) a pointer to one if its incoming edges (this information is kept in a bit associated with each communication register), and (ii) an integer measuring the distance in hops to the root of the tree. The distinguished node in the network acts as the root.

The algorithm works as follows: the network nodes periodically exchange their distance value (current distance from the root node) with each other. After reading the distance values of all neighbors, a network node chooses the neighbor with minimum distance $\text{dist}$ as its new parent. It then writes its own distance into its output registers, which is $\text{dist} + 1$. The distinguished root node does not read the distance values of its neighbors and always sends a value of 0.

The algorithm stabilizes starting from the root process. After sufficient activations of the root, it has written 0 values into all of its output variables. These values will not change anymore. Note that without a distinguished root process the distance values in all nodes would grow without bound. More specifically, after reading all neighbors values for $k$ times, the distance value of a process is at least $k + 1$. This means that, after the root has written its output registers, the direct neighbors of the root – after inspecting their input variables – will see that the root node has the minimum distance of all other nodes (the other nodes have distance at least 1). Hence, all direct neighbors of the root will select the root as their parent and update their distance correctly to 1. This line of reasoning can be continued incrementally for all other distances from the root. That is, after all nodes at distance $d$ from the root have computed their distance from the root correctly and written it in their registers, their registers no longer change and nodes at distance $d + 1$ from the root are ready to compute their distance from the root. After $O(\delta)$ update cycles, the entire tree will have stabilized.
Variables:

- \( no\_neighbors \) = Number of processor’s neighbors
- \( i \) = the writing processor
- \( m \) = for whom the data is written
- \( lr_j \) (local register \( r_j \)) the last value of \( r_j \) read by \( P_i \)

Root Node:
{do forever}
while TRUE do
  for \( m := 1 \) to \( no\_neighbors \) do
    write \( lr_{im} := <0,0> \)
  end
end

Other Nodes:
{do forever}
while TRUE do
  for \( m := 1 \) to \( no\_neighbors \) do
    \( lr_{mi} := \text{read}(lr_m) \)
    FirstFound := false
    \( dist := 1 + \min(lr_m,\text{dist}) \) \( \forall m: 1 \leq m \leq no\_neighbors \)
    for \( m := 1 \) to \( no\_neighbors \) do
      if not FirstFound and \( lr_{mi,\text{dist}} = dist - 1 \) then
        write \( r_{im} := <1, dist> \)
        FirstFound := true
      else write \( r_{im} := <0, dist> \)
    end
  end
end

Algorithm 17.2 Dolev et al.’s spanning-tree construction algorithm for \( P_i \) [40].

Dolev et al.’s self-stabilizing algorithm for constructing spanning trees is shown in Algorithm 17.2. Two neighbors \( P_i \) and \( P_j \) communicate with each other by reading from and writing to two shared registers, \( r_{ij} \) and \( r_{ji} \). To communicate, \( P_i \) writes to \( r_{ij} \) and reads from \( r_{ji} \) and \( P_j \) writes to \( r_{ji} \) and reads from \( r_{ij} \).

The root node repeatedly writes values \(<0,0>\) in the registers of all of its neighbors. All other processors repeatedly perform the following steps: in each iteration, the processor reads the registers of all of its neighbors and computes the a value for variable \( \text{dist} \) as follows: it chooses the minimum distance of their neighbors, sets its \( \text{dist} \) variable to the minimum distance plus 1, and
updates the registers of its neighbors. The internal variable corresponding to register $r_{ij}$ is denoted by $lr_{ij}$. It stores the last value of $r_{ji}$ that is read by $P_i$.

A snapshot of the system state in Dolev et al.’s self-stabilizing algorithm is given in Figure 17.4. This algorithm has been used as the basis for a topology update algorithm in dynamic networks. Based on a similar idea, Collin and Dolev [31] present a semi-uniform spanning-tree algorithm under a central demon and read/write atomicity that constructs a DFS tree (instead of a BFS tree). A similar algorithm, which also constructs a DFS tree but uses composite atomicity, was developed by Herman. In this algorithm, the outgoing links at every process are ordered, and the DFS tree is defined as the tree resulting from a DFS graph traversal always selecting the smallest outgoing edge. Instead of writing its current level into the output registers, it writes a representation of its current estimate of the path (the sequence of outgoing link identifiers) to the root. The root repeatedly writes the “empty path” ($\perp$) to its output registers. If a node has $k$ neighbors, there are $k$ alternative paths to choose from. From these, the node chooses the path that is minimal according to a lexicographic order that prefers smaller link identifiers. For example, $(\perp) < (\perp, 1) < (\perp, 1, 1) < (\perp, 2) < (1)$. Thus, a node does not choose the shortest path to the root but a path along the smallest link identifiers.

The memory requirement for the DFS algorithm is $O(n \log K)$ bits, where $K$ is an upper bound on the maximum degree of a node. The time complexity is $O(\delta n K)$ rounds.