8 Universal Election Protocols

We have so far studied in detail the election problem in *specific* topologies; i.e., we have developed solution protocols for restricted classes of networks, exploiting in their design all the graph properties of those networks so to minimize the costs and increase the efficiency of the protocols. In this process we have learned some strategies and principles which are however very general (e.g., the notion of *electoral stages*), as well as the use of known techniques (e.g., broadcasting) as modules of our solution.

We will now focus on the main issue, the design of *universal election protocols*, that is protocols which run in every network, requiring no a priori knowledge of the topology of the network nor of its properties (not even its size !). In terms of communication software, such protocols are obviously are totally *portable*, and thus highly desirable.

We will describe two such protocols, radically different from each other. The first, *Mega-Merger*, constructs a rooted spanning-tree, is highly efficient (optimal in the worst case); the protocol is however rather complex both in terms of specifications and analysis, and its correctness is still without a simple formal proof. The other, *Yo-Yo*, is a minimum-finding protocol which is exceedingly simple to specify and to prove correct; its real cost is however not yet known.

8.1 Mega-Merger (Algorithme GHS)

In this section we will discuss the design of an efficient algorithm for leader election, called *Mega-Merger*. This protocol is topology independent (i.e., universal) and constructs a (minimum cost) rooted spanning tree of the network.

Nodes are small villages, and edges are roads each with a different name and distance. The goal is to have all villages merge into one large megacity. A city (even a small village will be considered such) always tries to merge with the closest neighbouring city.

When merging, there are several important issues that must be resolved. First and foremost, the *naming of the new city*. The resolution of this issue depends on how far the involved cities have progressed in the merging process, i.e. on the *level* they have reached, and on whether the merger decision is shared by both cities.

The second issue to be resolved during a merging is the decision of which roads of the new city will be serviced by *public transports*. When a merge occurs, the roads of the new city serviced by public transports will be the roads of the two cities already serviced plus only the shortest road connecting them.

Let us clarify some of these concepts and notions, as well as the basic rules of the game.

- (1) A *city* is a rooted tree; the nodes are called *districts*, the root is also known as *downtown*.
- (2) Each city has a level and a unique name; all districts eventually know the name and the level of their city.

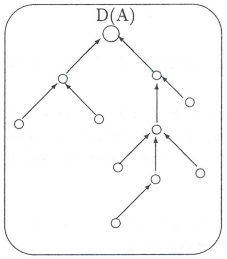


Figure 49: A city is a tree rooted in its downtown.

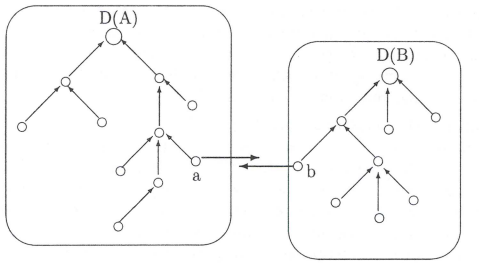
- (3) Edges are *roads*, each with a distinct name and distance. The city roads are only those serviced by public transport.
- (4) Initially, each node is a city with just one district, itself, and no roads. All cities are initially at the same level.

Note that, as a consequence of rule (1), every district knows the direction (i.e., which of its links in the tree leads) to its downtown (Figure 49).

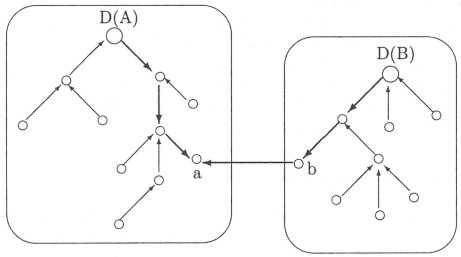
- (5) A city must merge with its closest neighbouring city. To request the merging, a *Let-us-Merge* message is sent on the shortest road connecting it to that city.
- (6) The decision to request for a merger must originate from downtown, and there cannot be more than one request at a time from the same city.
- (7) When a merge occurs, the roads of the new city serviced by public transports will be the roads of the two cities already serviced plus only the shortest road connecting them.

Thus, to merge, the downtown of city A will first determine the shortest link, which we shall call the *merge link*, connecting it to a neighbouring city; once this is done, a *Let-us-Merge* is sent through that link; the message will contain information identifying the city and the chosen merge link. Once the message reaches the other city, the actual merger can start to take place. Let us examine the components of this entire process in some details.

We will consider city A, denote by D(A) its downtown, by $level(A)$ its current level, and by $e(A) = (a, b)$ the *merge link* connecting A to its closest neighbouring city; let B be such a city.



(a)



(b)

Figure 51: Friendly Merger (a) The two cities have the same level and choose the same merge link. (b) The new downtown is the exit node (a or b) with smallest id.

Then each will broadcast this information to its old city; as a result, all districts of A and B will join the new city, and know its name and its level.
Both A and B must be transformed so they are rooted in the new downtown. As discussed in the case of absorption, it is sufficient to "flip" the logical direction only of the edges in the path from the a to the old downtown of A, and of those in the path from b to the old downtown of B (Figure 51).

Suspension
In two cases (rules (10) and (11)), the merge request of A must be suspended: b will then locally enqueue the message until the level of its city is such that it can apply rule (8) or (9). Notice that, in case of suspension, nobody from city A knows that their request has been suspended; because of rule (6), no other request can be launched from A.

8.1.2 Choosing the merging edge

According to rule (6), the choice of the merging edge $e(A)$ in A is made by the downtown $D(A)$; according to rule (5), $e(A)$ must be the shortest road connecting A to a neighbouring city. Thus, $D(A)$ needs to find the minimum length among all the edges incident on the nodes of the rooted tree A; this will be done as following.

- (5.1) Each district a_i of A determines the length d_i of the shortest road connecting it to another city (if none goes to another city, then $d_i = \infty$).
  (5.2) $D(A)$ computes the smallest of all the d_i.

Part (5.2) is easy to accomplish; it is just a minimum-finding in a rooted tree, which we have discussed in Section **??**.

Concentrate on part (5.1), and consider a district a_i; it must find, among its incident edges the shortest one that leads to an another city.

IMPORTANT. Obviously, a_i does not need to consider the *internal roads* (i.e., those that connect it to other districts of A). Unfortunately, if a link is *unused* i.e., no message has been sent or received through it, it is impossible for a_i to know if this road is internal or leads to a neighbouring city (Figure 52). In other words, a_i must try also the internal unused roads.

Thus, a_i will determine the shortest unused edge e, prepare a *Outside?* message, send it on e and wait for a reply. Consider now the district c on the other side of e, which receives this message; c knows the $name(C)$ and the $level(C)$ of its city (which could however be changing).

If $name(A) = name(C)$ (recall that the message contains the name of A), c will reply *Internal* to a_i, the road e will be marked as internal (and no longer used in the protocol) by both districts, and a_i will restart its process.

If $name(A) \neq name(C)$ it does not necessarily mean that the road is not internal. In fact, it is possible that, while c is processing this message, its city C is being absorbed by A ! Observe that, in this case, $level(C)$ must be *smaller* than $level(A)$ (because, by rule (8) only a city with smaller level will be absorbed). This means that, if $name(A) \neq name(C)$ but

Progress and Deadlock

We will first discuss the progress of the computation and the absence of deadlock . To do so, let us pinpoint the cases when the activity of a city C is halted by a district d of another city D. This can occur only when computing the merge edge, or when requesting a merger on the merge edge $e(C)$; more precisely, there are four cases:
(i) when computing the merge edge, a district c of C sends the *Outside?* message to d and D has a smaller level than C;
(ii) after receiving the *Let-us-Merge* message on the merge edge $e(C) = (c, d)$, d realizes that D has smaller level than C;
(iii) after receiving the *Let-us-Merge* message on the merge edge $e(C) = (c, d)$, d realizes that D and C have the same level but it is not a friendly merger.
(iv) after receiving the *Let-us-Merge* message on the merge edge $e(C) = (c, d)$, d realizes that D and C have the same level but it does not know that it is a friendly merger.

In cases (i)-(iii) the activities of C are suspended and will be resolved (if the protocol is correct) only in the "future", i.e. after D changes level. Case (iv) is different from the others, as it will be resolved within the "present" (i.e., in this level); it is in fact a *delay* rather than a *suspension.*

Observe that, if there is no suspension, there is no problem.

Property 8.1 *If a city at level l will* not *be suspended, its level will eventually increase, unless it is the megacity.*

To see why this is true, consider the operations performed by a city C at a level l: compute the merge edge, and send a merge request on the merge edge. If it is not suspended, its merge request arrives at a city D with either a larger level (in which case, C is absorbed and its level becomes $level(D)$), or the same level and same merge-edge (in which case, the two cities friendly merge and their level increases).

So, only suspensions can create problems. But not necessarily so:

Property 8.2 *Let city C at level l be suspended by a district d in city D. If the level of the city of D becomes greater than l, C will no longer be suspended and its level will increase.*

This is because, once the level of D becomes greater than the level of C, d can answer the *Outside?* message in case (i), as well as the *Let-us-Merge* message in cases (ii) and (iii).

Thus, the only real problem is the presence of a city suspended by another whose level will not grow. We are going now to see that this can not occur.

Consider the *smallest* level l of any city at time t, and concentrate on the cities $\mathcal{C}$ operating at that level at that time.

Property 8.3 *No city in $\mathcal{C}$ will be suspended by a city at higher level.*

This is because, for a suspension to exist, the level of D can *not* be greater that the level of C (see the cases above).

Thus, if a city $C \in \mathcal{C}$ is suspended, it is for some other city $C' \in \mathcal{C}$. If C' will not be suspended at level l, its level will increase; when that happens, C will no longer be suspended. In other words, there will be no problems as long as there are no cycles of suspensions within $\mathcal{C}$; that is, as long as there is no cycle $C_0, C_1, \ldots, C_{k-1}$ of cities of $\mathcal{C}$ where C_i is suspended by C_{i+1} (and the operation on the indices are modulo k). The crucial property is the following:

Property 8.4 *There will be no cycles of suspensions within $\mathcal{C}$.*

The proof of this property is based heavily on the fact that each edge has a unique length (we have assumed that !), and that the merge edge $e(C)$ chosen by C is the shortest of all the unused links incident on C. Remember this fact, and let us proceed with the proof.

By contradiction, assume that the property is false. That is, assume there is a cycle $C_0, C_1, \ldots, C_{k-1}$ of cities of $\mathcal{C}$ where C_i is suspended by C_{i+1} (the operation on the indices are modulo k). First of all observe that since all these cities are at the same level, then the reason they are suspended can only be that each is involved in an "unfriendly" merger, i.e., case (iii). Let us examine the situation more closely: each C_i has chosen a merge edge $e(C_i)$ connecting it to C_{i+1}; thus C_i is suspending C_{i-1} and is suspended by C_{i+1}. Clearly both $e(C_{i-1})$ and $e(C_i)$ are incident on C_i. By definition of merging edge (recall what we said at the beginning of the proof), $e(C_i)$ is shorter than $e(C_{i-1})$ (otherwise C_i would have chosen it instead); in other words, the length d_i of the road $e(C_i)$ is smaller than the length d_{i1} of $e(C_{i+1})$. This means that $d_0 > d_1 > \ldots > d_{k-1}$; but since it is a circle of suspensions, C_{k-1} is suspended by C_0, that is $d_{k-1} > d_0$. We have reached a contradiction, which implies that our assumption that the property does not hold is actually false; thus the property is true.

As a consequence of the property, all cities in $\mathcal{C}$ will eventually increase their level: first the ones involved in a friendly merger, next those that had chosen them for a merger (and thus absorbed by them), then those suspended by the latter, and so on.

This implies that, *at no time there will be deadlock* and *there is always progress*: use the properties to show that the ones with smallest level will increase their value; when this happens, again the ones with smallest level will increase it, and so on. That is

Property 8.5 *Protocol* Mega-Merger *is deadlock-free and ensures progress.*

Termination

We have just seen that there will be no deadlock, and that progress is guaranteed. This means that the cities will keep on merging, and eventually the megacity will be formed. The problem is how to detect that this has happened. Recall that no node has knowledge of the network, not even of its size (it is not part of the standard set of assumptions for election); how does an entity finds out that all the nodes are now part of the same city ? Clearly it is sufficient for just one entity to determine termination (as it can then broadcast it to all the others).

total number or the publicly serviced roads is exactly $n-1$. Thus the total number of the other roads is exactly $m-(n-1)$. This means that the total number of useless messages will be

Property 8.10 $Useless = 2(m-n+1)$

The total
Combining Properties 8.8, 8.9, and 8.10, we obtain the total number of messages exchanged in total by protocol *Mega-Merger* during all its levels of execution. To these, we need to add the $n-1$ messages due to the downtown of the megacity broadcasting termination (eventhough these could be saved: Exercise 9.82), for a total of

$$\mathbf{M}[Mega-Merger] \leq 2m + 4n\log n + 1 \tag{41}$$

8.1.6 Road Lengths and Minimum-Cost Spanning Trees

In all the previous discussions we have made some non-standard assumptions about the edges. We have in fact assumed that each link has a value, which we called length, and that those values are unique. (This last assumption provides for free unique names to the links, as the unique length can be used as the link's name.)

The existence of link values is not uncommon. In fact, dealing with networks, usually there is a value associated with a link denoting e.g., the cost of using that link, the transmission delays incurred when sending a message through it, etc.

In these situations, when constructing a spanning-tree (e.g. to use for broadcasting), the prime concern is how to construct the one of *minimum cost*, i.e., where the sum of the values of its link is as small as possible. For example, if the value of the link is the cost of using it, a minimum-cost spanning tree is one where broadcasting would be cheapest (regardless of who is the originator of the broadcast). Not surprisingly, the problem of constructing a minimum-cost spanning tree is important and heavily investigated.

We have seen that protocol *Mega-Merger* constructs a rooted spanning tree of the network. What we are going to see now is that this tree is actually the unique minimum-cost spanning-tree of the network. We are also going see how the non-standard assumptions we have made about the existence of unique lengths can be easily removed.

Minimum-Cost Spanning Trees
In general, a network can have several minimum cost spanning trees. For example, if all links have the same value (or have no value), then every spanning tree is minimal. On the other hand,

Property 8.11 *If the link values are distinct, a network has a unique minimum-cost spanning tree.*