Leader election algorithm for modular robots
Master 2 IoT

Abdallah Makhoul[1]

Work to do

- Read and understand the text and the algorithms.
- Implement the distributed algorithms of the leader election in a modular robot system with the visibleSim simulator.
- Realize simulations of the algorithm for a modular robot composed of 199 modules and for 3 different topologies: Chains, Full Grid, and Random.
- For each execution, you should evaluate the execution time and the number of messages exchanged in the network.
- Write a report with all the results and send it to me by email by December 20th.

1. Background study

This section will be dedicated to basic definitions and terminology in modular robots. A modular robotic system is a system that is presented as an undirected graph where the vertices are the modules (particles) and the edges are the communication links between neighbouring particles. We consider that all the particles are uniform. They are anonymous and indistinguishable they have the same clockwise orientation and execute the same program. The communication among the modules happens through exchanging messages. Each

Email address: abdallah.makhoul@univ-fcomte.fr (Abdallah Makhoul)
[1]Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, Montbéliard, France

particle can communicate with its adjacent neighbors only. A particle's neighbor is defined as being the module that is directly connected to that particle through an interface. Modules are only aware of their status and their neighbors' status only. Next we will give some definitions and assumptions before starting the development of the algorithm.

Definition 1 (Particle/module). *A particle, also referred to as module, is defined by being a conceptual model for a computational entity in an infinite regular grid. A module can take different forms. In one system, only one type of particles is used. Multiple types of modules exist leading to different organisations, such as a regular like square lattice (Smart Blocks), hexagonal , cubic lattice (Blinky Blocks), FCC lattice (3D Catom), free positions (Kilobots) [**?**], and droplet [**?**]. It is important to note that modules have a constant memory size.*

Definition 2 (Weight). *The weight of a robot is defined as an intrinsic value calculated from the list of its connection to neighbors. This concept is explained in details in subsection 2.1*

Definition 3 (Symmetrical system). *Axial symmetry is usually defined as the geometrical quality of being made up of exactly similar parts facing each other around an axis. In addition, in our approach, the weight of the modules was taken into consideration as an added condition to accomplish, a so called, symmetrical system.*

In modular robots, many algorithms need a leader even though it is just to start a distributed algorithm, the distributed leader election algorithm is therefore very important.

Leader election is a fundamental problem in modular robots systems. In this context, leader election is essential and important for various and different applications such as shape formation, self-repair, self-assembly algorithms, decision making, etc. Leader election is a fairly old problem in distributed systems, it consists in electing a unique leader from a fixed set of node.

2. Leader election for modular robots

This section will be describing the proposed solution in details. The algorithm is based in six phases. First, each module generate locally a integer, called *weight* based on the numbering of their connected interfaces. After that, spanning trees will be created using a recruitment method, then the values of the recruited modules will be calculated, the generated spanning tree will compete and the loosing trees are dismantled, this step is repeated until one tree remains. And finally the leader is elected.

2.1. Computing the weight parameter

Considering that each robot is connected to up to d neighbors, (d is the degree of the associated graph), and defining C_i the state of the ith connector (setting $C_i = 1$ if connected and $C_i = 0$ elsewise) we encode the weight concatenating the d bits. Then using the connected interfaces and its orientation, each module will be generating its *weight* w as follows:

$$w = \sum_{i=0}^{d} 2^{C_i} \tag{1}$$

Where C_i is the state of the connector i (1 if connected 0 otherwise), d is the degree of the graph (the number of connectors).

Some examples of the number generation are presented in Figure 1 with different kind of robots. Both position and orientation of the module affect its *weight*. For example, if a module in a square lattice has the interfaces 0 and 3 connected, the respective binary number is $(1001)_{bin}$ which gives $(9)_{dec}$ in decimal.

By the end of this step all the modules will have generated an integer as previously described. This integer will be used in the third phase of the algorithm to be able to compute the tree value. The algorithm part concerning the integer number generation is presented in Algorithm 1 (cf. Appendix A).

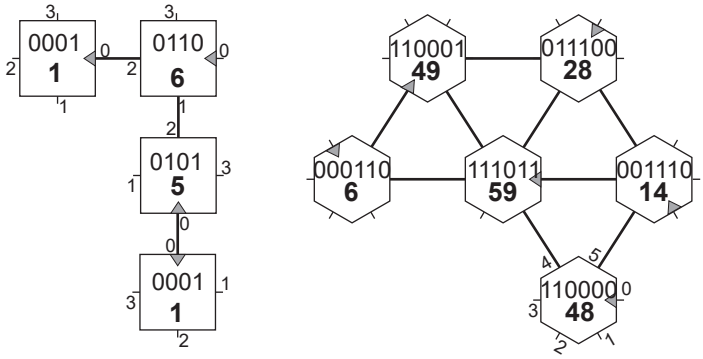

Figure 1: Examples of weight generation based on the interfaces connections for two different 2D lattices (cubic on the left and hexagonal on the right). Grey triangles mark #0 connectors, others are placed clockwise.

2.2. Recruitment phase

In this section, prospective leaders will start competing by recruiting modules to create their spanning tree. In this scenario, prospective leaders are chosen as follow: if the module has only one connection, it directly becomes a prospective leader. If a module has two connections, it will wait for a T_2 time and after that time if it was not recruited it will become a prospective leader and start the recruitment process. Likewise, modules with 3 connections will wait for T_3 time and so on. The delay T_i, where i is the number of connected interfaces, can be fixed by the user before the execution of the algorithm. It can be computed in function of the diameter of the network and the speed of the message propagation.

When an available module receives a recruitment message, it marks the interface from where the message was received as parent interface, changes the module flag to recruited, informs its parent that the recruitment was successful and starts recruiting its remaining connected modules. This process will continue until all modules in the system are recruited. When a module has no neighbor to recruit or all its neighbors are already recruited, it will be depicted as a leaf of the tree since no more children can be recruited. After that, each spanning tree will have two properties, the total number of modules recruited

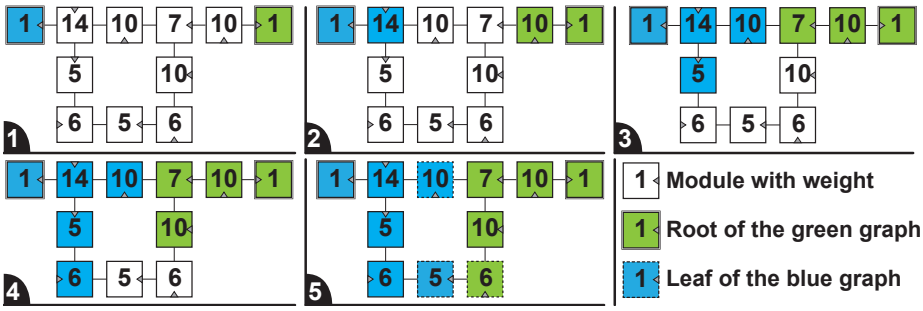

Figure 2: 5 steps of the recruitment process: Roots start recruiting modules until no more modules are available.

in the tree and the sum of the integer values (calculated in phase 1) of the modules in the same tree. Algorithms 2 and 3 (cf. Appendix A) give more details what happens when a module receives a recruit message and when an already recruited module receives a recruit message respectively.

An illustration example of the recruitment phase is presented in Figure 2. In this example we have two prospective leaders that start recruiting modules until no more modules are available.

2.3. Tree value computation

Once a module appoints itself as leaf, it sends its generated integer number to its parent. And every time a parent receives such a value, it will increment the counter that counts the number of modules, and it will flag the interface from where the value was received as value sent. Once all the interfaces that are connected to its children sent their values, the module computes the sum of all the received values and sends it to its parent. This process will keep on executing until it reaches the tree root. In like manner, the root itself will process the message as previously described and, as a result, the final value of that tree and the total number of modules are calculated. Forthwith, the root will then broadcast the tree value and the number of modules in its tree to all its children. As soon as a module receives this type of message, it will save those variables in its local memory and sets its "ready to compete" flag to true.

Identically, this module will forward the same message to its children, that will treat this message in the same fashion, until it reaches the leaves. As previously described, the competition among the trees is done through the connected leaves or modules. Therefore, the tree root should send the total sum of the tree to all its children in order to compete with other trees. Algorithm 4 shows how the sum message is being handled by the modules and Algorithm 5, explains the steps done by the modules to broadcast the sum of the tree to all its children (cf Appendix A).

Figure 3 shows an illustrative example of this phase. Starting from the leaf, each module sends its value to its parent. The parent adds it up to its own and - after having received all values from its children - sends it backward until the message reaches the root of the tree. Once the sum reaches the root, it will be added to the root's value and then the root broadcast it to all the modules in the same tree.

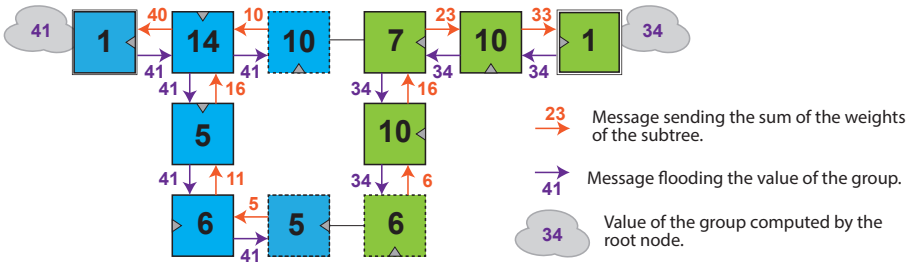

Figure 3: Group value computation: Starting from the leaf of each group, each leaf sends its weight to its parent. The parent adds the received value with its weight and then send the result backward until it reaches the root of the tree. Once the sum reaches the root, it will be added to the root's weight to define the group value. This group value is then broadcast to all the modules of the tree.

2.4. Competition with other trees

In this section, all the modules are ready to compete. To put it in other words, all modules received the total number of particles in their tree and the sum of all its integers, and in addition, they have their "ready to compete" flag set to true. Modules that have neighbors recruited to a different tree, send

them a competing message containing the sum of the tree and the number of modules. The receiving party then compares those variables with the one stored in their memory. The module with the higher sum wins. If both trees have the same value, then the tree with more modules is the winner. There are rare and negligible cases where both sum and total number of modules are the same for the competing trees, this is called conflict and is discussed in details in section **??** of this paper. Algorithms 6, 7 and 8 show in more details the competition phase (cf. Appendix A).

2.5. Dismantling trees and more recruitment

As a result of the previous competitions, loosing trees will become dismantled. In this scenario, dismantle is defined as breaking into pieces. This means that the modules that were the children of the loosing tree will no longer be related to that tree and they will be restored to their state before the recruitment. The process operates as follow: the leaf that received the loosing message, will send that message back to the tree root. In his turn, the root broadcasts to all its children the dismantle message. Consequently, the particles receiving the dismantle message set the recruited flag to "false", the interfaces that are flagged as parent and children are no longer flagged. Furthermore, the variables containing the value of the tree and the number of modules are set back to zero. Once the dismantle process reaches the leaf of the loosing tree, it will send a message to the winning tree that the dismantle is finished. As a result, the winning tree will re-initiate the recruitment and start recruiting the dismantled modules in the same manner. Algorithm 9 explains the dismantling process in details and Algorithm 10 shows how to reset the sum of the tree (cf. Appendix A).

Figure 4 presents an example of the dismantling recruitment. In this example the loosing modules send a dismantle message to their parent until it reaches the root of the loosing tree. In the mean time, the winning tree sends a message to its parent until it reaches the root to reset the value of the tree. Furthermore, the winning modules start recruiting the loosing tree.

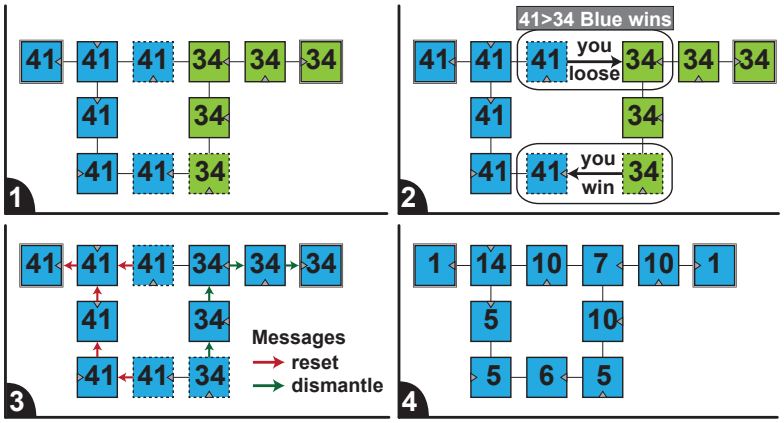

Figure 4: 4 steps of the dismantling recruitment: First step presents the initial state, step 2 shows the competition between the two groups, the blue group win on the green group, then each leaf of the two groups produces reset and dismantle messages. Step 3 shows the broadcasting of reset and dismantle messages. At the end, step 4, their is only one group and nodes are reset.

2.6. Leader election phase

As long as the system contains multiple trees, the competition will go on and recruitment will be re-initiated until one tree remains. This occurs when all the modules of the tree are not connected to any other tree. The root of this tree will then be elected as leader (cf.Appendix A, Algorithm 11).

After the election, the leader will change its leader flag and leader found flag to true, broadcast a leader found message to all its neighbors, and restore to its initial state before the beginning of the algorithm except for the leader flag and the leader found flag that should remain set to true. Correspondingly, every module that receives the leader found message, will restore to its initial state and then sets the leader found flag to true, (cf.Appendix A, Algorithm 12).

Figure 5 presents an example of the last phase of the leader election approach. Once a module can not recruit or compete further, it will send a "potential leader" message to its parent. Each parent that receives this type of message, will wait until all its children send him the same message then send it backward to its parent. So far until the message reaches the root. If this message reaches

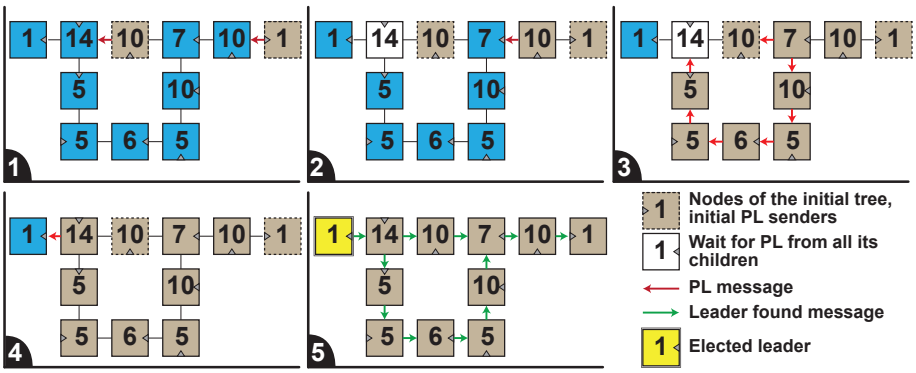

Figure 5: Leader election: 1) Once a module can not recruit or compete further, it sends a Potential Leader message (PL) to its parent. 2)& 3) Each parent that receives PL message waits for all its children PL message and then sends it backward to its parent. 4) So far until the message reaches the root, which becomes the leader and broadcast (5) to all modules in the system that the leader is elected.

the root, the root becomes the leader and broadcasts to all modules in the system that it is the leader.

Appendix A. Algorithms and Diagrams

This appendix presents the detailed algorithms of the different phases of our proposed approach for leader election in modular robots systems. Our technique is based on the interfaces connections of the modules. In the algorithms below we use the interfaces status as shown in Table A.1. InterfaceStatus is an array of type integer that contains the status of the interfaces. Table A.1 shows the different status the interfaces can take during the execution of the algorithm. At the end of this appendix, we present a complete and detailed diagram of the proposed leader election process (cf. Figure **??**).

Status	Definition
0	interface not connected
1	interface connected
2	interface connected to a neighbor that is not in its tree
3	interface connected to its parent
4	this interface received a sum message
5	this interface sent a potential leader message

Table A.1: Interface status

In Algorithm 1 we present the integer number generation based on the connected interfaces and the orientation. The integer generated is referred to as BChoice, NumbOfInterfaces is the number of interface each module has. The number of interfaces in this scenario are four for the 2D simulation and 6 for the 3D simulation.

Algorithm 1: Choice generation and interface set up

Input: NumbOfInterfaces

Output: BChoice, interfaceStatus

Data: Testing Connected Interfaces

for ($i = 0$ *to NumbOfInterfaces*) {

  if *getInterface[i] is connected* **then**

    interfaceStatus[i] = 1

    totalConnectedInt++

    bchoice = bchoice + 2^i

  end

  else if *getInterface[i] is NOT connected* **then**

    interfaceStatus[i] = 0

  end

}

Algorithm 2: Received recruit message

```
if received message is a recruit message then
    if Not recruited and not root then
        recruited = true
        parent = sender
        interfaceStatus[sender] = 3
        if all interfaces != 1 then
            if i am leaf then
                total = choice
                Send sum Msg to parent
            end
        end
        else
            Send recruit Msg through the connected interfaces except
              parent interface
        end
    end
    else
        Reply back with already recruited
    end
end
```

Algorithm 3: Received recruited message

```
if received message is a recruited message then
    interfaceStatus[sender] = 2
    if all interfaces != 1 then
        i am leaf = true
        send sum message to parent
        if I don't have neighbors from another tree then
            send potential leader message to parent
        end
    end
end
```

Algorithm 4: Received sum message

```
if received message is a sum message then
    interfaceStatus[sender] = 4
    tmpTotal += rcvdMsg
    if all interfaces != 1 then
        tmpTotal += BChoice
        totalNode + 1
        if i am root then
            treeValue = tmpTotal
            Send update tree value Msg to all interfaces with status 1
             and 4
        end
        else
            Send Sum Msg to parent
        end
    end
end
```

Algorithm 5: Updating tree value

```
if received message is an update tree value message then
    MyTreeValue = rcvdMsg
    readyToCompete = true
    send update tree value Msg through interfaces with status 1 and 4
    Send choice Msg through interface with status 2  // to start the
     competition with other trees
end
```

Algorithm 6: Receiving choice message

```
if received message is a choice message then
    if ready to compete then
        if Msg < MyTreeValue then
            I am leaf = false
            Send lost message to sender
        end
        else if Msg > MyTreeValue then
            Set all interfaces with status 3 and 4 to 1
            Send win message to sender
            Send dismantle message to parent
        end
    end
end
```

Algorithm 7: Receiving win message

```
if received message is a win message then
    if recvdMsg = 1 then
        Send Reset Sum Msg to parent
    end
    else
        if ReceivedMsg=0 and I am not leaf then
            Set all interfaces with status 2 to 1
            Re-initiate recruitment
        end
    end
end
```

Algorithm 8: Receiving lost message

```
if received message is a lost message then
    iAmLeaf=false
    Dismantle=true
    Set all interfaces withs status 3 and 4 to 1
    Send dismantle Msg to parent
end
```

Algorithm 9: Dismantling tree

```
if received message is a dismantle message then
    if Node Not reset then
        if Not root and rcvdmsg = 0 and dismantle not sent then
            Send dismantle Msg to parent;
        end
        else
            if I am root then
                iAmRoot=false
                nodeReset=true
                myParent=NULL
                recruited=false
                Send dismantle Msg with value 1 through interface with
                 status 1 and 4
            end
            else
                if rcvdMsg != 1 then
                    NodeReset=true
                    Parent=NULL
                    Recruited=false
                    Send dismantle Msg with value 1 through interface
                     with status different than 0 and 2 except the sender
                     interface
                end
            end
        end
    end
end
interfaceStatus=1
iAmLeaf=false when interfaceStatus=2
Send win Msg with value 0 through interfaces with status 2
```

Algorithm 10: Reseting the sum of the tree

```
if received message is a reset sum message then
    if iAmNotRoot and rcvdMsg = 0 and resetSum not sent then
        Send reset sum message with value 0 to parent
    end
    else
        if iAmRoot and rcvdMsg = 0 and resetSum not sent then
            ResetSumSent=true
            tmpTotal=0
            MyTreeValue=0
            ReadyToCompete=false
            Set all interfaces with status 4 to 1
            Send reset sum Msg with value 1 through interfaces with
             status 4 or 1
        end
        else
            if rcvdMsg=1 then
                ResetSumSent=true
                tmpTotal=0
                MyTreeValue=0
                ReadyToCompete=false
                Set all interfaces with status 0 or 2 to 1
                Send reset sum Msg with value 1 through interfaces with
                 status 0 and 2 except the sender's interface
            end
        end
    end
end
```

Algorithm 11: Potential leader message

```
if received message is a potential leader message then
    ReceivingInterface = 5
    if All interfaces =5 then
        if i am root then
            iAmLeader=true
            LeaderFound=true
            Send Leader found Msg to all neighbors
        end
        else
            Send potential leader Msg to parent
        end
    end
end
```

Algorithm 12: Leader found message

```
if received message is a leader found message then
    leaderFound = true
    if I am not a leaf then
        Send Leader found Msg to all neighbors
    end
end
```