ABC-Center: Approximate-Center Election in Modular Robots

André Naz[1], Benoît Piranda[1], Seth Copen Goldstein[2] and Julien Bourgeois[1]

***Abstract*— Modular robots are composed of many independent connected modules which are able to achieve common goals through communications. Many distributed algorithms have better performance if the modules that have to communicate with all the others, are placed at the center of the system. In this paper, we propose ABC-Center, an iterative algorithm for electing an approximate-center module in modular robots. ABC-Center uses $O(1)$ space per module and $O(kd)$ time, where k is the number of iterations required to terminate and d the diameter of the system. We evaluated our algorithm both on hardware modular robots and in a simulator for large ensemble of robots. The average expected eccentricity of the module elected by ABC-Center is less than 1.25 blocks off for random systems composed of up to 1000 modules. Furthermore, experiments show that our algorithm terminates after a few iterations. Hence, ABC-Center is scalable and adapted to modular robots with low memory resources.**

***Index Terms*— Distributed algorithm, Modular robotic, Center election**

I. Introduction

Over the past two decades, reconfigurable modular robotic has emerged as a new way to design robotic systems. Those robots with reconfigurable shapes are composed of many independent connected modules which cooperatively perform specific tasks and thus achieve common goals. Each module can be seen as an embedded system that holds its own computational and communication capabilities, sensors and actuators. In this article, we consider modular robot systems forming a non-anonymous point-to-point unweighted network in which modules use neighbor-to-neighbor communications. To illustrate our work, we used lattice-based modular robots called Blinky Blocks [1]. Blinky Blocks are centimeter-size blocks that are connected to each other through serial links on the faces. Every block can have up to 6 neighbors. Each block is equipped with an ATMEL ATxmega256A3-AU 8/16-bits 32-MHz micro-controller having 256KB ROM and 16KB RAM [2]. Moreover, the Blinky Blocks can change their color thanks to embedded RGB LEDs. It is important to note that such embedded systems have very limited resources. Storage and computational costs are major concerns when programming these kinds of devices.

This paper addresses the problem of distributively electing a module at the center of a modular robot. Many distributed algorithms and protocols require a single module to act as a coordinator and to communicate with all the other modules. Placing this coordinator at the center of the system often improves the performance. For instance, distributed clock synchronization protocols [3], [4] in which the modules periodically adjust their clock on the clock of the master, are more scalable if the master is located at the center of the system. Indeed, this strategy reduces the time required to synchronize all the system and increases the precision achievable because cumulative estimations are made every hop. This central node can be manually defined but it is more flexible if the system itself elects its center.

The contribution of this paper is to introduce ABC-Center, an iterative algorithm that distributively elects an approximate center module in a modular robot. ABC-Center uses $O(1)$ space per module and $O(kd)$ time, where k is the number of iterations required to terminate and d the diameter of the system. We implemented our algorithm in C and C++ and evaluated it both on hardware Blinky Blocks and in a simulator. Our large set of experiments shows that the average expected eccentricity of the block elected by ABC-Center is less than 1.25 blocks off for random systems with a size up to 1000 blocks. Moreover, we observed that our algorithm terminates after a few iterations. Thus, ABC-Center is suitable for large lattice-based modular robot ensembles with low memory resources. To the best of our knowledge, this is the first asynchronous distributed algorithm that elects an approximate center in such systems with both a constant storage cost and a reasonable execution time in practice.

In Section II, we define the concepts and symbols used in this paper. Afterwards, we briefly discuss the existing methods to compute the center in Section III. Then, in Section IV, we present the general idea of ABC-Center algorithm. In Section V, we provide implementation details and complexity analysis of ABC-Center. In Section VI, we report experiment results.

II. Definitions and notations

The modules constitute a distributed system that can be modeled by an undirected and unweighted graph of interconnected entities $G = (V, E)$, with V a set of vertices (representing the modules), E a set of edges (representing the connections), $|V| = n$ the number of vertices, $|E| = m$ the number of edges. We use the general concepts of graph theory such as the distance d_{xy} between two nodes x and y, the diameter d, the radius r, and the eccentricity of a node.

Various definitions and metrics for graph centrality have been proposed in the past: (Jordan) center, centroid, median, degree centrality, closeness centrality, betweenness centrality,

[1]André Naz, Benoît Piranda, Julien Bourgeois are with Université de Franche-Comté, FEMTO-ST Institute, UMR CNRS 6174, 1 Cours Leprince-Ringuet, 25200 Montbéliard, France, {andre.naz, benoit.piranda, julien.bourgeois}@femto-st.fr

[2]Seth Copen Goldstein is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA, seth@cs.cmu.edu

etc. In this paper, we are interested in the *center* [5]: the set of all modules of minimum eccentricity where the eccentricity of a module is the maximum distance from this module to any other.

III. Related works

As explained in the previous section, several types of central vertices exist. We restrict our study to the existing method to compute the center. For memory cost comparisons, we adopt a system approach and consider that an integer uses $O(1)$ space.

[6] proposes a distributed algorithm to compute the center of asynchronous arbitrary networks in which all the nodes compute their eccentricity. Using breadth-first searches (BFSes) [7], computing the eccentricities of all the nodes at the same time distributively has a time complexity of $O(d)$ if we ignore pileups and a storage cost per node of $O(n)$. Indeed, the longest path from a node to any other is at most a diameter. Furthermore, each node has to store some information for each BFS in progress such as the identifier of the node that launches the BFS and the distance to that node. Performing the BFSes node by node uses $O(nd)$ time and $O(1)$ space per node. Regardless of the variant we consider, these exhaustive approaches are not scalable. They either have a too high time complexity or a too big memory cost to be used in systems composed of thousands of modules with very limited memory resources.

More efficient heuristics have been proposed to compute the center of tree graphs. For instance, [8], [9] propose a distributed method to compute the center of a tree graph in r rounds and $O(1)$ space per node where each node determines its height in the tree. The center is the set of nodes which have a greater height value than all neighbors. Unfortunately, this algorithm is not directly generalizable to arbitrary graphs. [10] proposes Minimax, an efficient sequential algorithm to compute the center of undirected tree graphs using only two BFSes. It picks A, a random node of the tree, B the farthest node from A and C the farthest node from B. The center is at midpoint of path between B and C. However, in arbitrary graphs, Minimax does not always return the exact center. For example, in Figure 1, Minimax returns one of the module in the diagonal in blue.

Some sequential approaches inspired from Minimax have recently been proposed in order to compute a central vertex of arbitrary graphs using a limited number of BFSes. For example, [11] proposes the 4-sweep algorithm which finds a node with low eccentricity in arbitrary graphs using 4 BFSes. It returns the best result of two Minimax. However, in the general case, 4-sweep algorithm does not return the exact center. For instance, in Figure 1, 4-sweep returns one of the node on the two diagonals. [12] proposes SumSweep, a sequential method to find the exact center of arbitrary graphs using in practice a small number of BFSes. The eccentricity of each node is estimated during BFSes performed from least central vertices. However, termination detection and local eccentricity estimations will not be immediate in a distributed adaptation of SumSweep. More information exchanges will be required than BFSes.

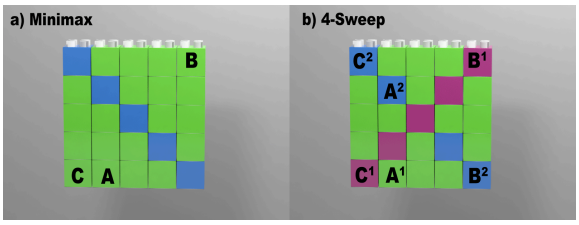


Fig. 1: Minimax and 4-Sweep failure case.

IV. ABC-Center at a glance

In this section we present the general idea of ABC-Center to elect an approximate center of the system. Our algorithm is based on Minimax and 4-Sweep. We assume that each Blinky Block has a unique identifier noted id and that there is no topology change or failure during the election process. ABC-Center uses the id to break the ties. It is a way to select a single block in distributed context.

For pedagogical purpose, a sequential version of ABC-Center is shown in Algorithm 1. ABC-Center iteratively finds an approximate center of the system. At the beginning, all the blocks are candidates (line 1). At each iteration i, we pick A^i the minimum-id block among the candidates (line 3). Then, we select B^i, one of the farthest candidates from A^i (line 4) and C^i one of the farthest candidates from B^i (line 5). B^i and C^i are extremities of the system composed with the candidates.

Candidates for the $(i+1)^{th}$ iteration are the most equidistant blocks from B^i and C^i among the candidates at iteration i (line 6). The most equidistant blocks are defined as the blocks that minimize $|distance\ to\ B^i - distance\ to\ C^i|$. Let $\mathscr{P}^i$ be the plane that contains the most equidistant candidate blocks from B^i and C^i. The candidates at iteration $n+1$ are the blocks that belong to $\bigcap_{i=1..n} \mathscr{P}^i$.

When less than 3 blocks remain candidates, one of the closest candidates from the last B and the last C is designated as the approximate center of the system (line 8).

Figure 2 shows ABC-Center step-by-step execution on a cube. At each iteration $\mathscr{I}$, the problem is simplified by one dimension. The first iteration $\mathscr{I}^1$ gives a discrete plane, $\mathscr{I}^2$ a discrete line, $\mathscr{I}^3$ a set of 3 blocks and $\mathscr{I}^4$ isolates a center.

V. Distributed implementation of ABC-Center

Our distributed version of ABC-Center uses for each iteration i, a multi-criteria leader election algorithm to find A^i, B^i and C^i. We first describe this election algorithm. We subsequently detail step-by-step how the distributed version of ABC-Center works on a basic example. We then discuss the complexity of this version in terms of execution time, messages and storage cost.

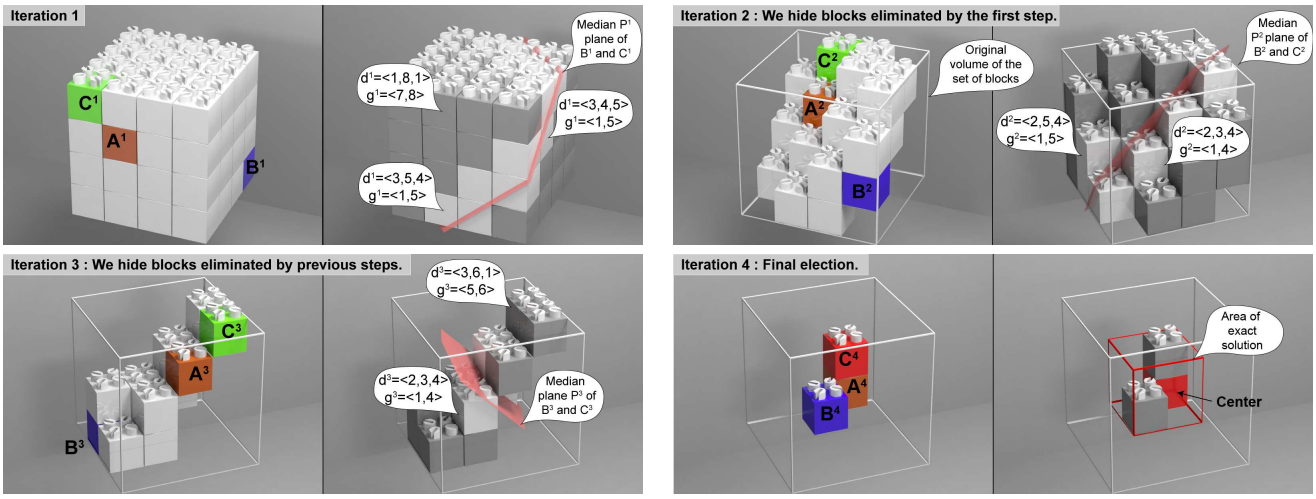


Fig. 2: ABC-Center step-by-step execution on a 4*4*4 cube of Blinky Blocks. For every block k we note $d_k^i = < d_{kA^i}, d_{kB^i}, d_{kC^i} >$ and $g_k^i = < |d_{kB^i} - d_{kC^i}|, max(d_{kB^i}, d_{kC^i}) >$.

Data: $G = (V, E)$ // graph representing the system
Result: $Center$ // an approximate center of the system

1 $Candidates \leftarrow V$;
2 **while** $|Candidates| > 2$ **do**
3 $\quad A \leftarrow min - id(Candidates)$;
4 $\quad B \leftarrow min - id(\{k \in Candidates \mid d_{kA} = \max_{l \in Candidates} d_{lA}\})$;
5 $\quad C \leftarrow min - id(\{k \in Candidates \mid d_{kB} = \max_{l \in Candidates} d_{lB}\})$;
6 $\quad Candidates \leftarrow \{k \in Candidates \mid |d_{kB} - d_{kC}| = \min_{l \in Candidates} |d_{lB} - d_{lC}|\}$;
7 **end**
8 $Center \leftarrow min - id(\{k \in Candidates \mid max(d_{kB}, d_{kC}) = \min_{l \in Candidates} max(d_{lB}, d_{lC})\})$;

Algorithm 1: Sequential version of ABC-Center.

A. Multi-criteria leader election algorithm

Our multi-criteria leader election algorithm is based on T.-Y. Cheung's algorithm [7] that builds a breadth-first spanning-tree with feedback in an arbitrary asynchronous network. M. Raynal showed in [13] that network traversal algorithms such as T.-Y. Cheung's algorithm can be used for leader election. We modified T.-Y. Cheung's algorithm into $electBlock(c, optFunc, x, id)$ to elect, among the candidate blocks for which the boolean c is equal to true, a single block that optimizes a variable x according to $optFunc \in \{min, max\}$ using the unique identifier id as a tie breaker (see Algorithm 2). Each block has its own variables c, x and id. c is equals to true if the module is candidate for the election, false otherwise. x can be a tuple. A comparison order has to be defined on x. In case of equality, the tuple with the lowest id is selected.

In $electBlock$, every block locally stores the temporary optimized value of x in the variable $optX$. The id of the candidate block that optimizes x is stored in $optId$ and the distance to that block is stored in $optDist$. The values of $optX$, $optId$ and $optDist$ are progressively learned by all the blocks during the execution of $electBlock$.

The evaluation function $evaluation(optFunc, x, i)$ returns $BETTER$ if the tuple (x, i) optimizes the local solution $(optX, optId)$ according to $optFunc$. It returns $EQUAL$ if $(x, i) = (optX, optId)$. Otherwise, it returns $WORSE$. For instance, if $optX = 2$ and $optId = 1$, $evaluation(max, 3, 2)$ returns $BETTER$. The same call, returns $EQUAL$ if $optX = 3$ and $optId = 2$, whereas it returns $WORSE$ if $optX = 3$ and $optId = 1$.

In our leader election algorithm, each candidate block starts a graph traversal by sending to all its neighbors an $ELECT$ message that contains its value of x and its id (lines 2-12). Graph traversals are concurrent. If a block receives better values according to $optFunc$ via an $ELECT$ message, it forgets about the previous graph traversal, and starts participating in new one (lines 13-29). Modules send back confirmation messages $CONFIRM$ which progressively go back up to the module that will win the election. A module b_l sends a $CONFIRM$ message to the block b_j, either if b_l has received from b_j an $ELECT$ message containing values equal to the current optimal values stored in $optX$ and $optId$ but with a farther distance to b_{optId} (lines 16 and 28), or if b_l has received a $CONFIRM$ message from each of its other neighbors (lines 25 and 37). A graph traversal terminates as soon as the module that initiated it, has been informed by all its neighbors that it has the best values for x and id among the candidate blocks (line 35). Although all modules initiate a graph traversal, only a single one will terminate and the block that initiates this traversal will win the election. We assume that there is no topology change or failure in the system during the election process.

B. ABC-Center detailed execution on a line of 4 blocks

Figure 3 shows ABC-Center step-by-step execution on a line of four blocks. Election messages are tagged with the

```
1  electBlock(c,optFunc,x,id) algorithm detailed for any
   block b_l:

2  Initialization of b_l
3  optX ← WORST_X_VALUE; optId ← 0;
4  optDist ← 0; parent ←⊥; Wait ← ∅;
5  if c = true then
6  |  optX ← x; // value of the variable we
   |      want to optimize during the election
7  |  optId ← id;
8  |  for each b_j ∈ Neighbors do
9  |  |  send ELECT(optX, optId, optDist) to b_j;
10 |  |  wait ← wait ∪ {b_j};
11 |  end
12 end

13 When ELECT(x,i,d) is received by b_l from b_k do :
14 if (evaluation(optFunc,x,i) = BETTER) OR
   ((evaluation(optFunc,x,i) = EQUAL AND (optDist > d+1))
   then
15 |  if (evaluation(optFunc,x,i) = EQUAL) AND (parent !=
   |  ⊥) then
16 |  |  send CONFIRM(optX, optId, optDist − 1) to
   |  |  parent;
17 |  end
18 |  optX ← x; optId ← i; optDist ← d + 1;
19 |  parent ← b_k; Wait ← ∅;
20 |  for each b_j ∈ Neighbors \ {b_k} do
21 |  |  send ELECT(optX, optId, optDist) to b_j;
22 |  |  Wait ← Wait ∪ {b_j};
23 |  end
24 |  if Wait = ∅ then
25 |  |  send CONFIRM(optX, optId, optDist − 1) to
   |  |  parent;
26 |  end
27 else if evaluation(optFunc,x,i) = EQUAL then
28 |  send CONFIRM(x, i, d) to b_k;
29 end

30 When CONFIRM(x,i,d) is received by b_l from b_k do:
31 if (evaluation(optFunc,x,i) = EQUAL) AND (optDist = d)
   then
32 |  Wait ← Wait − {b_k};
33 |  if Wait = ∅ then
34 |  |  if optId = id then
35 |  |  |  // block b_l wins the election
36 |  |  else
37 |  |  |  send CONFIRM(optX, optId, optDist − 1) to
   |  |  |  parent;
38 |  |  end
39 |  end
40 end
```

Algorithm 2: Multi-criteria leader election algorithm $electBlock(c, optFunc, x, id)$ detailed for any block b_l.

iteration number and the role (A,B or C) to prevent the current election from being disrupted by delayed messages of a previous election. At the beginning every block is candidate and launches the election of A^1. When A^1 is finally elected, all the blocks know their distance to A^1. A^1 starts the election of B^1 which is one of the farthest blocks from A^1. Similarly, B^1 starts the election of C^1. Then, C^1 launches the election of A^2. Only the blocks k that have $g_k^1[0] = g_{A^2}^1[0]$ remain candidates for the second iteration. That is to say, the blocks at half-distance between B^1 and C^1. If less than 3 blocks remain candidates, A^2 is elected as an approximate center of the system and the algorithm terminates. Otherwise, the same scheme is repeated until less than 3 blocks remain candidates. An easy way to determine if less than 3 blocks remain candidates is to store the identifiers of 2 remaining candidates in the $CONFIRM$ election messages. If A^i receives 2 different identifiers, it means that at least 3 blocks remain candidates: A^i and the two others. Message size is thus constant.

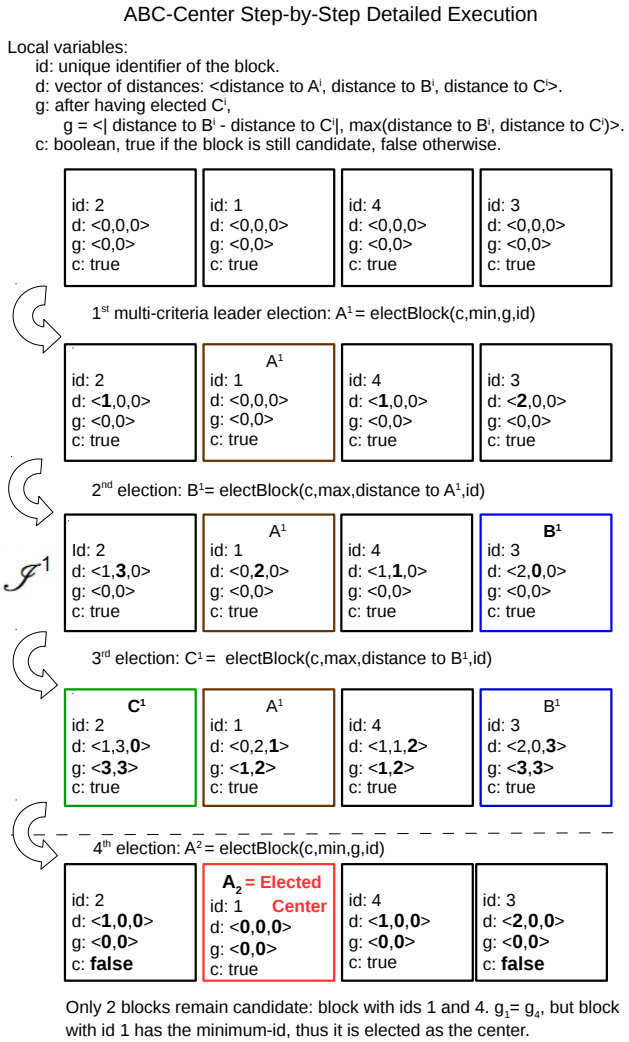


Fig. 3: ABC-Center detailed execution on a line of 4 blocks.

C. Complexity analysis

M. Raynal's termination proof for T.-Y. Cheung's breadth-first spanning-tree algorithm [13] is applicable to show the termination of our multi-criteria leader election. Since the number of candidate blocks always decreases at each iteration, ABC-Center algorithm necessarily terminates.

T.-Y. Cheung's breadth-first spanning-tree algorithm has a time complexity of $O(d)$ [13], because the length of the shortest-path from the root of the tree to any other node is at

most as long as the diameter. If we ignore pileups, our multi-criteria leader election algorithm also has a time complexity of $O(d)$. An iteration is composed of exactly three multi-criteria elections, thus, the time complexity of an iteration is $O(d)$. Hence, ABC-Center uses $O(kd)$ time with k the number of iterations required to terminate.

Breadth-first spanning-tree algorithm has a message complexity of $O(nm)$ where m is the number of edges [14]. During the multi-criteria leader election process, in the worst case, each block launches a graph traversal and a single one terminates. Hence, the message complexity of ABC-Center is $O(kmn^2)$ with k the number of iterations.

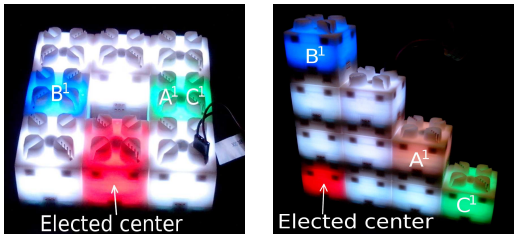
Moreover, ABC-Center algorithm uses a constant number of variables per module and does not dynamically allocate memory. Thus, it uses $O(1)$ space per module.

VI. Experiments and results

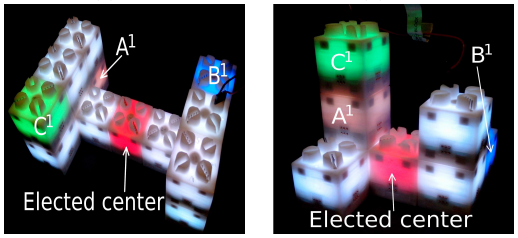
This section presents our experimental evaluation of ABC-Center algorithm performed both on hardware Blinky Blocks and in a simulator for Blinky Blocks called VisibleSim [15]. Through our experiments, we show the effectiveness, the efficiency and the scalability of our algorithm. More precisely, we first check that ABC-Center works well on hardware. Then, we show through some examples that VisibleSim accurately simulates Blinky Blocks behavior. We finally use VisibleSim to study the scalability of our algorithm in terms of accuracy, execution time and number of messages.

A. Evaluation on hardware and validation of VisibleSim

Figure 4 shows ABC-Center results on some basic configurations with hardware Blinky Blocks. For all the considered configurations, the algorithm converges in only one iteration. Moreover, in these cases, computed centers exactly match one of the blocks in the exact center of the systems. Table I gives the execution times of ABC-Center on these configurations.



(a) Torus. (b) Stairs.



(c) Dumbbell. (d) Random.

Fig. 4: ABC-Center executions on different hardware Blinky Blocks configurations.

Shape	Diameter	Execution time (ms)	
		ABC-Center	
		Hardware	Simulator
Torus	4	315.00	304.64
Stairs	6	356.50	335.24
Dumbbell	8	614.90	554.63
Random	5	280.50	280.39

TABLE I: Evaluation of ABC-Center on hardware Blinky Blocks.

Table I shows that execution times for small systems are acceptable. We observe that the execution time does not only depends on the diameter, but on the topology as well. Moreover, simulated execution times on VisibleSim are really close to execution times on hardware. Thus, VisibleSim can be used to accurately benchmark the performance of ABC-Center on much bigger configurations.

B. Scalability evaluation using VisibleSim

In this section, we show the scalability of ABC-Center. We focus our evaluation on two criteria: the effectiveness and the efficiency. We used VisibleSim to simulate the execution of our algorithm on five different categories of configurations namely lines, tori, squared grids, cubes and random systems. Random systems are generated by connecting blocks one by one to the system starting from a single block.

1) Effectiveness evaluation: In order to exhibit the accuracy of ABC-Center, we compare the eccentricity of the elected block and the eccentricity of a block in the exact center. This last block has been determined thanks to the algorithm of Floyd-Warshall [16] for distance computations. We define the absolute error as the difference between the eccentricity of the block elected by ABC-Center and the eccentricity of the exact center. The accuracy of our algorithm depends on two factors: the topology and the unique identifier distribution used as a tie breaker during the elections. ABC-Center always finds a block belonging to the exact center for line, torus, squared-grid configurations and almost always for cube configurations regardless of the size of the system. We focus our study on random systems.

The identifier distribution is especially important during the first iteration: the location of the minimum-id block, used as the starting block of ABC-Center, impacts the performance. Figure 5 shows the influence of the choice of the starting block A^1 on the distribution of the absolute error of ABC-Center for ten random configurations with a size of 1000 blocks and a radius of 15 blocks. For instance, in configuration #1, 94.4% of the blocks lead to an exact solution if they are selected as A^1, while 5.6% of the block lead to an absolute error of 1 block. In some configurations, ABC-Center results to an exact solution from none of the blocks. Hence, the quality of the solution depends on the choice of the starting block A^1. It must be noted that, for these ten configurations, the block elected by our algorithm is always close to the exact center.

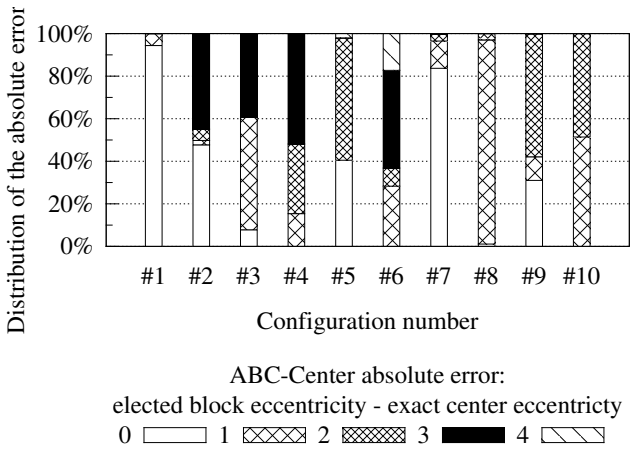


Fig. 5: Influence of the choice of the starting block A^1 on the distribution of ABC-Center absolute error for 10 random configurations of size 1000 blocks, radius 15 blocks and diameter ranging from 27 to 30 blocks.

We define the expected eccentricity of ABC-Center for a configuration as the average eccentricity of the blocks elected by n ABC-Center executions where A^1 is a different block every time. For instance, the expected eccentricity of ABC-Center of configuration #1 in Figure 5 is equal to 15.06 blocks[1]. The expected absolute error for a configuration is the difference between the expected eccentricity and the eccentricity of an exact center (0.6 block for configuration #1 in Figure 5). Figure 6 shows the average expected absolute error of ABC-Center versus the size of the system. We observe that the average expected absolute error increases slowly with the size of the system. The average expected absolute error is less than 1.25 blocks for random systems with a size ranging from 1 block to 1000 blocks.

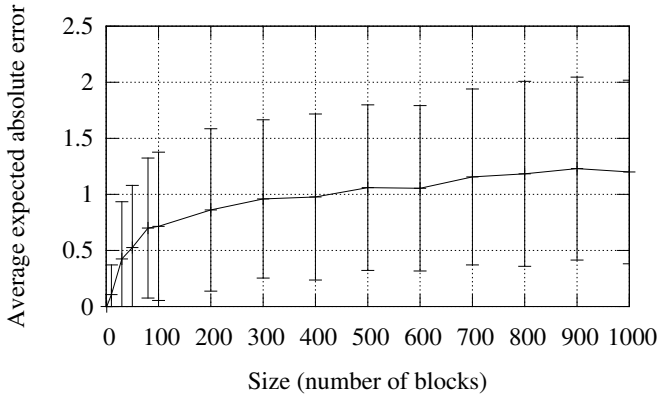


Fig. 6: Average expected absolute error (± standard-deviation) of ABC-Center versus the size of the system. Each point represents 1000 executions on different random systems.

2) Efficiency evaluation: In order to study the efficiency of ABC-Center, we measured for each configuration category three performance indicators: the number of iterations required to converge, the execution time and the number of messages exchanged.

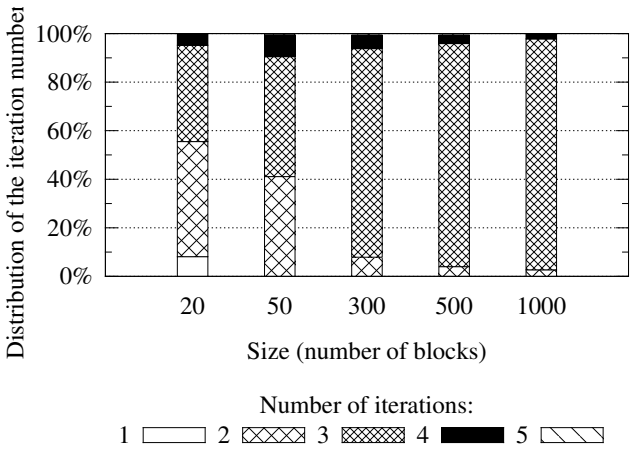


Fig. 7: Distribution of the number of iterations required by ABC-Center to terminate on random systems. For each size, 1000 executions on different random systems were performed.

a) Number of iterations: We observed that regardless of the system size, lines, tori, squared grids (with a side greater than 2 blocks) and cubes (with a side greater than 3 blocks) require a constant number of iterations (respectively one, one, two and three). That is why we continue our study on random systems. Figure 7 shows the distribution of the number of iterations required by ABC-Center to converge for random systems. We observe that most of the time ABC-Center needs 3 iterations for important systems with more than 300 blocks. For every tested system, at most 5 iterations were necessary.

b) Average execution time of an iteration: Figure 8 shows the simulated average execution time of an iteration of

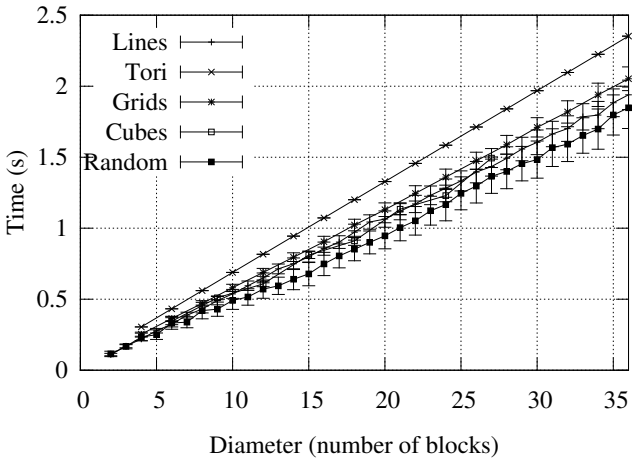


Fig. 8: Simulated average execution duration (± standard-deviation) of an iteration of ABC-Center versus the diameter of the system. Each point represents 1000 executions.

[1] $94.4\% * 15 + 5.6\% * 16 = 15.06$

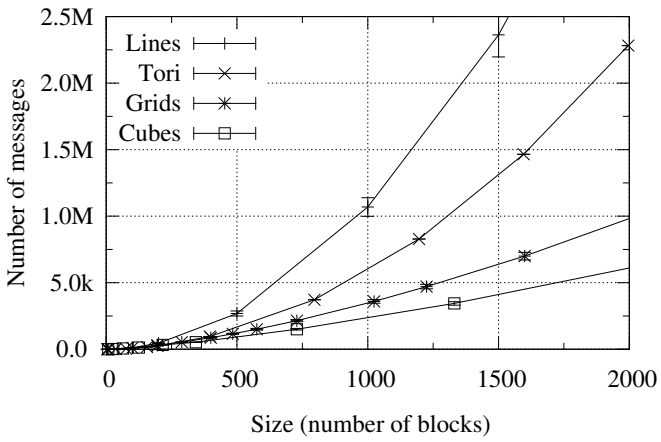


Fig. 9: Average number of messages ($\pm$ standard-deviation) exchanged during an iteration of ABC-Center versus the size of the system. Each point represents 100 executions.

ABC-Center according to the diameter of the system on lines, tori, grids, cubes and random systems. It must be understood that for the same diameter, two different topologies can have a completely different size. For instance, a line of diameter 36 blocks is composed of 37 blocks, whereas a cube of diameter 36 blocks is composed of 2197 blocks. We observe that regardless of the topology and the size of the system, the average execution time of an iteration is linear with respect to the diameter. Moreover, we note that for a given diameter, the average execution time of an iteration is smaller on random systems than on lines, tori, grids and cubes.

c) Number of messages: Figure 9 shows the number of messages exchanged during an iteration of ABC-Center according to the size of the system. We observe that the number of messages seems to have a polynomial growth with the size of the configuration. The worst-case configuration among the tested configurations is the line. For a line of 1500 blocks about 2.37 millions of messages are sent during the execution of ABC-Center.

VII. Conclusions

In this paper, we proposed ABC-Center algorithm, a scalable iterative algorithm for electing an approximate-center module in modular robots. Our algorithm uses $O(1)$ space per module and $O(kd)$ time, where k is the number of iterations required to terminate and d the diameter of the system. The accuracy of our algorithm depends on two factors: the topology and the unique identifier distribution.

We evaluated our algorithm on Blinky Blocks systems. We used both hardware and simulations. We observed that the average expected eccentricity of the block elected by ABC-Center is less than 1.25 blocks off for random systems with a size ranging from 1 block to 1000 blocks. Moreover, our large set of experiments shows that our algorithm terminates after a few iterations. Thus, ABC-Center is suitable for large lattice-based modular robot ensembles with low memory resources.

In future work, we would like to understand why ABC-Center performs better on some systems than on others and why on a given configuration ABC-Center performs better from certain initial A than from others. Moreover, we would like to reduce the message complexity. Currently, all the modules participate to the election of A, B and C, even the modules that have already been eliminated. We are looking for an ingenious heuristic to only involve candidate blocks in the elections. In addition to complexity comparisons, we also would like to experimentally compare ABC-Center to existing algorithms.

VIII. Acknowledgment

This work has been funded by the Labex ACTION program (contract ANR-11-LABX-01-01) and ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F) and ANR (contract ANR-2011-BS03-005).

References

[1] B. T. Kirby, M. Ashley-Rollman, and S. C. Goldstein, "Blinky blocks: a physical ensemble programming platform," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: ACM, 2011, pp. 1111–1116.

[2] "Xmega a3 microcontroller data-sheet." [Online]. Available: http://www.atmel.com/Images/Atmel-8068-8-and16-bit-AVR-XMEGA-A3-Microcontrollers_Datasheet.pdf

[3] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, "Blinky blocks time protocol (BBTP) : protocole de synchronisation des horloges internes de dispositifs embarqués," FEMTO-ST, Research Report RR-FEMTO-ST-2671, Jan. 2015.

[4] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 39–49.

[5] S. Wasserman, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[6] E. Korach, D. Rotem, and N. Santoro, "Distributed algorithms for finding centers and medians in networks," *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 3, pp. 380–401, July 1984.

[7] T.-Y. Cheung, "Graph traversal techniques and the maximum flow problem in distributed computation," *Software Engineering, IEEE Transactions on*, no. 4, pp. 504–512, 1983.

[8] S. C. Bruell, S. Ghosh, M. H. Karaata, and S. V. Pemmaraju, "Self-stabilizing algorithms for finding centers and medians of trees," *SIAM Journal on Computing*, vol. 29, no. 2, pp. 600–614, 1999.

[9] S. Patterson, "In-network leader selection for acyclic graphs," *arXiv preprint arXiv:1410.6533*, 2014.

[10] G. Y. Handler, "Minimax location of a facility in an undirected tree graph," *Transportation Science*, vol. 7, no. 3, pp. 287–293, 1973.

[11] P. Crescenzi, R. Grossi, M. Habib, L. Lanzi, and A. Marino, "On computing the diameter of real-world undirected graphs," *Theoretical Computer Science*, vol. 514, pp. 84–95, 2013.

[12] M. Borassi, P. Crescenzi, M. Habib, W. Kosters, A. Marino, and F. Takes, "On the solvability of the six degrees of kevin bacon game," in *Fun with Algorithms*. Springer, 2014, pp. 52–63.

[13] M. Raynal, *Distributed algorithms for message-passing systems*. Springer, 2013, vol. 500.

[14] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1996.

[15] J. Bourgeois, J. Cao, M. Raynal, D. Dhoutaut, B. Piranda, E. Dedu, A. Mostefaoui, and H. Mabed, "Coordination and computation in distributed intelligent MEMS," in *AINA 2013, 27th IEEE Int. Conf. on Advanced Information Networking and Applications*. Barcelona, Spain: IEEE Computer Society, Mar. 2013, pp. 118–123.

[16] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms: The Floyd-Warshall algorithm*, 2nd ed. McGraw-Hill Higher Education, 2001.