Electing an Approximate Center in a Huge Modular Robot with the k-BFS SumSweep Algorithm

Andre Naz, Benoit Piranda, Julien Bourgeois, Seth Copen Goldstein

Electing an Approximate Center in a Huge Modular Robot with the k-BFS SumSweep Algorithm

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

***Abstract*— Among the diversity of the existing modular robotic systems, we consider in this paper the subset of distributed modular robotic ensembles composed of resource-constrained identical modules that are organized in a lattice structure and which can only communicate with neighboring modules. These modular robotic ensembles that we name LMRs form asynchronous distributed embedded systems. In many algorithms dedicated to distributed system coordination, a specific role has to be played by a leader, i.e., a single node in the system. This leader can be elected using various criteria. A possible strategy is to elect a center node, i.e., a node that has the minimum distance to all the other nodes. Indeed, this node is ideally located to communicate with all the others and this leads to better performance in many algorithms. The contribution of this paper is to propose the k-BFS SumSweep algorithm designed to elect an approximate-center node. We evaluated our algorithm both on hardware modular robots and in a simulator for large ensemble of robots. Results show our algorithm is a good trade-off if accuracy is the main concern. Indeed, it is the most precise approximation algorithm for systems with less than 7,000 modules. In these systems, our algorithm exhibits a relative accuracy between 92% to 100%. Moreover the k-BFS SumSweep algorithm is reasonably efficient in terms of time and communication, and has a limited memory footprint.**

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

I. Introduction

Modular robots have been a rich and productive area of research in the last decades. Many hardware prototypes have been proposed as well as software for managing these new kinds of robots.

In this work, we focus on a specific class of modular robotic systems, namely distributed modular robotic ensembles composed of resource-constrained identical modules that are organized in a lattice structure and which can only communicate with neighboring modules. We name this class of robots LMRs. To be more specific, we are interested in LMRs composed of a large number of modules like it is, for instance, the case in programmable matter [1] and distributed MEMS [2].

These LMRs form distributed embedded systems. Many algorithms for distributed coordination require a specific role to be played by a leader, a single node in the system. Leaders are often used to provide such varied services as time synchronization [3] or general control [4]. The choice of the leader often has a direct impact on the performance. As the LMRs we target exhibit large-average-distance and large-diameter networks [29], a possible strategy is to select a center node, i.e., a node that has the minimum distance to all the other nodes. Indeed, this node is ideally located to communicate with all the others and this leads to better performance in many algorithms. For instance, in [3], we show that, in centralized time synchronization protocols, electing a central leader rather than selecting a random one increases the overall synchronization precision in these systems.

This work is an extension of our previous work on approximate-center node election [5]. In this paper, we propose the k-BFS SumSweep algorithm for approximate-center election.

The main idea behind our algorithm is that central nodes are first and foremost central to the most external nodes. In k-BFS SumSweep, the nodes compute their partial centrality value to a subset of root nodes composed of a random initial node and $k-1$ nodes among the most external ones. Root nodes are consecutively selected using the SumSweep approach that was originally proposed in the sequential algorithm for the exact radius and diameter computation of external graphs [7]. Distributed Breadth-First Searches (BFSes) are used for distributed Single-Source Shortest Paths (SSSP) computations.

The k-BFS SumSweep algorithm runs in $O(kd)$ time using $O(mn^2)$ messages and $O(\Delta)$ memory space per module, where k is an input parameter, d the diameter of the system, n the number of nodes, m the number of links, and Δ the maximum node degree.

To evaluate our algorithm, we applied it to the Blinky Blocks modular robotic systems [8] using both experiments on hardware[1][2] (with up to 63 modules) and simulations[3] (with up to 25,000 modules). Experimental results show our algorithm is a good trade-off if accuracy is the main concern. Indeed, it is the most precise approximation algorithm for systems with less than 7,000 modules. In these systems, our algorithm exhibits a relative accuracy between 92% to 100%. Moreover the k-BFS SumSweep algorithm is reasonably efficient in terms of time and communication, and has a

[1]André Naz, Benoît Piranda, Julien Bourgeois are with University Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, 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

[1]A video showing executions of the k-BFS SumSweep algorithm on hardware Blinky Blocks is available online at https://youtu.be/qaIL85TPZQY

[2]Official Blinky Blocks firmware repository that hosts our algorithm code for hardware experiments: https://github.com/claytronics/oldbb

[3]GitHub repository that hosts our algorithm codes for simulations: https://github.com/nazandre/thesis

limited memory footprint.

The rest of this paper is organized as follows: Section II defines the concepts and notation used in this paper. Then, Section III discusses the existing methods designed to elect a center node. Afterwards, Section IV presents our algorithm, provides its distributed implementation and gives a complexity analysis of it. Section V reports experiment results that are subsequently discussed in Section VI. Finally, Section VII concludes this paper and proposes some future works.

II. System Model and Definitions

We consider distributed embedded systems forming asynchronous non-anonymous point-to-point unweighted and undirected networks. Nodes can only communicate with their immediate neighbors (neighbor-to-neighbor communication model). Furthermore, we assume every node has a unique identifier and maintains a list of its immediate neighbors. We also assume there are no changes to topology nor any node or edge failures during the election process. A solution to manage network dynamics during an election can be found in [9].

We use the general concepts of graph theory to model our system which is represented as an undirected and unweighted graph of inter-connected modules $G = (V, E)$, with V a set of vertices (representing the nodes), E a set of edges (representing the links), $|V| = n$ the number of vertices, $|E| = m$ the number of edges. The distance between two nodes v_i and v_j is $d(v_i, v_j)$, the diameter of the graph is d. Δ represents the graph maximum degree.

Many definitions and metrics for graph centrality have been proposed in the literature. In this paper, we focus on the *center* [10], i.e., the set of all nodes of minimum eccentricity where the eccentricity of a node is the maximum distance from this node to any other.

For complexity calculations, we consider that a variable of a primitive data type (integer, boolean, etc.) uses $O(1)$ memory space. The number of values that can be coded using the algorithm variables may induce limitations on the system size.

III. Related Work

Existing algorithms for center computation can be categorized into four major families, namely exhaustive, graph-specific, sampling-based and probabilistic-counter-based.

a) Exhaustive Methods: Exhaustive methods are exact and involve a distributed All-Pair Shortest Paths (APSP) computation.

APSP can be computed using the distributed Floyd–Warshall's shortest path algorithm [11] which runs in $O(n^2)$ time using $O(n^3)$ messages with $O(n)$ messages that carry $O(n)$ distances [12]. APSP can also be computed using BFSes. Performing a single BFS using Cheung's algorithm [13] takes $O(d)$ time, if we ignore message pileups, and uses $O(nm)$ messages [12]. All nodes can initiate a BFS traversal in parallel. However, the network may get congested, since messages will pileup, thus incurring a large time and memory overhead. On the other hand, BFSes can be performed one by one but it is expensive in terms of time. It uses in total $O(nd)$ time and $O(\Delta)$ space per node if message pileups are ignored. Also note that computing all the distances in parallel require the storage of $O(n)$ distances per node while, in sequential approaches, only the distance to the current-BFS root along with the partial eccentricity are stored per node and progressively updated.

As a consequence, existing distributed algorithms [15], [16], [17], which are designed to elect a node belonging to the exact center of arbitrary networks are not scalable. They involve a distributed APSP computation which has either a large time complexity or/and a large storage cost in asynchronous systems composed of thousands of nodes with constrained computational power and restricted memory resources.

b) Graph-Specific Methods: Efficient heuristics have, for instance, been proposed to compute the center of tree graphs (e.g., [18], [19], [20]). However these algorithms do not fir our system model. Although these approaches are efficient for the graphs they target, they are not directly generalizable to arbitrary graphs.

c) Sampling-based Methods: Some input-graph analysis approaches have recently been proposed in order to find a center vertex of arbitrary graphs using a sampling of SSSP computations. Most of them use (BFS) computations.

Existing approaches based on shortest paths computation from a sampling of nodes like [21], [22], [23], [7], [24], [25], [26], [27] are promising but they do not fit our system model. They have been designed for input-graph analysis or target synchronous distributed systems.

d) Probabilistic-Counter-based Methods: Algorithms based on low-memory-footprint probabilistic counters to estimate node centrality measures have recently been proposed in [30], [31], [32]. These algorithms are approximately equivalent to running a BFS from every node but at less expense in terms of computations and communications. These three algorithms are efficient but they do not fit our assumptions for the same reasons as for sampling-based methods.

e) Summary: Computing exact center nodes in asynchronous distributed systems is an expensive operation in terms of messages and in terms of storage requirement and/or time. Algorithms designed for a specific class of graphs (e.g., tree graphs) are not generalizable to arbitrary graphs. Efficient sampling-based and probabilistic-counter-based methods have been proposed but they have not been applied to distributed asynchronous systems so far. In this paper, we propose k-BFS SumSweep a sampling-based algorithm to overcome the limits of current algorithms in the context of LMRs.

IV. The k-BFS SumSweep Algorithm

A. Description

The k-BFS SumSweep algorithm, which selects the center node, is based on the SumSweep heuristic proposed as a starting point of the sequential algorithm in [7] to compute the exact graph diameter and radius. SumSweep aims at consecutively selecting the most external vertices of a graph.

Our distributed implementation of k-BFS SumSweep uses distributed BFSes to compute SSSP.

The main idea behind our algorithm is that central nodes are first and foremost central to the most external nodes. In our algorithm, a partial eccentricity value is computed for every node using distances to $\{u^\lambda\}_{1\leq\lambda\leq k} \subseteq V$, a subset of k nodes, with $k \leq n$. This subset is formed from a random initial vertex and $k-1$ external nodes selected in a consecutive manner using the SumSweep heuristic, i.e., the next node to be selected is the node of maximum partial farness that has not been previously selected, where the partial farness of a node is the sum of its distance to all the previously selected nodes.

k is a user parameter that has to be provided as input. In the evaluation section, we show that $k = 10$ provides accurate results even with large-scale systems of 10^4 nodes.

Figure 1 depicts an execution of the k-BFS SumSweep framework on a 200-node Blinky Blocks system with $k = 10$. The elected node matches the theoretical node.

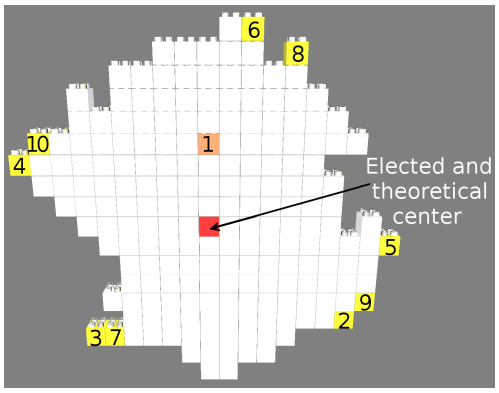

Fig. 1: The k-BFS SumSweep algorithm running on a 2D Blinky Blocks system composed of 200 modules, with $k = 10$. The initial module from which is performed the first distance computation is in brown. The other $k-1$ external nodes selected are in yellow and the order of selection is written on them. The module elected is in red and it matches the theoretical center.

B. Distributed Implementation

Algorithm 1 provides the pseudo-code of our distributed implementation of the k-BFS SumSweep algorithm.

a) Primitives and Message Types: Our implementation uses two BFS-based algorithmic primitives, namely INITIATOR_ELECTION_ALG and BFS_ALG, that have been defined in described in [9], under the names LE_CHEUNG-BFS-ST-CB_STB-STC and CHEUNG-BFS-ST-CB-AGG, respectively. These two algorithms are based on Cheung's algorithm for BFS computation [13]. INITIATOR_ELECTION_ALG enables to elect the minimum-identifier node as initiator and to build a path to a farthest node to it. BFS_ALG allows to perform a BFS and to computes network-wide aggregates. Moreover, our implementation uses also two specific types of messages (NEXT_BFS and ELECTED).

b) Pseudo-Code: During each iteration λ, a node u^λ is selected and the partial eccentricity value of every node is updated using the distance to u^λ. u^1 is elected using INITIATOR_ELECTION_ALG (lines 5 and 6). If both $k > 1$ and $n > 1$, then a NEXT_BFS message is sent toward u^2, a farthest node from that initiator (lines 9-10 and 22-34). Otherwise, u^1 is elected as the approximate center and k-BFS SumSweep terminates (line 12).

Every iteration $\lambda > 1$ starts when u^λ receives a NEXT_BFS message. Upon reception of that message, u^λ initiates a BFS_ALG (lines 9-10). BFS_ALG is used to compute the node distance to u^λ and to construct both a path to a candidate node of maximum partial farness and a path to a node of minimum partial eccentricity (lines 54-53. Upon termination of BFS_ALG, u^λ sends a NEXT_BFS message toward $u^{\lambda+1}$ in order to trigger a next iteration if $k > \lambda$ and $\lambda > n$ (line 21). Otherwise, u^λ elects the node of minimum partial centrality value. If u^λ has the minimum centrality value, k-BFS SumSweep terminates and u^λ is elected as the central node (line 17). Otherwise, u^λ sends an ELECTED messsage toward the node of minimum centrality value (lines 19 and 35-39). Upon reception of that message by the node of minimum eccentricity value, our algorithm terminates and this node is elected as the approximate-center node (line 37).

C. Termination Proof and Complexity Analysis

The k-BFS SumSweep algorithm sequentially runs $1\times$ INITIATOR_ELECTION_ALG, then $(k-1)\times$ BFS_ALG and finally forwards an ELECTED message toward the node of minimum centrality value through the last constructed spanning-tree. This message reaches its final destination using $O(d)$ time and $O(n)$ messages. All these steps terminate, thus our algorithm terminates. Moreover, we have $k \leq n$. Using the primitive complexity given in [9], the k-BFS SumSweep algorithm runs in $O(kd)$ time using $O(mn^2)$ messages and $O(\Delta)$ memory space per module.

V. Evaluation

This section presents our experimental evaluation performed both on hardware Blinky Blocks and in the VisibleSim simulator [33]. Through our experiments, we show the effectiveness, the efficiency and the scalability of our algorithms.

More precisely, we first show that k-BFS SumSweep works well on hardware through some examples. Then, we use VisibleSim to evaluate the performance of our algorithm in large-scale systems and to compare it to existing algorithms in terms of accuracy, execution time, number of messages and memory usage. As shown in [9], VisibleSim can be used to accurately benchmark the performance of algorithms on much bigger configurations.

A. Evaluation on Hardware

Figure 2 shows k-BFS SumSweep results on a line and a random configuration respectively composed of 50 and 63 hardware Blinky Blocks. For the two configurations,

Input : k // Number of BFSes to perform
Output : a single approximate-center node is elected
Primitive(s) : INITIATOR_ELECTION_ALG
BFS_ALG($handlers : handleBFSData, updateBFSAggs, getBFSAggs, resetBFSAggs$)

// Initialization and start handlers:
Initialization of v_i:
$candidate \leftarrow true$; $iteration \leftarrow 0$; $far \leftarrow 0$ $ecc \leftarrow 0$;
$branchEcc \leftarrow \{\}$; $branchFarCandidate \leftarrow \{\}$;
$nextHopToMinEcc \leftarrow \perp$; $nextHopToMaxFarCandidate \leftarrow \perp$;
start k-BFS SumSweep;

When k-BFS SumSweep **starts** at node v_i **do**:
start INITIATOR_ELECTION_ALG;

// Primitive termination handlers:
When INITIATOR_ELECTION_ALG **terminates** at root node v_i **do**:
$candidate \leftarrow false$;
$size \leftarrow$ INITIATOR_ELECTION_ALG.$size$;
if $size > 1$ *AND* $k > 1$ **then**
  send NEXT_BFS<$size$> **to** INITIATOR_ELECTION_ALG.$nextHopToFarthest$;
else
  k-BFS SumSweep **terminates**; // v_i is elected

When BFS_ALG **terminates** at root node v_i **do**:
$size \leftarrow$ BFS_ALG.$size$;
if $iteration + 1 = k$ *OR* $iteration + 1 = size$ **then**
  if $nextHopToMinEcc = \perp$ **then**
    k-BFS SumSweep **terminates**; // v_i is elected
  else
    send ELECTED<> **to** $nextHopToMinEcc$;
else
  send NEXT_BFS<$size$> **to** $nextHopToMaxFarCandidate$;

// k-BFS SumSweep message handlers:
When NEXT_BFS<$size$> message **is received** by the node v_i **do**:
$pathNextBFS = nextHopToMaxFarCandidate$;
if $iteration = 0$ **then** //
  $pathNextBFS \leftarrow$ INITIATOR_ELECTION_ALG.$nextHopToFarthest$;
if $pathNextBFS = \perp$ **then**
  $iteration \leftarrow iteration + 1$;
  $candidate \leftarrow false$; updateLocalValues();
  // Start a new BFS as root:
  re-initialize BFS_ALG;
  BFS_ALG.$size \leftarrow size$;
  BFS_ALG.$data[0] \leftarrow iteration$;
  start BFS_ALG;
else
  send NEXT_BFS<$size$> **to** $pathNextBFS$;

When ELECTED<> message **is received** by node v_i **do**:
if $nextHopMinEcc = \perp$ **then**
  k-BFS SumSweep **terminates**; // v_i is elected
else
  send ELECTED<> **to** $nextHopMinEcc$;

// Helper functions:
Function *updateLocalValues()* :
  $dist \leftarrow$ BFS_ALG.$distance$;
  if $iteration = 1$ **then**
    $dist \leftarrow$ INITIATOR_ELECTION_ALG.$distance$;
  $far \leftarrow far + dist$;
  $ecc \leftarrow \max\{ecc, dist\}$;

// Primitive handlers for aggregate
// computation and data propagation:
Function *handleBFSData()*:
  $iter \leftarrow$ BFS_ALG.$data[0]$;
  if $iter > iteration$ **then**
    $iteration \leftarrow iter$;
    updateLocalValues() ;
    // Take part in this BFS as non-root:
    re-initialize BFS_ALG;
Function *resetBFSAggs()*:
  $branchEcc \leftarrow \{\}$; $branchFarCandidate \leftarrow \{\}$;
Function *updateBFSAggs(v_j, child, aggs)* :
  if $child = true$ **then**
    $branchEcc[v_j] = aggs[1]$;
    $branchFarCandidate[v_j] = aggs[2]$;
  else
    remove $branchFarCandidate[v_j]$;
    remove $branchEcc[v_j]$;
Function *getBFSAggs()*:
  $dist \leftarrow$ BFS_ALG.$distance$;
  $maxCandidateFar \leftarrow 0$;
  $nextHopToMaxFarCandidate \leftarrow \perp$;
  if $candidate = true$ **then**
    $maxFar \leftarrow far + dist$;
  $v_f \leftarrow \text{argmax}_{v_k \in N^1_{v_i}} branchFarCandidate[v_k]$;
  if $v_f \neq \perp$ *AND* $branchFarCandidate[v_f] > maxFar$ **then**
    $maxCandidateFar \leftarrow branchFarCandidate[v_f]$;
    $nextHopToMaxFarCandidate \leftarrow v_f$;
  $minEcc \leftarrow \max\{minEcc, dist\}$;
  $nextHopToMinEcc \leftarrow \perp$;
  $v_f \leftarrow \text{argmax}_{v_k \in N^1_{v_i}} branchMinEcc[v_k]$;
  if $v_f \neq \perp$ *AND* $branchMinEcc[v_f] < minEcc$ **then**
    $minEcc \leftarrow branchMinEcc[v_f]$;
    $nextHopToMinEcc \leftarrow v_f$;
  return $<maxFar, minEcc>$;

Algorithm 1: Distributed implementation of the k-BFS SumSweep algorithm detailed for any node v_i.

the elected module belongs to the theoretical center of the network. Some other examples of execution of our algorithm can be seen online[4]. This video also shows that our algorithm is able to react to topology changes by reelecting an approximate-center module.

Table I gives the execution time of our algorithm on different configurations. This table shows that, in our experiments, the execution time of our algorithm is approximately linear in kd as announced in Section IV-C. The values for k were selected arbitrarily in order to provide a satisfactory accuracy at a reasonable cost. In line configurations, only 3 BFSes are necessary to find the theoretical center.

[4]https://youtu.be/qaIL85TPZQY

Shape	Size (module)	Diameter (hop)	k (BFSes)	Average execution time ± standard deviation (ms)	$\frac{\text{average execution time}}{k \times diameter}$
Line	5	4	3	267 ± 1	22
	10	9		650 ± 4	24
	50	49		3123 ± 2	21
Square	9	4	5	454 ± 8	23
	25	8		850 ± 19	21
	49	12		1237 ± 7	21
Cube	27	6	7	952 ± 21	23
Dumbbell	59	15	5	1513 ± 40	20
Random	63	19	10	3574 ± 26	19

TABLE I: Average execution time of k-BFS SumSweep on hardware Blinky Blocks. Statistics on the execution time were computed over 10 runs for every configuration.

a) Line (50 modules). k = 3

b) Random (63 modules). k = 10

Fig. 2: k-BFS SumSweep on hardware Blinky Blocks. We use the same color code as in Figure 1. The module elected (in red) matches the theoretical center.

B. Large-scale Evaluation and Comparison to Existing Algorithms

We use VisibleSim to execute an implementation of k-BFS SumSweep in a simulated physical environment and to compare it in terms of accuracy, execution time, number of messages and memory usage, on random large-scale Blinky Block systems, with existing solutions. Random systems were generated by connecting the modules one by one to the system at random, starting from a single node. Unless explicitly mentioned, every single point on the result plots represents 50 independent executions.

1) Compared Algorithms and Parameters: For the k-BFS SumSweep, we arbitrarily choose $k = 10$.

Our former algorithms: We consider ABC-Center [5] and PC2LE [9]. In our implementation of PC2LE, we use the same parameters as presented in [9].

n-BFS: We consider the exhaustive n-BFS algorithm presented under the name of BARYCENTER in [34]. This algorithm aims at electing a centroid node. It computes an APSP using n simultaneous asynchronous BFSes without acknowledgment. Even though this algorithm does not elect the center but the centroid, we decided to use it for cost comparisons because it is a simple method that shows the minimum cost induced by parallel asynchronous exhaustive methods. We use our own implementation of this approach. In our implementation, modules wait for 500 milliseconds after the reception of the last distance update triggered by a BFS message to check for convergence.

k-BFS-RAND: In [24], the author proposes to estimate node centrality using a partial value computed using distances to a random sample of nodes. We call k-BFS-RAND the algorithm to elect an approximate-center node using BFSes computation from a random sample of k nodes. We fix $k = 10$. In the k-BFS-RAND-SEQ, the BFSes are performed sequentially, while they are performed in parallel in k-BFS-RAND-PAR (more details on the implementation of these algorithms can be found in [9]).

2) Effectiveness Evaluation: In order to exhibit the accuracy of an algorithm, we use the relative center accuracy. We compute the exact center and node eccentricity using our tool[5] for external graph analysis.

Figure 3 shows the relative center accuracy of the different algorithms considered.

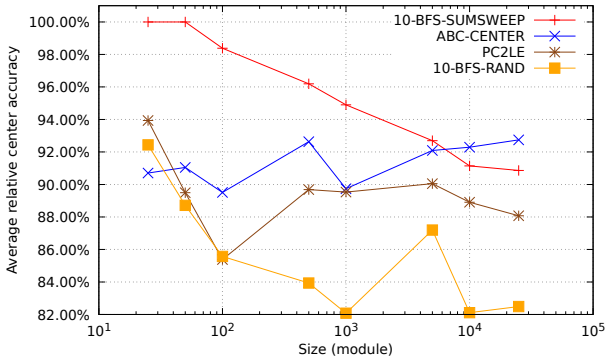

Fig. 3: Effectiveness of center leader algorithms : relative center accuracy versus the number of modules in the system.

We observe that ABC-Center and k-BFS SumSweep are more accurate than the other approximation algorithms. For systems, below 7,000 modules, k-BFS SumSweep is the most accurate method with an accuracy between 92% to 100%. However, for systems larger than 10,000, ABC-Center is slightly more precise than k-BFS SumSweep with an accuracy of 93% compared to 91%.

[5]GraphAnalyzer. Tool available online at: `https://github.com/nazandre/GraphAnalyzer`

Furthermore, we observe that performing BFSes from external nodes using the SumSweep heuristic (10-BFS-SUMSWEEP) leads to more accurate results than performing the BFSes from a random sample of nodes (10-BFS-RAND-CENTER). This is because the center is first and foremost central to most external nodes. When selecting random node, 10-BFS-RAND-CENTER may perform BFSes from nodes close to the center which does not improve the precision of the current eccentricity estimation.

3) Efficiency Evaluation: In this section, we study the time efficiency, the communication efficiency and the memory usage of the different algorithms.

a) Simulated Execution Time: To measure the execution time, we consider that an algorithm terminates when the node to be elected considers itself elected.

Figure 4 shows that the simulated average execution time of the considered algorithms. All the algorithms, except for n-BFS, seem to increase linearly with the diameter of the system. The average execution time of n-BFS explodes in systems with more than 1,000 modules which is due to network congestion.

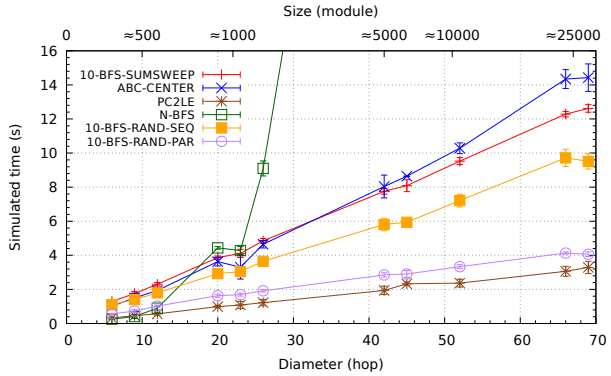

Fig. 4: Simulated average execution duration ($\pm$ standard deviation) versus the system diameter. For each point, at least 5 executions were performed.

ABC-Center and k-BFS SumSweep are longer to converge than the other algorithms, except for n-BFS. Nevertheless, as previously shown, these algorithms are definitely more precise than all the other approximation algorithms.

PC2LE and k-BFS-RAND-PAR scale well in terms of execution time. For Blinky Blocks systems with a diameter of more than 65 hops and a size of approximately 25,000 modules, PC2LE elects an approximate-center module in less than 4 seconds.

b) Number of Messages: Figure 5 shows the average number of messages per module during the execution of the algorithms according to the size of the system. The number of messages used by an algorithm includes all the messages that it generates, even those sent after the final node has been elected. The number of messages sent reflects the energy consumption of the modules.

We observe that n-BFS uses a lot more messages than the other algorithms. For large-scale systems with 25,000 Blinky Blocks, PC2LE uses about 700 messages per module while

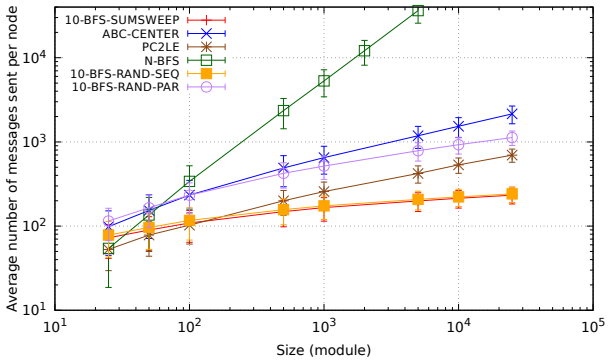

Fig. 5: Average number of messages ($\pm$ standard deviation) send per module according to the size of the system.

10-BFS-SumSweep and 10-BFS-RAND-SEQ use about 250 messages per node.

c) Memory Usage: Figure 6 shows the maximum memory usage of the different algorithms. The memory usage of an algorithm is composed of its memory footprint, both at the application level and in the different message queues.

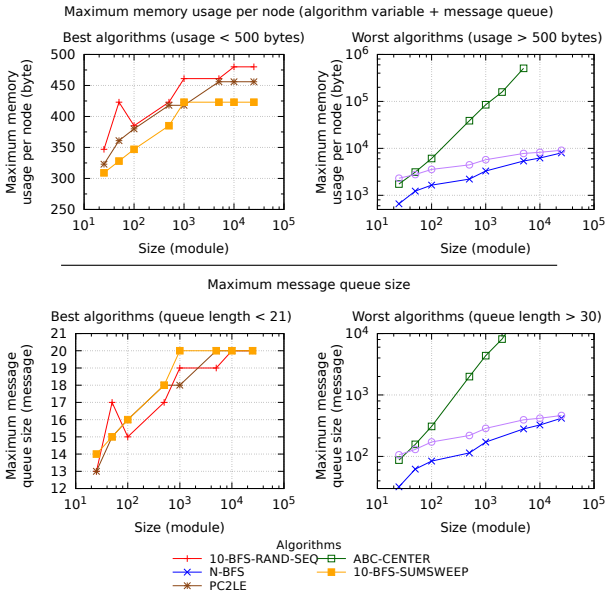

Fig. 6: Above, the maximum memory usage (considering both the local algorithm variables and the message queue usage) according to the size of the system. Below, the maximum message queue per module (considering both the incoming and outgoing queues).

We recall that, in n-BFS, every node locally stores $O(n)$ information at the application level and PC2LE stores $O(c+\Delta)$, where c is the cost of the probabilistic counter used [9]. The other algorithms store $O(\Delta)$ information.

PC2LE, k-BFS-SumSweep and k-BFS-SEQ scale well in terms of memory usage. In systems with 25,000 nodes, they use less than 500 bytes of memory, among which 380

bytes[6] are due to message queue occupancy. ABC-Center and 10-BFS-RAND use up to 10 kbytes in systems with 25,000 modules because of the memory overhead due to message pileups. 10-BFS-RAND perform BFSes in parallel, thus being faster but requiring much more memory. n-BFS uses 600 kbytes in systems with 5,000 modules.

VI. Discussion

Electing a central node involves a trade-off between the cost that can be afforded in terms of resources (time, memory, computation, energy) and the desired level of accuracy. Thus the algorithm to be used in order to elect a central node depends on the application, i.e., the role that this central node will play, the stability of the network, the scarcest resource, etc.

Exact approaches (e.g., n-BFS) are exhaustive and tend to overwhelm the network. They are definitely not suitable for large-scale systems since they are slow to converge, they generate a significant number of messages and may have a large memory footprint. It is paradoxical, since the importance of central nodes increases with the system size. In 5,000 node systems, n-BFS requires nearly 45 seconds to converge and uses more than 500 kbytes per node.

k-BFS-SumSweep is the most accurate center approximation algorithms for system smaller than 7,000 modules. It performs BFSes from $k \leq n$ nodes, which uses less resources than performing n-BFS. Moreover, k-BFS-SumSweep uses external roots and thus is more accurate than k-BFS-RAND in which BFSes are run from random nodes.

k-BFS-SumSweep and ABC-Center are, however, slower to converge as the BFSes are performed consecutively. In 25,000-module systems, they run in almost 13 seconds. BFSes cannot be parallelized in these two algorithms, but if it was possible, naively performing BFSes in parallel would overwhelm the network and may incur a large memory overhead. PC2LE is the fastest algorithm but it is less precise and uses more messages.

k-BFS SumSweep and PC2LE have a limited memory cost. They use between 400 and 480 bytes per node max whereas n-BFS, 10-BFS-RAND-PAR and ABC-Center use between 8 kbytes and almost 600 kbytes which is definitely not suitable for modular robotic systems with scarce memory resources.

VII. Conclusions

In this paper, we proposed the k-BFS SumSweep algorithm to elect an approximate-center node in LMRs. Our algorithm runs in $O(kd)$ time using $O(mn^2)$ messages and $O(\Delta)$ memory space per module.

We evaluated our algorithm both on hardware modular robots and in a simulator for large ensemble of robots. Results show our algorithm is a good trade-off if accuracy is the main concern. k-BFS SumSweep is the most precise approximation algorithm for systems composed of less than 7,000 modules. In these systems, our algorithm exhibits a relative accuracy between 92% to 100%. Moreover the k-BFS SumSweep algorithm is reasonably efficient in terms of time and communication, and has a limited memory footprint.

In future work, it will be interesting to carry out a formal analysis of the accuracy of our algorithm in order to derive bounds. Moreover, we want to find a way to dynamically infer a good value for k in a given system. In addition, we plan to study the problems of approximate-center node election in networks that exhibit a high degree of dynamics due to nodes failure and/or mobility. Currently, our algorithm restarts computations from scratch upon neighbor change detection. This mechanism will be too expensive in terms of resource usage in highly dynamic networks.

[6] $20 \times 19 = 380$ bytes

Acknowledgment

This work has been funded by ANR (ANR-16-CE33-0022-02), the French Investissements d'Avenir program, ISITE-BFC project (ANR-15-IDEX-03), Labex ACTION program (ANR-11-LABX-01-01) and Mobilitech project.

References

[1] J. Bourgeois, B. Piranda, A. Naz, H. Lakhlef, N. Boillot, H. Mabed, D. Douthaut, and T. Tucci, "Programmable matter as a cyber-physical conjugation," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. Budapest, Hungary: IEEE, October 2016.

[2] J. Bourgeois and S. Goldstein, "Distributed intelligent mems: Progresses and perspectives," *ICT Innovations 2011*, pp. 15–25, 2012.

[3] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, "A time synchronization protocol for large-scale distributed embedded systems with low-precision clocks and neighbor-to-neighbor communications," *Journal of Network and Computer Applications*, vol. 105, pp. 123 – 142, 2018.

[4] O. B. Bayazit, J.-M. Lien, and N. M. Amato, "Better group behaviors in complex environments using global roadmaps," *Artificial life*, vol. 8, p. 362, 2003.

[5] A. Naz, B. Piranda, S. C. Goldstein, and J. Bourgeois, "ABC-Center: Approximate-center election in modular robots," in *IROS 2015, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Hamburg, Germany, Oct. 2015, pp. 2951–2957.

[6] ——, "Approximate-centroid election in large-scale distributed embedded systems," in *AINA 2016, 30th IEEE Int. Conf. on Advanced Information Networking and Applications*. Crans-Montana, Switzerland: IEEE, Mar. 2016, pp. 548–556.

[7] 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.

[8] 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.

[9] A. Naz, "Distributed algorithms for large-scale robotic ensembles: Centrality, synchronization and self-reconfiguration," Ph.D. dissertation, FEMTO-ST Institute, Univ. Bourgogne Franche-Comt, CNRS, 2017.

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

[11] S. Toueg, "An all-pairs shortest paths distributed algorithm," IBM, Tech. Rep., 1980.

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

[13] T.-Y. Cheung, "Graph traversal techniques and the maximum flow problem in distributed computation," *Software Engineering, IEEE Transactions on*, vol. SE-9, no. 4, pp. 504–512, 1983.

[14] S. Holzer and R. Wattenhofer, "Optimal distributed all pairs shortest paths and applications," in *Proceedings of the 2012 ACM symposium on Principles of distributed computing*. ACM, 2012, pp. 355–364.

[15] 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.

[16] K. A. Lehmann and M. Kaufmann, "Decentralized algorithms for evaluating centrality in complex networks," 2003.

[17] P. S. Almeida, C. Baquero, and A. Cunha, "Fast distributed computation of distances in networks," in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012, pp. 5215–5220.

[18] 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.

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

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

[21] 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.

[22] J. Shun, "An evaluation of parallel eccentricity estimation algorithms on undirected real-world graphs," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1095–1104.

[23] F. W. Takes and W. A. Kosters, "Computing the eccentricity distribution of large graphs," *Algorithms*, vol. 6, no. 1, pp. 100–118, 2013.

[24] D. Eppstein and J. Wang, "Fast approximation of centrality," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 228–229.

[25] B. Dissler, S. Holzer, and R. Wattenhofer, "Distributed local multi-aggregation and centrality approximation," *arXiv preprint arXiv:1605.06882*, 2016.

[26] L. Roditty and V. Vassilevska Williams, "Fast approximation algorithms for the diameter and radius of sparse graphs," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM, 2013, pp. 515–524.

[27] S. Chechik, D. H. Larkin, L. Roditty, G. Schoenebeck, R. E. Tarjan, and V. V. Williams, "Better approximation algorithms for the graph diameter," in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 1041–1052.

[28] K. Wehmuth and A. Ziviani, "Daccer: Distributed assessment of the closeness centrality ranking in complex networks," *Computer Networks*, vol. 57, no. 13, pp. 2536–2548, 2013.

[29] A. Naz, B. Piranda, T. Tucci, S. C. Goldstein, and J. Bourgeois, "Network characterization of lattice-based modular robots with neighbor-to-neighbor," in *2016 13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*. London, UK: Springer, November 2016.

[30] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec, "Hadi: Mining radii of large graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, p. 8, 2011.

[31] U. Kang, S. Papadimitriou, J. Sun, and H. Tong, "Centralities in large networks: Algorithms and observations." in *SDM*, vol. 2011. SIAM, 2011, pp. 119–130.

[32] F. Garin, D. Varagnolo, and K. H. Johansson, "Distributed estimation of diameter, radius and eccentricities in anonymous networks," *IFAC Proceedings Volumes*, vol. 45, no. 26, pp. 13–18, 2012.

[33] D. Dhoutaut, B. Piranda, and J. Bourgeois, "Efficient simulation of distributed sensing and control environments," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 452–459.

[34] M. Mamei, M. Vasirani, and F. Zambonelli, "Self-organizing spatial shapes in mobile particles: The tota approach," in *Engineering Self-Organising Systems*. Springer, 2005, pp. 138–153.

[35] C. Kim and M. Wu, "Leader election on tree-based centrality in ad hoc networks," *Telecommunication Systems*, vol. 52, no. 2, pp. 661–670, 2013.