2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

A UNIQUE IDENTIFIER ASSIGNMENT METHOD FOR DISTRIBUTED MODULAR ROBOTS

Joseph Assaker[1], Abdallah Makhoul[1], Julien Bourgeois[1] and Jacques Demerjian[2]

Abstract— **Modular robots are autonomous systems with variable morphology, composed of independent connected computational elements, called particles or modules. Due to critical resource constraints and limited capabilities, globally unique identifier (ID) assignment to each particle is a very challenging task in modular robots. However, having a unique ID in each one remains essential for various operations and applications in this domain. For instance, it is required to establish communications between nodes and implement routing protocols. It helps in saving energy consumption and enhancing the security mechanisms. In this paper, we propose a distributed unique ID assignment method for modular robots. It is a three phases based algorithm. The first phase consists in discovering the system while building a logical tree. The second phase finds the total size of particles in the system needed for several operations in modular robots, and the third one is dedicated to the unique ID assignment. After fully optimizing the distributed algorithm, the effects of various system shapes and leader positions on the energy and time complexity are studied, while proposing fitting solutions for different requirements.**

I. INTRODUCTION

Modular robots, or modular self-reconfiguring robotic systems, are autonomous systems with variable morphology. These systems are composed of independent connected elements called modules or particles, whose connections to one another form the overall shape of the system. Beyond sensing, processing and communication capabilities, a modular robot includes actuation and motion capabilities that allow it to reconfigure its shape by rearranging connections between modules [20, 9]. A wide range of applications for modular robots reside unexplored [2], from surgical applications, to transportation applications and space exploration. The main tasks of a modular robots system are to reconfigure its shape in order to accommodate for variable conditions that need to be met in order to complete a given final goal, and the ability to read sensing values from anywhere on the system [19]. Both of these tasks rely on message transmission between modules directly connected or connected through other modules. As modular robots are essentially systems with ever-changing shapes, and thus ever-changing modules locations, it is therefore important to have some kind of identification for each module that remains constant throughout all of the modifications the system may go through.

*This work has been supported by the EIPHI Graduate School (contract "ANR-17-EURE-0002")

[1]Authors are with FEMTO-ST institute, Univ. Bourgogne Franche-Comté, CNRS,, 1 cours Leprince-Ringuet, 25200, Montbéliard, France. {first}.{last}@femto-st.fr

[2]Jacques Demerjian with LaRRIS, Faculty of Sciences, Lebanese University, Fanar, Lebanon, jacques.demerjian@ul.edu.lb

A global unique ID is naturally proposed as a solution for such a requirement. It is required for various operations and applications [5, 12, 16]. Unique IDs are used to establish communications between nodes and to implement routing protocols. It helps in saving energy consumption and enhancing the security mechanisms. Furthermore, globally unique IDs in modular robots are used to find the mapping between two configurations [14] or for self-assembly algorithms [19, 18]. To accomplish this goal one might suggest at the manufacturing level and whenever a module is manufactured to directly assign a global unique ID to it (similar to a MAC on most electronics). However, such an approach is highly inefficient and limiting, as it would impose very long IDs in order for them to be globally unique and would require many extra steps in the manufacturing process (e.g., the ID assignment process and the maintenance of unique IDs between various manufacturers). In addition, modular robots suffer from scarcity of energy, and message passing tasks play an important role in its energy consumption. As information passed between modules is relatively small in size and often aggregated, the message's header containing the origin and destination IDs presents a more weighted role in the message's size, and thus in energy consumption. Therefore, due to the large number of particles it is almost impossible to manually assign a unique ID to each particle. It is up to the particles to assign themselves each a unique ID in a distributed and memory/energy efficient manner. The first and simple technique that comes to mind is a random ID assignment [17]. However, we need very long IDs to ensure a low probability of two particles choosing the same ID which is not suitable for modular robots with low memory and energy resources. An approach is proposed in [7] which considers the k-local identifier problem. In this approach the IDs are presented as variables assigned to each particle of the network, that are different for every two modules at a distance of at most k (k hops). In this case, we do not have long ID at each particle. However, different nodes can have the same ID, and in particular in the same neighborhood. Thus, it is important to uniquely identify the next hop node during message routing. Furthermore, in this situation the matching between two configurations based on particles IDs as described in [14] becomes more complicated or even impossible to be achieved.

A more closely related problem is ID Assignment in Wireless Sensor Networks (WSN) [13, 22, 11, 10, 21, 15, 8]. We can observe that many of the issues and constraints are shared between WSN and modular robots: a large number of nodes, limited memory, processing power and energy,

etc. Nevertheless, there are significant differences as well, notably that the topology of modular robots will evolve continuously as the robot changes its morphology. The communications in modular robots are generally done only with the adjacent neighbour modules (i.e., communication through modules borders, as wireless communication is much more costly and prone to errors) [3] which reduces the impact on messages loss. Numerous works have been conducted in the sensor networks field concerning the task of ID assignment; from attribute-based ID assignments, to unique ID assignment and a variable-length ID scheme [11]. For instance, the Self-organized ID Assignment (SIDA) approach has been proposed in [11] that essentially implements a variable-length ID scheme that assigns longer IDs to the nodes the closest to the sink, with the shortest IDs being assigned to the furthest nodes in the system. While the SIDA approach can be very useful in sensor networks, once again due to the variable morphology of the modular robots systems, such an approach would not be beneficial in our use case, and in fact, could have a negative effect on the performance of the system (as modules with long IDs could end up in a very far position from the leader particle). In [13] the authors propose a work to assign unique and static-length IDs to all sensor nodes. A tree structure was built starting from the sink while temporary IDs are assigned in order to find the total number of sensor nodes. Then ID assignment responsibilities were distributed from the sink to the sensor nodes to complete the final ID assignment.

In this paper, we explore the potential for synergy between WSNs and modular robots while focusing on the unique ID assignment task. We aim to develop a distributed algorithm inspired from those proposed in the domain of sensor networks [13, 11] and adapted to modular robots. The goal is to achieve the smallest possible ID length in order to cover all modules and to reduce the number of messages exchanged between particles. Then, it aims to discover the whole system of modules while along the way building a logical tree structure rooted at the leader module and without the need of temporary IDs assignment. This reduces the time complexity of the algorithm. Furthermore, we studied the effects of the different modular robots configurations and the leader's position on the performance of the algorithm in terms of the number of messages exchanged between modules and the time complexity.

The remainder of this paper is organized as follows. In Section II, we develop the distributed algorithm that can be split into three phases: exploration phase, system size reporting phase and unique ID assignment phase. In Section III, discussions on the effect of modular robots properties (system shapes and leader position) on the time and energy complexity are presented, as well as several simulation results. Finally, Section IV concludes with a brief description of future work.

II. Unique ID Assignment Algorithm

Our distributed ID assignment algorithm could be split into three main phases: exploration phase, system size reporting phase and unique ID assignment phase. In the first phase, and after electing the leader [6, 4], the goal is to discover the whole system of modules while along the way building a tree structure rooted at the leader module. In parallel, the second phase is launched, in which the algorithm will be collecting the system size (i.e., number of modules in the system) with the final goal of reporting the total size from the whole system to the leader module. Having the system size in hand, the leader module can calculate the least amount of bits needed in order to code global unique IDs for every module in the system. In the third phase, and after building the tree structure that logically connects modules and calculating the least amount of bits needed, the final step of unique ID assignment is launched from the leader to the whole system.

A. System assumptions

Several assumptions shall be presented before starting the development of the algorithm:

- The size of the system (i.e., the number of modules) as well as the initial shape of the system are unknown.
- The leader is elected and can be any module in the system and in any position (center, border, etc.). It is elected to "lead" the process of ID assignment.
- All communications are only possible between adjacent neighbour modules; The sender module sends a message through its border n, and the receiving module receives the message from its border m. If a reply message is needed, the receiving module would reply via its border m, which would then be received by the initial sender module via its border n.
- Each module is aware of the connections it has at any given moment (i.e., it is aware which of its borders are connected to other modules, and which borders are free). In this case, we consider that the loss of any message transmitted between the borders of modules is considered to be highly improbable, and thus, not taken into consideration.
- All modules in the system are considered to be alive until the completion of the algorithm and no new modules are introduced during the execution. However we plan on relaxing the latter assumption in our future work.

B. Phase 1: Exploration

This phase's goal is to discover the whole system of modules, while building a logical tree structure rooted at the leader module along the way. The algorithm starts with the initiation from the leader module and by using 3 types of messages. Type 1, 2 and 3 messages are all that is needed to discover and create a tree structure for the system (cf. Table I). Type 1 messages represent potentially discovering new modules; type 2 messages are confirmations that a new module has been discovered and it is one of the children of the sender module; and type 3 messages reply by neglecting the fact that a module is to be discovered, and notify the sender module that the destination module is already

TABLE I

MESSAGES' ROLE DESCRIPTION

Type	Role Description
1	Explore neighbours for potential children
2	Confirm that the explored node is a child
3	Decline that the explored node is a child
4	Report the node's sub-tree size to its parent
5	Distribute the global unique IDs to children

discovered. Also, to notice that type 2 messages are the only way to expand the tree structure of the system.

The leader module starts by specifying that it is now discovered, and that it has no parent (as it is the root node in the tree), and would finally transmit type 1 messages to each of its neighbours. Whenever a module receives a type 1 message, two cases are possible. The first case, if it is the first time it receives a type 1 message it initializes its parent to be the sender module. It then replies back by a type 2 message which notifies the parent that this node is now one of its children. Finally, it proceeds by sending type 1 messages to its own neighbours (excluding the already known to be parent neighbour). The second case, if it is not the first time that it receives a type 1 message (the module is already discovered by another module) it sends back a reply message of type 3, which notifies the sender module that this module is not its child. The Algorithm's part concerning the first phase is presented in algorithm - phase 1.

Global Unique ID Assignment Algorithm - Phase 1

```
is_discovered ← false
leader ← false
parent ← −1
unique_id ← −1
neighbours ← ∅
children ← ∅

if leader = true AND is_discovered = false then
  is_discovered ← true;
  for each neighbour in neighbours do
    send type 1 message to neighbour
  end for
end if

if received message then
  switch message.type do
    case 1
      if is_discovered = true then
        send type 3 message to message.origin
      else
        is_discovered ← true
        parent ← message.origin
        neighbours ← neighbours − {message.origin}
        send type 2 message to message.origin
        for each neighbour in neighbours do
          send type 1 message to neighbour
        end for
      end if
    case 2
      neighbours ← neighbours − {message.origin}
      children ← children ∪ {< message.origin, −1 >}
    case 3
      neighbours ← neighbours − {message.origin}
end if
```

An illustration example is presented in Figure 1. In this example, and in further illustrations and results, we will utilize the use case of a square module, having 4 borders. In Figure 1, the light grey box represents the leader module, medium grey boxes represent discovered modules, and black boxes represent the absence of a module. At stage 1, all modules are yet to be discovered and the leader module is selected. In stage 2, the leader module initiates phase 1 of the algorithm by sending type 1 messages to its neighbours. In stage 3, as all modules are not yet discovered and it is the first time that they receive a type 1 message, all four neighbours reply by a type 2 message to the leader module. This informs the leader module that all four of its neighbours are its children. In stage 4, the newly discovered modules now try to discover in their turn their neighbours, via type 1 messages. In stage 5, we can observe that the module to the left of the leader won the race in discovering the new module, and receives a type 2 message confirming this fact. Whereas the module under the leader receives a type 3 message, meaning that the module is already discovered. This process keeps on repeating until all modules in the system are discovered. Modules that already received all replies from their neighbours or don't have neighbours left to discover (i.e., their only neighbour is their parent) are ready to start phase 2.

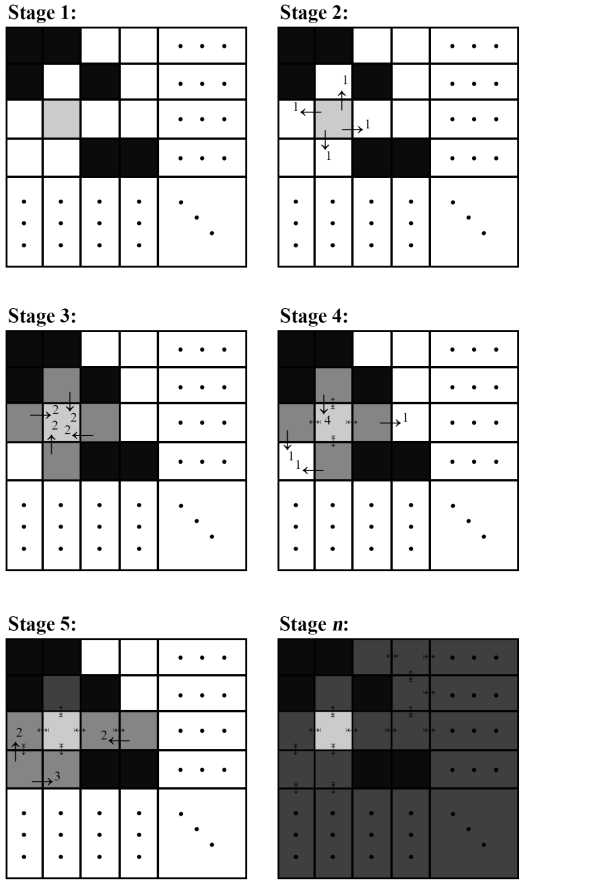

Fig. 1. Illustration of an example of the 5 first stages in Phase 1 and 2, as well as the final stage n reached at the end of those phases.

C. Phase 2: System Size Reporting

In this phase, that is run in parallel with the previous phase, the main idea is that whenever a module is discovered and all of its subsequent children finish discovering their respective sub-trees (or if this module doesn't have any children), this module should report its own sub-tree size to its parent. The sub-tree size of a given module is the size of the tree structure that is rooted at this module. Thus, this process of sub-tree sizes reporting will emerge from the leafs to the leader module.

Type 4 messages are now introduced (cf Table I). These messages are only passed from child to parent and they carry out an integer value named "sub-tree_size" that represents the sub-tree size of the child node.

After a module has been discovered, and it has sent out type 1 messages to all of its neighbours (potential children) and it received back all reply messages (either type 2 or type 3), it would: If all neighbours replied with type 3 messages or if it has no neighbours other than its parent (both cases represent the case of a leaf) → send a type 4 message to its parent, with value 1. If one or more neighbours replied with a type 2 message (i.e., the node has at least one child) → wait for all children to send their sub-tree sizes, and only when they do, proceed by sending a type 4 message to its own parent, which now carries its own sub-tree size that is the sum of all sub-tree sizes received from its children + 1 (plus itself). This process keeps on repeating until eventually all sub-tree sizes would be transmitted from child to parent and the leader module would receive the final total system size (the sum of its direct children sub-tree sizes + 1). The Algorithm's part concerning the second phase is presented in algorithm - phase 2.

Continuing and expanding on the illustration example in Figure 1, medium grey boxes represent more precisely discovered modules that did not send their sub-tree size, while dark grey boxes represent modules that have been discovered and have sent their sub-tree size to their respective parent. In stage 4, and as the module above the leader has no neighbours to discover, this module directly proceed to phase 2 by reporting its sub-tree size (that is 1) to its parent (the leader module). In stage 5 the leader module receives the sub-tree size of the module above it, rendering the module above it in the dark grey state, meaning it has finished its tasks for phases 1 and 2. This process keeps on repeating, until all modules are in the dark grey state, and the total system size is reported to the leader (stage n). Having such information available, the leader calculates the least amount of bits needed to code globally unique IDs for the whole system and is thus ready for the ID assignment phase. This state of the algorithm represents the end of phases 1 and 2.

D. Phase 3: Unique ID Assignment

This phase's goal is to distribute the globally unique IDs (all coded with the same number of bits) to all modules of the system. After the calculation of the least amount of bits needed, the leader module proceed by assigning the ID 0 to itself, and sending IDs to its children via type 5 messages

Global Unique ID Assignment Algorithm - Phase 2

```
subtree_size ← 1
subtree_size_sent ← false

if received message then
  switch message.type do
    case 1
      if is_discovered = true then
        ...
      else
        ...
        if neighbours.size = 0 AND !subtree_size_sent then
          CHECK( )
        else
          for each neighbour in neighbours do
            send type 1 message to neighbour
          end for
        end if
      end if
    case 2
      ...
    case 3
      ...
      if neighbours.size = 0 AND !subtree_size_sent then
        CHECK( )
      end if
    case 4
      child ← find message.origin in children
      child.subtree_size ← message.subtree_size
      subtree_size += message.subtree_size
      if neighbours.size = 0 AND !subtree_size_sent then
        CHECK( )
      end if
end if

procedure CHECK( )
  if children.size = 0 OR received all children subtree_size then
    if leader = false then
      send type 4 message to parent
      subtree_size_sent ← true
    else
      calculate least necessary bits
      ...
    end if
  end if
end procedure
```

(cf Table I). When a module receives a type 5 message, it assigns the ID number in this message as its unique ID and proceed by sending type 5 messages to its children (if it has any). The sending of type 5 messages to children should respect the following: (i) send the module's ID + 1 to the first child, (ii) send the module's ID + 1 + $\sum_{n=1}^{i-1} S_n$ to the i^{th} child, where S_n is the sub-tree size of the n^{th} child. This assures that whenever a module receives an ID via a type 5 message, the range of IDs going from its own ID to its ID + its sub-tree size, is reserved for it and for its sub-tree.

An illustration example of the execution of this phase is presented in Figure 2, where white boxes now represent modules that finished their execution. Supposing that the module to the left of the leader has a sub-tree size of 9, the leader would send the ID 0 + 1 (i.e., its own ID + 1) to it, while sending the ID 0 + 1 + 9 = 10 to the module too its right. This reserves all IDs from 1 till 9 to the sub-tree of the module to the left of the leader. The same principle applies supposing that the right module has a sub-tree size of 87, the ID 0 + 1 + 9 + 87 = 97 will be sent to the third

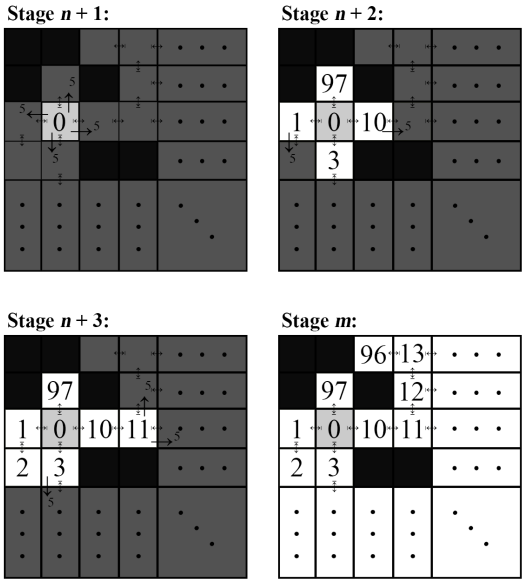

Fig. 2. Illustration of the first three stages of phase 3, plus the final stage m of the system after the completion of the algorithm.

and last child of the leader. This process keeps on repeating from parent to child until all modules in the system receive their globally unique IDs. The Algorithm's part concerning the third phase is presented in algorithm - phase 3.

Global Unique ID Assignment Algorithm - Phase 3

```
if received message then
  switch message.type do
    case 1
      ...
    case 2
      ...
    case 3
      ...
    case 4
      ...
    case 5
      final_id ← message.id
      next_id ← message.id + 1
      for each child in children do
        send type 5 message to child
        next_id += child.subtree_size
      end for
end if

procedure CHECK( )
  if children.size = 0 OR received all children subtree_size then
    if leader = false then
      ...
    else
      calculate least necessary bits
      unique_id ← 0
      next_id ← 1
      for each child in children do
        send type 5 message to child
        next_id += child.subtree_size
      end for
    end if
  end if
end procedure
```

E. Complexity Study

As the time and energy complexity (i.e., execution time of the algorithm and energy consumed throughout the algorithm) are directly linked to the number of messages transmitted in the system, the latter will be developed in this section. Let us group the messages into three groups: messages of type 1-2-3, messages of type 4 and messages of type 5. Let B be the number of borders for each module (i.e., the maximum number of neighbours for each node). As one of the those neighbours must be the module's parent, $B-1$ is thus the maximum number of possible children for any module (except for the leader having a maximum of B possible children). Let N be the total number of modules in the system, and P is the maximum number of possible children for any module in the system except the leader (i.e., $P = B-1$).

For the first group of messages (of type 1-2-3), each discovered module would try to discover a maximum of $B-1$, or P, neighbours. This process would require the discovered module to send a maximum of P type 1 messages and accordingly receive back a maximum of P type 2 and/or type 3 messages (either confirming or declining the type 1 messages). Let M_1 be the number of messages transmitted from the first group, M_1 would be strictly less than: $2 \times P \times N$ (as there will be edge modules in the system with neighbours strictly less than P).

For the second group of messages (of type 4), each discovered module with no children, or with all of its children reporting their respective sub-tree sizes, would report its own sub-tree size to its parent (unless the module in question is the leader). Thus M_2 would be equal to: $N-1$, where M_2 is the number of messages transmitted from the second group.

For the third group of messages (of type 5), and after the initiation of the third phase by the leader node, each module in the system would eventually receive its unique ID via a message from its parent (except for the leader). Thus M_3 would be equal to $N-1$, where M_3 is the number of messages transmitted from the third group.

The total number of messages transmitted throughout the execution of the distributed algorithm would then be $M = M_1 + M_2 + M_3$, thus $M < 2 \times P \times N + (N-1) + (N-1)$. As the maximum number of borders (i.e., neighbours) for a module in any given modular robots system would never exceed 8 to 10, the complexity of message transmission of our algorithm is linear to N, considering P as a constant.

III. Analysis Study and Numerical Results

In this section we will study the impact of the modular robots' initial properties on the performance of the proposed algorithm, mainly the initial system shape and the position of the leader. In order to assert our studies and discussions, our distributed algorithm has been implemented in VisibleSim [1], simulator dedicated communicating modular robots.

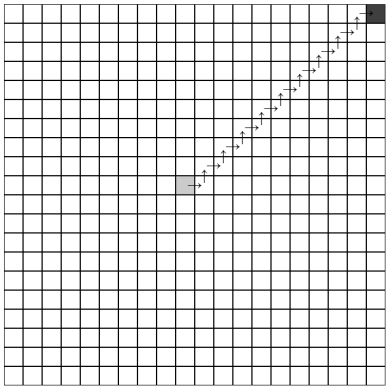

Fig. 3. Filled shape system example: square

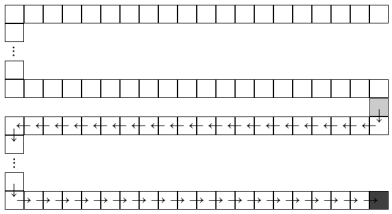

Fig. 4. Chain shaped system example: snake

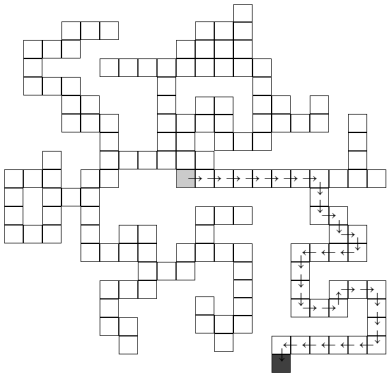

Fig. 5. Randomly shaped system example

A. Impact of system shapes

In the previous section we showed that the complexity of our algorithm is affected by the number of neighbours which is largely related to the shape of the system. Next we develop three cases of initial system shapes: filled shape, chain shape, random shape. Throughout our development, we compared the simulated energy and time complexity for each system shape, for system sizes ranging from 100 modules to 40,000 modules.

1) Filled Shape: This kind of shapes aims at maximising the number of neighbours at each module, meaning it aims at having the minimum number of holes in the system. An example of such shapes is presented in Figure 3. While such shapes maximises the energy complexity of the algorithm (i.e., total number of messages exchanged), it minimised its time complexity (i.e., time to finish execution). This is due to the distance (in terms of number of hops) between the leader and the furthest module in the system being minimised. As this algorithm is distributed, multiple tasks (i.e., messages) can be executed simultaneously at different modules. We will define a ”tick” as the unit of time needed to transmit a message (through a pair of connected borders) between any two neighbour modules. Let us consider a case in which we have 40,000 modules. One way to create a filled shape would be to structure them in a 200*200 square. After 4 ticks, the leader (positioned at the center of the system) would have sent out four type 1 messages to its neighbours. After 4 additional ticks, each of the discovered modules would have replied to the leader with a type 2 message (1 tick) and proceeded by transmitting type 1 messages to their neighbours (3 ticks), and so on. Let us consider the module in the top-right corner of the system as the furthest module from the leader. In order to discover this module (via messages from the first group), messages initiating from the leader module should traverse a staircase pattern in order to reach this furthest module. As the leader is at the center of the network, the number of messages needed to reach the furthest module is 2 * (100 - 1) + 1 = 199 messages (of type 1). In other words, considering that in average the module approaching the furthest module in the system is discovered second (between all three neighbours), 199 * (1 + 2) = 597 ticks are needed to reach any module in the system via type 1 messages (while simultaneously replying to other modules with type 2/3 messages). After that, 199 other ticks are needed to report the whole network size to the leader (with the longest reporting route being from the furthest module to the leader). Finally, 199 additional ticks are needed for the global unique IDs assignment (with the longest assignment route being from the leader to the furthest module). All in all, 995 ticks are required for the algorithm to terminate successfully. This is in fact, as we will observe in further discussions, the minimum time complexity among all proposed shapes. Thus, reaching the furthest module in the system via the shortest messages route possible would maximise the energy consumption while minimizing the time complexity. In actuality, the furthest module in the logical tree cannot be assured to be the corner module, as this algorithm is distributed, and message handling and concurrent events could result in unbalanced logical tree structures (i.e., even in a 200*200 square, the furthest module could be above 400 hops away, instead of 199). Figure 6 plots the simulation results obtained with VisibleSim of the energy and time complexity of filled shapes systems for various system sizes.

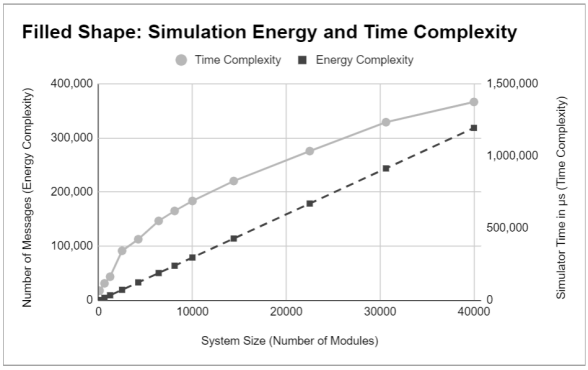

Fig. 6. Complexity progression for filled shapes systems

2) Chain Shape: This kind of shapes aims at minimising the number of neighbours at each module, meaning it aims at building a chain-like structure, where each node has a

maximum of two neighbours (i.e., one parent neighbour and one child neighbour). An example of such shapes is presented in Figure 4. While such shapes minimises the energy complexity of the algorithm, it maximises its time complexity. This is due to the distance between the leader and the furthest module being maximised. Setting the leader as the middle module in the system, reaching the furthest module in the system requires 40,000 / 2 = 20,000 sequential type 1 messages. Following the same reasoning from the previous subsection, 20,000 * 4 = 80,000 ticks are required for the algorithm to terminate successfully. This would in fact result in the maximum time complexity among all proposed shapes. Figure 7 plots the simulation results obtained with VisibleSim of the energy and time complexity of filled shapes systems for various system sizes.

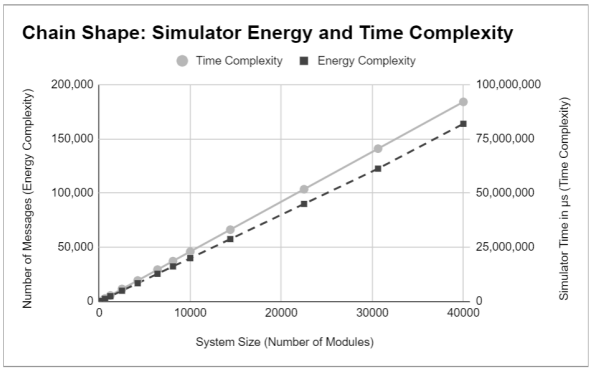

Fig. 7. Complexity progression for chain shaped systems

3) Random Shape: This kind of shapes is random, thus resulting in a variable number of neighbours at each module (ranging from the minimum of 2, to the maximum of $B-1$). An example of such shapes is presented in Figure 5.

Such shapes would result in a random energy and time complexity, both bounded by the time and energy complexity of the previous two subsections. Defining T as the time complexity and E as the energy complexity, we have:

$$T_{filled} \leq T_{random} \leq T_{chain}$$

$$E_{chain} \leq E_{random} \leq E_{filled}$$

This is due to the distance between the leader and the furthest module in the system being randomized between the minimum and maximum distance values, coming respectively from the filled shaped and chain shaped systems for a particular system size. In fact, a randomly shaped system is a combination of multiple filled and chain shaped subsystems. The more filled shapes subsystems composing the randomly shaped system, the more the energy and time performances approach those of a random shape system. The more chain shaped subsystems composing the randomly shaped system, the more the energy and time performances approach those of a chain shaped system. Figure 8 plots the simulation results of the energy and time complexity of filled shapes systems for various system sizes. As each simulation generates a completely random and new system shape, the resulting curves don't follow a certain pattern. Nevertheless, both complexities are bounded by the complexities of the filled and chain shaped systems, as expressed previously. An example of a randomly generated system shape in VisibleSim can be seen in Figure 9.

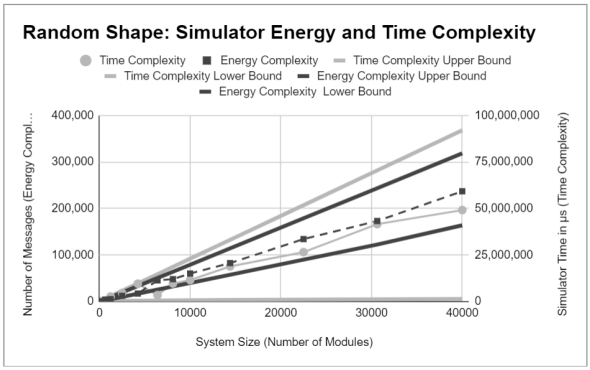

Fig. 8. Complexity progression for randomly shaped systems

In actuality, users could shape their modular robots system to their liking, prior to the execution of our algorithm, in order to accommodate to their particular needs and requirements. For a system size of 40,000 modules, a network shape of a 200*200 matrix could be chosen for the best time performance, or a 1*40,000 chain shaped system could be chosen for the best energy performance, or any system in between (e.g., 2*20,000 or 4*10,000 or any randomly shaped system) in order to balance both metrics.

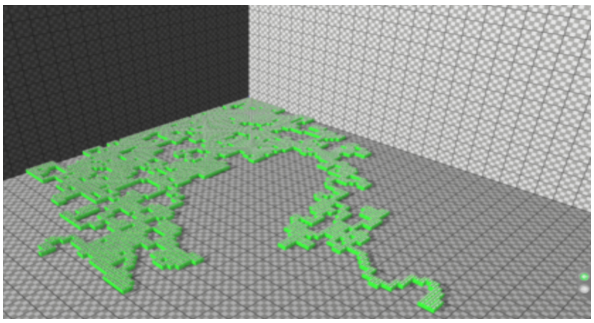

Fig. 9. An example of a randomly generated system shape in VisibleSim with a system size of 2500 modules

B. Impact of the position of the leader

Finally, it is worth noting that in all our cases, the leader is elected as the center module in the system. This is fairly intuitive as you would like your leader to be the closest possible to the whole system, or in other words, you would like your leader to have the shortest distance with respect to the furthest module in the system (as this was proven in the previous subsections to be a key factor in the performance of the algorithm). For example, if in a 200*200 matrix, we were to place the leader at the bottom-left corner, the distance between it and the ideal furthest module would've been 2 * (200 - 1) + 1 = 399 messages, thus the algorithm's termination would've required 1,995 ticks (compared to the

995 ticks required if the leader was at the center of the system). On the other hand, electing the leader anywhere on the system would not affect the energy consumption (i.e., number or messages transmitted), it would only affect the degree at which multiple messages are simultaneously being handled at different modules.

IV. Conclusion and Future Work

In this paper, we studied the unique node ID assignment related issues in modular robots. It is inspired from the algorithms proposed in the sensor networks domain and adapted in order to suit the use case of modular robots. Our proposed algorithm has been greatly optimised, bringing down the number of exchanged messages between modules while also greatly decreasing the messages' length. The three phases of the distributed algorithm are explained. We showed that our algorithm has $O(N)$ complexity. Following that, a detailed discussion and numerical results were presented to show the performance of our algorithm while comparing the effects of various system shapes on the energy and time complexity of the algorithm. The discussion also equipped the users with enough knowledge in order to be able to decide how to shape their modular robots systems prior to the execution of our distributed algorithm, in the aim of balancing both energy and time requirements. Also, simulation results in VisibleSim are showcased that further support our theoretical analysis.

As discussed in the previous section, electing the center module of the system as the leader for our algorithm results in the best time complexity for the algorithm (whereas the energy complexity remains constant regardless of the leader's position). In future work, we plan on studying the effects of having multiple leaders distributed evenly throughout the system on the time complexity of the algorithm. We also aim at developing an analogous algorithm that supports the removal and addition of modules throughout the execution of the algorithm as well as after its completion. This is specifically tricky in the modular robots use case, as the detection of a removal or addition of a module is extremely hard. This is due to having modules constantly moving from one position to another, that is, disconnecting from one module's border in order to connect to another. Such behaviour for example should not be treated as a removal plus an addition of a module. Finally, we would want to further assert our results and simulations by executing our distributed algorithm on real-life modular robots systems.

Acknowledgement

This work has been supported by the EIPHI Graduate School (contract ANR-17-EURE-0002).

References

[1] Visiblesim. `https://projects.femto-st.fr/projet-visiblesim/en`. Accessed: 2020-03-01.

[2] Reem J Alattas, Sarosh Patel, and Tarek M Sobh. Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems*, 95(3-4):815–828, 2019.

[3] Alberto Brunete, Avinash Ranganath, Sergio Segovia, Javier Perez de Frutos, Miguel Hernando, and Ernesto Gambao. Current trends in reconfigurable modular robots design. *International Journal of Advanced Robotic Systems*, 14(3):1729881417710457, 2017.

[4] Joshua J Daymude, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. Improved leader election for self-organizing programmable matter. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 127–140. Springer, 2017.

[5] Bo Dong, Tianjiao An, Fan Zhou, Keping Liu, Weibo Yu, and Yuanchun Li. Actor-critic-identifier structure-based decentralized neuro-optimal control of modular robot manipulators with environmental collisions. *IEEE Access*, 7:96148–96165, 2019.

[6] Yuval Emek, Shay Kutten, Ron Lavi, and William K Moses Jr. Deterministic leader election in programmable matter. *arXiv preprint arXiv:1905.00580*, 2019.

[7] Nicolas Gastineau, Wahabou Abdou, Nader Mbarek, and Olivier Togni. Distributed leader election and computation of local identifiers for programmable matter. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 159–179. Springer, 2018.

[8] Md. Rakibul Haque, Mahmuda Naznin, and Rifat Shahriyar. Distributed low overhead id in a wireless sensor network. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 12:1–12:4, 2016.

[9] Andrew B Jones, Thomas Cameron, Benjamin Eichholz, David Loegering, Taylor Kray, and Jeremy Straub. Self-reconfiguring modular robot learning for lower-cost space applications. In *2019 IEEE Aerospace Conference*, pages 1–6. IEEE, 2019.

[10] Jung Hun Kang and M Park. Structure-based id assignment for sensor networks. *International Journal of Computer Science and Network Security*, 6(7):158–163, 2006.

[11] Jialiu Lin, Yunhuai Liu, and Lionel M Ni. Sida: self-organized id assignment in wireless sensor networks. In *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–8. IEEE, 2007.

[12] André Naz, Benoît Piranda, Seth Copen Goldstein, and Julien Bourgeois. A time synchronization protocol for modular robots. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 109–118. IEEE, 2016.

[13] ElMoustapha Ould-Ahmed-Vall, Douglas M Blough, Bonnie Heck Ferri, and George F Riley. Distributed global id assignment for wireless sensor networks. *Ad Hoc Networks*, 7(6):1194–1216, 2009.

[14] Michael Park, Sachin Chitta, Alex Teichman, and Mark Yim. Automatic configuration recognition methods in modular robots. *The International Journal of Robotics Research*, 27(3-4):403–421, 2008.

[15] Roberto Petroccia. A distributed id assignment and topology discovery protocol for underwater acoustic networks. In *2016 IEEE Third Underwater Communications and Networking Conference (UComms)*, pages 1–5. IEEE, 2016.

[16] Federico Pratissoli, Andreagiovanni Reina, Y Kaszubowski Lopes, Lorenzo Sabattini, and Roderich Groß. A soft-bodied modular reconfigurable robotic system composed of interconnected kilobots. In *Proceedings of the 2019 IEEE international symposium on multi-robot and multi-agent systems (MRS 2019)*, 2019.

[17] Joshua R Smith. Distributing identity [symmetry breaking distributed access protocols]. *IEEE Robotics & Automation Magazine*, 6(1):49–56, 1999.

[18] Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. Distributed self-reconfiguration using a deterministic autonomous scaffolding structure. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 140–148, 2019.

[19] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. A distributed self-assembly planning algorithm for modular robots. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018*, pages 550–558, 2018.

[20] Meibao Yao, Christoph H Belke, Hutao Cui, and Jamie Paik. A reconfiguration strategy for modular robots using origami folding. *The International Journal of Robotics Research*, 38(1):73–89, 2019.

[21] Qingchao Zheng, Zhixin Liu, Liang Xue, Yusong Tan, Dan Chen, and Xinping Guan. An energy efficient clustering scheme with self-organized id assignment for wireless sensor networks. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems*, pages 635–639. IEEE, 2010.

[22] Hongbo Zhou, Matt W Mutka, and Lionel M Ni. Reactive id assignment for sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, pages 6–pp. IEEE, 2005.