Chapter 3

Leader Election

Some algorithms (e.g. the slow tree coloring Algorithm 1.14) ask for a special node, a so-called "leader". Computing a leader is a very simple form of symmetry breaking. Algorithms based on leaders do generally not exhibit a high degree of parallelism, and therefore often suffer from poor time complexity. However, sometimes it is still useful to have a leader to make critical decisions in an easy (though non-distributed!) way.

3.1 Anonymous Leader Election

The process of choosing a leader is known as *leader election*. Although leader election is a simple form of symmetry breaking, there are some remarkable issues that allow us to introduce notable computational models.

In this chapter we concentrate on the ring topology. Many interesting challenges in distributed computing already reveal the root of the problem in the special case of the ring. Paying attention to the ring also makes sense from a practical point of view as some real world systems are based on a ring topology, e.g., the antiquated token ring standard.

Problem 3.1 (Leader Election)**.** *Each node eventually decides whether it is a leader or not, subject to the constraint that there is exactly one leader.*

Remarks:

- More formally, nodes are in one of three states: undecided, leader, not leader. Initially every node is in the undecided state. When leaving the undecided state, a node goes into a final state (leader or not leader).

Definition 3.2 (Anonymous)**.** *A system is anonymous if nodes do not have unique identifiers.*

Definition 3.3 (Uniform)**.** *An algorithm is called uniform if the number of nodes n is not known to the algorithm (to the nodes, if you wish). If n is known, the algorithm is called non-uniform.*

Whether a leader can be elected in an anonymous system depends on whether the network is symmetric (ring, complete graph, complete bipartite graph, etc.)

or asymmetric (star, single node with highest degree, etc.). We will now show that non-uniform anonymous leader election for synchronous rings is impossible. The idea is that in a ring, symmetry can always be maintained.

Lemma 3.4. *After round k of any deterministic algorithm on an anonymous ring, each node is in the same state s_k.*

Proof by induction: All nodes start in the same state. A round in a synchronous algorithm consists of the three steps sending, receiving, local computation (see Definition 1.8). All nodes send the same message(s), receive the same message(s), do the same local computation, and therefore end up in the same state.

Theorem 3.5 (Anonymous Leader Election)**.** *Deterministic leader election in an anonymous ring is impossible.*

Proof (with Lemma 3.4): If one node ever decides to become a leader (or a non-leader), then every other node does so as well, contradicting the problem specification 3.1 for $n > 1$. This holds for non-uniform algorithms, and therefore also for uniform algorithms. Furthermore, it holds for synchronous algorithms, and therefore also for asynchronous algorithms.

Remarks:

- Sense of direction is the ability of nodes to distinguish neighbor nodes in an anonymous setting. In a ring, for example, a node can distinguish the clockwise and the counterclockwise neighbor. Sense of direction does not help in anonymous leader election.
- Theorem 3.5 also holds for other symmetric network topologies (e.g., complete graphs, complete bipartite graphs, ...).
- Note that Theorem 3.5 does generally not hold for randomized algorithms; if nodes are allowed to toss a coin, some symmetries can be broken.
- However, more surprisingly, randomization does not always help. A randomized uniform anonymous algorithm can for instance not elect a leader in a ring. Randomization does not help to decide whether the ring has $n = 3$ or $n = 6$ nodes: Every third node may generate the same random bits, and as a result the nodes cannot distinguish the two cases. However, an approximation of n which is strictly better than a factor 2 will help.

3.2 Asynchronous Ring

We first concentrate on the asynchronous model from Definition 2.7. Throughout this section we assume non-anonymity; each node has a unique identifier. Having IDs seems to lead to a trivial leader election algorithm, as we can simply elect the node with, e.g., the highest ID.

Theorem 3.7. *Algorithm 3.6 is correct. The time complexity is $\mathcal{O}(n)$. The message complexity is $\mathcal{O}(n^2)$.*

Algorithm 3.6 Clockwise Leader Election

```
1: Each node v executes the following code:
2: v sends a message with its identifier (for simplicity also v) to its clockwise
   neighbor.
3: v sets m := v {the largest identifier seen so far}
4: if v receives a message w with w > m then
5:     v forwards message w to its clockwise neighbor and sets m := w
6:     v decides not to be the leader, if it has not done so already.
7: else if v receives its own identifier v then
8:     v decides to be the leader
9: end if
```

Proof: Let node z be the node with the maximum identifier. Node z sends its identifier in clockwise direction, and since no other node can swallow it, eventually a message will arrive at z containing it. Then z declares itself to be the leader. Every other node will declare non-leader at the latest when forwarding message z. Since there are n identifiers in the system, each node will at most forward n messages, giving a message complexity of at most n^2. We start measuring the time when the first node that "wakes up" sends its identifier. For asynchronous time complexity (Definition 2.8) we assume that each message takes at most one time unit to arrive at its destination. After at most $n-1$ time units the message therefore arrives at node z, waking z up. Routing the message z around the ring takes at most n time units. Therefore node z decides no later than at time $2n-1$. Every other node decides before node z.

Remarks:

- Note that in Algorithm 3.6 nodes distinguish between clockwise and counterclockwise neighbors. This is not necessary: It is okay to simply send your own identifier to any neighbor, and forward a message to the neighbor you did not receive the message from. So nodes only need to be able to distinguish their two neighbors.
- Careful analysis shows, that while having worst-case message complexity of $\mathcal{O}(n^2)$, Algorithm 3.6 has an *average* message complexity of $\mathcal{O}(n \log n)$. Can we improve this algorithm?

Theorem 3.9. *Algorithm 3.8 is correct. The time complexity is $\mathcal{O}(n)$. The message complexity is $\mathcal{O}(n \log n)$.*

Proof: Correctness is as in Theorem 3.7. The time complexity is $\mathcal{O}(n)$ since the node with maximum identifier z sends messages with round-trip times $2, 4, 8, 16, \ldots, 2 \cdot 2^k$ with $k \leq \log(n+1)$. (Even if we include the additional wake-up overhead, the time complexity stays linear.) Proving the message complexity is slightly harder: if a node v manages to survive round r, no other node in distance 2^r (or less) survives round r. That is, node v is the only node in its 2^r-neighborhood that remains active in round $r+1$. Since this is the same for every node, less than $n/2^r$ nodes are active in round $r+1$. Being active in round r costs $2 \cdot 2 \cdot 2^r$ messages. Therefore, round r costs at most $2 \cdot 2 \cdot 2^r \cdot \frac{n}{2^{r-1}} = 8n$

Algorithm 3.8 Radius Growth

1: **Each node** v does the following:
2: Initially all nodes are *active*. {all nodes may still become leaders}
3: Whenever a node v sees a message w with $w > v$, then v decides to not be a leader and becomes *passive*.
4: Active nodes search in an exponentially growing neighborhood (clockwise and counterclockwise) for nodes with higher identifiers, by sending out *probe* messages. A probe message includes the ID of the original sender, a bit whether the sender can still become a leader, and a time-to-live number (TTL). The first probe message sent by node v includes a TTL of 1.
5: Nodes (active or passive) receiving a probe message decrement the TTL and forward the message to the next neighbor; if their ID is larger than the one in the message, they set the leader bit to zero, as the probing node does not have the maximum ID. If the TTL is zero, probe messages are returned to the sender using a *reply* message. The reply message contains the ID of the receiver (the original sender of the probe message) and the leader-bit. Reply messages are forwarded by all nodes until they reach the receiver.
6: Upon receiving the reply message: If there was no node with higher ID in the search area (indicated by the bit in the reply message), the TTL is doubled and two new probe messages are sent (again to the two neighbors). If there was a better candidate in the search area, then the node becomes passive.
7: If a node v receives its own probe message (not a reply) v decides to be the leader.

messages. Since there are only logarithmic many possible rounds, the message complexity follows immediately.

Remarks:

- This algorithm is asynchronous and uniform as well.
- The question may arise whether one can design an algorithm with an even lower message complexity. We answer this question in the next section.

3.3 Lower Bounds

Lower bounds in distributed computing are often easier than in the standard centralized (random access machine, RAM) model because one can argue about messages that need to be exchanged. In this section we present a first difficult lower bound. We show that Algorithm 3.8 is asymptotically optimal.

Definition 3.10 (Execution)**.** *An execution of a distributed algorithm is a list of events, sorted by time. An event is a record (time, node, type, message), where type is "send" or "receive".*

Remarks:

- We assume throughout this course that no two events happen at exactly the same time (or one can break ties arbitrarily).
- An execution of an asynchronous algorithm is generally not only determined by the algorithm but also by a "god-like" scheduler. If more than one message is in transit, the scheduler can choose which one arrives first.
- If two messages are transmitted over the same directed edge, then it is sometimes required that the message first transmitted will also be received first ("FIFO").

For our lower bound, we assume the following model:

- We are given an asynchronous ring, where nodes may wake up at arbitrary times (but at the latest when receiving the first message).
- We only accept uniform algorithms where the node with the maximum identifier can be the leader. Additionally, every node that is not the leader must know the identity of the leader. These two requirements can be dropped when using a more complicated proof; however, this is beyond the scope of this course.
- During the proof we will "play god" and specify which message in transmission arrives next in the execution. We respect the FIFO conditions for links.

Definition 3.11 (Open Schedule)**.** *A schedule is an execution chosen by the scheduler. An open (undirected) edge is an edge where no message traversing the edge has been received so far. A schedule for a ring is open if there is an open edge in the ring.*

The proof of the lower bound is by induction. First we show the base case:

Lemma 3.12. *Given a ring R with two nodes, we can construct an open schedule in which at least one message is received. The nodes cannot distinguish this schedule from one on a larger ring with all other nodes being where the open edge is.*

Proof: Let the two nodes be u and v with $u < v$. Node u must learn the identity of node v, thus receive at least one message. We stop the execution of the algorithm as soon as the first message is received. (If the first message is received by v, bad luck for the algorithm!) Then the other edge in the ring (on which the received message was not transmitted) is open. Since the algorithm needs to be uniform, maybe the open edge is not really an edge at all, nobody can tell. We could use this to glue two rings together, by breaking up this imaginary open edge and connect two rings by two edges. An example can be seen in Figure 3.13.

Lemma 3.14. *By gluing together two rings of size $n/2$ for which we have open schedules, we can construct an open schedule on a ring of size n. If $M(n/2)$ denotes the number of messages already received in each of these schedules, at least $2M(n/2) + n/4$ messages have to be exchanged in order to solve leader election.*

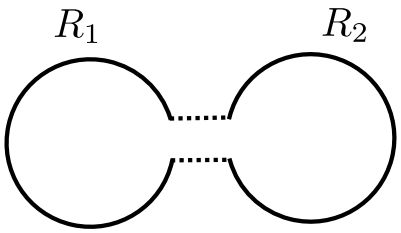

Figure 3.13: The rings R_1, R_2 are glued together at their open edge.

We divide the ring into two sub-rings R_1 and R_2 of size $n/2$. These subrings cannot be distinguished from rings with $n/2$ nodes if no messages are received from "outsiders". We can ensure this by not scheduling such messages until we want to. Note that executing both given open schedules on R_1 and R_2 "in parallel" is possible because we control not only the scheduling of the messages, but also when nodes wake up. By doing so, we make sure that $2M(n/2)$ messages are sent before the nodes in R_1 and R_2 learn anything of each other!

Without loss of generality, R_1 contains the maximum identifier. Hence, each node in R_2 must learn the identity of the maximum identifier, thus at least $n/2$ additional messages must be received. The only problem is that we cannot connect the two sub-rings with both edges since the new ring needs to remain open. Thus, only messages over one of the edges can be received. We look into the future: we check what happens when we close only one of these connecting edges.

Since we know that $n/2$ nodes have to be informed in R_2, there must be at least $n/2$ messages that must be received. Closing both edges must inform $n/2$ nodes, thus for one of the two edges there must be a node in distance $n/4$ which will be informed upon creating that edge. This results in $n/4$ additional messages. Thus, we pick this edge and leave the other one open which yields the claim.

Lemma 3.15. *Any uniform leader election algorithm for asynchronous rings has at least message complexity* $M(n) \geq \frac{n}{4}(\log n + 1)$.

Proof by induction: For the sake of simplicity we assume n being a power of 2. The base case $n = 2$ works because of Lemma 3.12 which implies that $M(2) \geq 1 = \frac{2}{4}(\log 2 + 1)$. For the induction step, using Lemma 3.14 and the induction hypothesis we have

$$\begin{aligned} M(n) &= 2 \cdot M\left(\frac{n}{2}\right) + \frac{n}{4} \\ &\geq 2 \cdot \left(\frac{n}{8}\left(\log \frac{n}{2} + 1\right)\right) + \frac{n}{4} \\ &= \frac{n}{4}\log n + \frac{n}{4} = \frac{n}{4}\left(\log n + 1\right). \end{aligned}$$

□

Remarks:

- To hide the ugly constants we use the "big Omega" notation, the lower bound equivalent of $\mathcal{O}()$. A function f is in $\Omega(g)$ if there are constants x_0 and $c > 0$ such that $|f(x)| \geq c|g(x)|$ for all $x \geq x_0$.

- In addition to the already presented parts of the "big O" notation, there are 3 additional ones. Remember that a function f is in $\mathcal{O}(g)$ if f grows at most as fast as g. A function f is in $o(g)$ if f grows slower than g.
- An analogous small letter notation exists for Ω. A function f is in $\omega(g)$ if f grows faster than g.
- Last but not least, we say that a function f is in $\Theta(g)$ if f grows as fast as g, i.e., $f \in \mathcal{O}(g)$ and $f \in \Omega(g)$.
- Again, we refer to standard text books for formal definitions.

Theorem 3.16 (Asynchronous Leader Election Lower Bound)**.** *Any uniform leader election algorithm for asynchronous rings has $\Omega(n \log n)$ message complexity.*

3.4 Synchronous Ring

The lower bound relied on delaying messages for a very long time. Since this is impossible in the synchronous model, we might get a better message complexity in this case. The basic idea is very simple: In the synchronous model, *not* receiving a message is information as well! First we make some additional assumptions:

- We assume that the algorithm is non-uniform (i.e., the ring size n is known).
- We assume that every node starts at the same time.
- The node with the minimum identifier becomes the leader; identifiers are integers.

Algorithm 3.17 Synchronous Leader Election

1: **Each node** v concurrently executes the following code:
2: The algorithm operates in synchronous phases. Each phase consists of n time steps. Node v counts phases, starting with 0.
3: **if** phase $= v$ **and** v did not yet receive a message **then**
4: v decides to be the leader
5: v sends the message "v is leader" around the ring
6: **end if**

Remarks:

- Message complexity is indeed n.
- But the time complexity is huge! If m is the minimum identifier it is $m \cdot n$.
- The synchronous start and the non-uniformity assumptions can be dropped by using a wake-up technique (upon receiving a wake-up message, wake up your clockwise neighbors) and by letting messages travel slowly.

- There are several lower bounds for the synchronous model: comparison-based algorithms or algorithms where the time complexity cannot be a function of the identifiers have message complexity $\Omega(n \log n)$ as well.
- In general graphs, efficient leader election may be tricky. While time-optimal leader election can be done by parallel flooding-echo (see Chapter 2), bounding the message complexity is more difficult.

Chapter Notes

[Ang80] was the first to mention the now well-known impossibility result for anonymous rings and other networks, even when using randomization. The first algorithm for asynchronous rings was presented in [Lan77], which was improved to the presented clockwise algorithm in [CR79]. Later, [HS80] found the radius growth algorithm, which decreased the worst case message complexity. Algorithms for the unidirectional case with runtime $\mathcal{O}(n \log n)$ can be found in [DKR82, Pet82]. The $\Omega(n \log n)$ message complexity lower bound for comparison based algorithms was first published in [FL87]. In [Sch89] an algorithm with constant error probability for anonymous networks is presented. General results about limitations of computer power in synchronous rings are in [ASW88, AS88].

Bibliography

[Ang80] Dana Angluin. Local and global properties in networks of processors (Extended Abstract). In *12th ACM Symposium on Theory of Computing (STOC)*, 1980.

[AS88] Hagit Attiya and Marc Snir. Better Computing on the Anonymous Ring. In *Aegean Workshop on Computing (AWOC)*, 1988.

[ASW88] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. volume 35, pages 845–875, 1988.

[CR79] Ernest Chang and Rosemary Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, May 1979.

[DKR82] Danny Dolev, Maria M. Klawe, and Michael Rodeh. An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle. *J. Algorithms*, 3(3):245–260, 1982.

[FL87] Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, 1987.

[HS80] D. S. Hirschberg and J. B. Sinclair. Decentralized extrema-finding in circular configurations of processors. *Commun. ACM*, 23(11):627–628, November 1980.

[Lan77] Gérard Le Lann. Distributed Systems - Towards a Formal Approach. In *International Federation for Information Processing (IFIP) Congress*, 1977.

[Pet82] Gary L. Peterson. An $O(n \log n)$ Unidirectional Algorithm for the Circular Extrema Problem. 4(4):758–762, 1982.

[Sch89] B. Schieber. Calling names on nameless networks. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, PODC '89, pages 319–328, New York, NY, USA, 1989. ACM.