Robust Parallel Redeployment Algorithm for MEMS Microrobots

Hicham Lakhlef, Julien Bourgeois and Hakim Mabed
FEMTO-ST/DISC, University of Franche-Comte, 1 Cours Leprince Ringuet, 25201, Montbéliard, France
$\{hlakhlef, julien.bourgeois, hmabed\}$ $@femto-st.fr$

***Abstract*—In this paper we propose a distributed and robust parallel redeployment algorithm for MEMS microrobots. MEMS microrobots are low-power and low-memory capacity devices that can sense and act. To deal with the MEMS microrobots characteristics, in this paper, we present an efficient redeployment algorithm without predefined positions of the target shape, which reduces the memory usage to a constant complexity. This algorithm optimizes the energy consumption by minimizing the amount of displacement and the number of messages. This solution improves the memory usage (number of states), the execution time and the number of movements by using movement of different microrobots at the same time. In addition, we show how to predict the number of movement for each node to make the algorithm robust.**

***Index Terms*—MEMS; Distributed Algorithms; Parallel Algorithms; Redeployment; Self-reconfiguration; Optimization; Logical topology;**

I. Introduction

Micro electro mechanical systems (MEMS) are miniaturized and low-power devices that can sense and act. It is expected that these small devices, referred to as MEMS nodes, will be mass-produced, making their production cost almost negligible. Their applications require a massive deployment of nodes, thousands or even millions [7] which will give birth to the concept of Distributed Intelligent MEMS (DiMEMS) [3]. A DiMEMS device is composed of typically thousands or even millions of MEMS nodes. Some DiMEMS devices are composed of mobile MEMS nodes [1], some others are partially mobile [28] whereas other are not mobile at all [3].
At the present time, swarm robotics is gaining increasing attention since large-scale swarms of microrobots can perform various missions and tasks in a wide range of applications including odor localization, firefighting, medical service, surveillance and security, and search and rescue [11], [23], the redeployment for MEMS microrobots is necessary to do these tasks. One of the major challenges in developing a microrobot is to achieve a precise movement to reach the destination position while using a very limited power supply. Many different solutions have been studied for example, within the $Claytronics$ project [1], [2], [8], [12], [20] each microrobot helps its neighbor to move to the desired position, which introduces the idea of a collaborative way of moving. But, even if the power requested for moving has been lowered, it still costs a lot regarding the communication and computation requirements. Optimizing the number of movements of microrobots is therefore crucial in order to save energy.
In the literature, the redeployment (or self-reconfiguration) can be seen from two different points of view. First, it can be defined as a protocol, centralized or distributed, which transforms a set of nodes to reach the optimal logical topology from a physical topology [10]. On the other hand, the redeployment is built from modules which are autonomously able to change the way they are connected, thus changing the overall shape of the network [8], [24]. This process is difficult to control, because it involves the distributed coordination of a large numbers of identical modules connected in time-varying ways. This work takes place within the Claytronics project and aims at optimizing the logical topology of the network through rearrangement of the physical topology as we will see in the next sections.

II. Related Works

Many terms refer to the concept of redeployment. In several works on wireless networks the term used is $self-organization$, this term is also used to express the clustering of ad-hoc networks. Also, the self-organization term can be found in protocols for sensors networks to form a sphere or a polygon from a center node [19], [26]. Others algorithms for the redeployment of sensor networks in [14].
A growing number of research on self-reconfiguration for microrobots using centralized algorithms have been done, among them we find centralized self-assembly algorithms [22]. Other approaches give each node a unique ID and a predefined position in the final structure; see for instance [25]. The drawback of these methods is the centralized paradigm and the need for nodes identification. More distributed approaches include [5], [9], [13]. The authors in [4] have shown how a simulated modular robot (Proteo) can self-configure into useful and emergent morphologies when the individual modules use local sensing and local control rules.
Claytronics, is the name of a project led by Carnegie Mellon University and Intel corporation. Many works have already been done within the Claytronics project. In [6], [8] the authors propose a metamodel for the reconfiguration of catoms starting from an initial configuration to achieve a desired configuration using $creation$ and $destruction$ primitives. The authors use these two functions to simplify the movement of each catom. Another scalable algorithm can be found in [20]. In [2], a scalable protocol for Catoms self-reconfiguration is proposed, written with the MELD language [1], [21] and using the creation and destruction primitives. In all these works, the

authors assume that all Catoms know the correct positions composing the target shape at the beginning of the algorithm and each node is aware of its current position. The first self-reconfiguration without predefined positions of the target shape appears in [15], [18]. However, this solution is not parallelized. In [17] another solution that guarantees the connectivity of the network through the execution time of the algorithm, this solution is not robust.

III. Contributions

In this paper, we propose a new distributed approach for parallelized redeployment of MEMS microrobots, where the target form is built in parallel incrementally, and each node can predict its number of movements to make the algorithm robust, because the node can make sure that it has correctly followed the algorithm. Contrary to existing works, in our algorithm each node has no information on the correct positions (predefined positions) of the target shape, and movement of microrobots is fully implemented. The redeployment with shared map (predefined positions of the target shape) does not scale. Because with the map each node should store all predefined positions (may be millions) of the target shape, therefore this is not always possible as MEMS nodes have a low-memory capacity.

We propose an efficient, distributed and parallelized algorithm for nodes redeployment where each node can communicate only with its physical neighbors. The performance of the self-organization algorithm is evaluated according to the number of movements, the amount of memory used and the time taken. In this paper the MEMS network is organized initially as a chain. By choosing a straight chain as the initial shape, we aim to study the performance of our approach in extreme case. Indeed, the chain form represents the worst physical topology for many distributed algorithms in terms of fault tolerance, propagation procedures and convergence. First, the number of direct contacts between microrobots is minimal and secondly the average distance between two robots (in term of number of hops) is of $(n+1)/3$ where n is the number of microrobots. Also, a chain of microrobots represents the worst case for message broadcasting complexity with $O(n)$, after reconfiguring into a square the complexity will be $O(\sqrt{n})$ in the worst case.

To assess the distributed algorithm performance, we present the simulation results and we compare to former results.

Outline of the paper. The rest of the paper is organized as follows: Section 4 discusses the model, definitions and some tools. Section 5 discusses the proposed algorithm, it shows how to predict the number of movements and shows the generalization of the algorithm. Section 6 details the simulation results. Finally, section 7 summarizes our conclusions and illustrates our suggestions for future work.

IV. Model, definitions and tools

Within Claytronics, a Catom (figure 1) that we call in this paper a node is modeled as a sphere which can have at most six neighbors without overlapping. Within Claytronics, each node

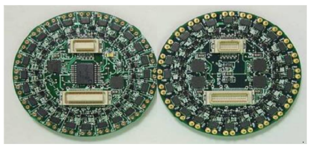
Figure 1. Two catoms

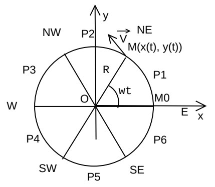

Figure 2. Node modeling, in each movement the node travels the same distance

is able to sense the direction of its physical neighbors (east (E), west (W), north-east (NE), south-east (SE), south-west (SW) and north-west (NW)). The starting physical topology is a chain of n nodes linked together. We will take the example of nodes that have neighbors in NW and SE directions and we will show after how to generalize. A node A is in neighbor's list of node B if A touches physically B (figure 1). Communications are only possible through contact, which means that only neighbors can have a direct communication. Therefore, if the node didn't have a neighbor at the previous round it cannot join the other nodes of the network, so it is lost. Because it does not know where is the group, or at least one node connected to the group. Thus, redeployment algorithms have to make sure that no node will be lost.

Consider the connected undirected graph *G = (V, E)* modeling the network, where $v \in V$, is a node that belongs to the network and, $e \in E$ a bidirectional edge of communication between two physical neighbors. For each node $v \in V$, we denote the set of neighbors of v as $N(v)$. We define the following terminology:

$Snap-Connectivity$: there is a snap-connectivity within a dynamic graph if there is always a path from every node to every node.

$Non-Snap-Connectivity$: there is a non-snap-connectivity if the graph that models the network is connected only at the end of the algorithm.

We call the *highest number of movements* the highest number of movements was performed by a node belongs to the network.

To calculate the highest number of movements we define the following:

Consider the figure 2, we say that a microrobot has done a single movement if the distance between its former position and its new position is exactly twice the radius $D = 2R$. In each movement the node travels Ra (with $a = 60°$) from $m0$ to m. More details in [15].

Theorem 4.1: [1] Let y be an odd\even square number (y is an integer that is the square of an integer), then the next odd\even square number is $y + 4\sqrt{y} + 4$

Theorem 4.2: Let y be a square number (y is an integer that is the square of an integer), then if y is odd\even the next even\odd square number is $y + 2\sqrt{y} + 1$

[1]The character "\" means respectively (resp.) in lemmas and theorems

V. Proposed Protocol

A. *Parallel Algorithm with Unsafe Connectivity (PAUC)*

In this section we present our protocol that ensures the property of non-snap-connectivity.

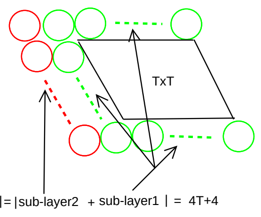

Figure 3. Represents how many nodes added to reach the next square when n is odd

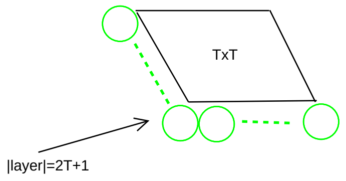

Figure 4. Represents how many nodes added in the last layer to reach the last square when n is even

To form the matrix of the square with $\sqrt{N} \times \sqrt{N}$ nodes, we begin (according to *Theorem 4.1* and *Theorem 4.2*) with an incremental process with a correct square (for example 1x1). Then, we add each time a new sub-layer contains $3T+2$ nodes, with $T \times T$ is the last square. After, we add another sub-layer with $T+2$ nodes taking positions at the W direction relative to nodes of the last shape. If N is even, at the last layer we add $2T+1$ nodes, with $T \times T$ is the last square, figures 3 and 4 show an example. The choice of the middle node depends on the optimality of parallelism. To apply an optimal parallelism the middle node mi which is also the initiator of the algorithm should be found with the followings:
Let N be the network size, $n = \lfloor \sqrt{N} \rfloor \lfloor \sqrt{N} \rfloor$ and $dif = N - n$ then:

- If n is odd and $dif < 2\sqrt{n}$ then the middle node will be $mi = (n+1)/2$
- If n is odd and $dif \geq 2\sqrt{n}$ then the middle node will be $mi = ((n+1)/2) + \sqrt{n} - 2$
- If n is even and $dif < 2\sqrt{n}$ then the middle node will be $mi = n/2 - ((\sqrt{n}/2) - 1)$
- If n is even and $dif \geq 2\sqrt{n}$ then the middle node will be $mi = (n/2 - ((\sqrt{n}/2) - 1)) + \sqrt{n} - 2$

The middle node mi can be found by knowing the network size, an end node of the chain initializes a counter and broadcasts it, each node receives this message increments the counter until its arrives to the concerned node mi, that will have the satisfied predicate $medChain(v)$.

Variables and Predicates

- initiator$_v$(): node v that initializes the algorithm.
- state$_v(X)$: v takes the state $X \in \{well, bad, int, nper, sint, rint, mnper, top, bottom\}$, v cannot take the states $well$ and bad in the same time.
- moveAround$state_v(u, P_x)$: move around neighbor u that has the state $state$ in such a way u becomes v's neighbor in the direction x relative to v.
- moveTo$_v(P_{N_x})$: move to the old position of the former neighbor at direction x relative to v.

B. *Description and analysis*

The algorithm PAUC (presented here after) runs in rounds, in each round satisfied predicates are chosen to run. The distributed algorithm seeks the desired form by using an incrementally process. In a completed increment, the nodes that build it belong already to the form; these nodes will help neighbor or future neighbor nodes to get correct positions.
The middle node (mi) of the chain declares itself as an initiator with the predicate (1). Initially, all nodes are initialized with the state bad except the initiator (2), the initiator takes the states $well$ and $nper$ with (3) and (4). Nodes that are above the initiator take the state top with predicate (10), the other nodes that are under the initiator take the state $bottom$ (11). Nodes having the state $well$ or int are nodes already in the target shape and cannot move, they became steady.
To make an optimal parallelism and correct square, the number of nodes having the state top (10) to be in the same line as the initiator (in the E direction relative to the initiator) must be equal to the number of nodes having state $bottom$ (11) to be in the same line as the initiator if N is odd. If N is even, another node is added to the nodes having state top. The state $nper$ is used to achieve this purpose. That is, the node having the state $nper$ does not permit to some nodes to move around it. The initiator takes the state $nper$ (4), by taking this state the initiator and each node has this state does not allow to neighbor nodes having the state $bottom$ to move around it in order to join the line of the initiator (to became in the E direction relative to the initiator). This is done with guard $(\neg state_v(nper))$. The state $mnper$ is an intermediate state used to propagate the state $nper$ to the other concerned nodes, that will keep the parallelism optimal. The state $sint$ (12) is another intermediate state used as a reference to the first node that can get the state $nper$, the state $sint$ is an indispensable state because the second node that should take the state $nper$ (13) is not a neighbor node of the initiator which is the first node that takes the state $nper$. The node that has a neighbor in the E direction having the state $nper$ takes the state $mnper$ (15). The node that has the initiator as neighbor node in the E direction takes the state $sint$ (12). The other (next) nodes that will take the state $nper$ are nodes having in the E direction a neighbor that has the state $mnper$ (16). Therefore, the node having the state $nper$ does not allow neighbor nodes having the state $bottom$ to join the line of the initiator, as these nodes are checking the predicates (30), (32), (35) and (36).
The state int is an intermediate state used to add a non-complete layer to the square shape. Thus, the nodes that have neighbors having the state $well$ take the state int with predicate (17). The first node that changes its state to int is the one in the line of the initiator. After, the state int is propagated to nodes that have neighbors having the $well$ state. Notice that, nodes take the state int if have at least one neighbor having the state $well$, but when making a new layer there is one node that will not have a neighbor having the state $well$ and it should take the state int, the state $rint$ (14) is used to deal with this case.

Predicates checked only in the first round

1) initiator$_v() \equiv medChain(v)$.
2) state$_v(bad) \equiv connected_v() \wedge \neg initiator_v()$.
3) state$_v(well) \equiv initiator_v()$.
4) state$_v(nper) \equiv initiator_v()$.

Predicates checked in each round

5) thisRound$\equiv GetCurrentRound()$.
6) hasN$_{nw}v(thisRound) \equiv (N_{nw}(v) = u) \wedge state_u(bad)$.
7) N$_{nw}$lastRound$_v(LastRound) \equiv hasN_{nw}v(thisRound - 1)$.
8) hasN$_{se}v(thisRound) \equiv (N_{se}(v) = u) \wedge state_u(bad)$.
9) N$_{se}$lastRound$_v(LastRound) \equiv hasN_{se}v(thisRound - 1)$.
10) state$_v(top) \equiv (N_{se}(v) = u, initiator_u()) \vee (N_{se}(v) = u, state_u(top))$.
11) state$_v(bottom) \equiv (N_{nw}(v) = u, initiator_u()) \vee (N_{nw}(v) = u, state_u(top))$.
12) state$_v(sint) \equiv (N_e(v) = u, initiator_u())$.
13) state$_v(nper) \equiv (N_{se}(v) = u, state_u(sint))$.
14) state$_v(rint) \equiv (N_e(v) = u, state_u(well)) \wedge (N_{ne}(v) = u, \neg state_u(well), \neg state_u(int))$.
15) state$_v(mnper) \equiv (N_e(v) = u, state_u(nper)) \wedge (\neg state_v(nper)) \wedge (state_v(int) \vee state_v(well))$.
16) state$_v(nper) \equiv (N_e(v) = u, state_u(mnper)) \wedge (\neg state_v(mnper)) \wedge (N_{se}(v))$.
17) state$_v(int) \equiv ((N_e(v) = u, state_u(well)) \wedge (N_{nw}(v))) \vee ((N_{se}(v) = u, state_u(well)) \wedge (N_w(v))) \vee ((N_{se}(v) = u, state_u(rint))) \vee ((N_{ne}(v) = u, state_u(well)) \wedge (N_{nw}(v))) \vee ((N_{ne}(v) = u, state_u(well)) \wedge (N_w(v)))$.
18) state$_v(int) \equiv ((N_e(v) = u, state_u(well)) \wedge (N_{nw}(v))) \vee ((N_{se}(v) = u, state_u(well)) \wedge (N_w(v))) \vee (N_e(v) = u1, N_{se}(v) = u2, state_{u1}(int), state_{u2}(int)) \vee ((N_{ne}(v) = u, state_u(well)) \wedge (N_{nw}(v))) \vee ((N_{ne}(v) = u, state_u(well)) \wedge (N_w(v)))$.
19) state$_v(well) \equiv ((N_e(v) = u, state_u(int)) \wedge (N_{nw}(v))) \vee ((N_e(v) = u, state_u(int)) \wedge (N_{se}(v)))$.
20) state$_v(well) \equiv (N_w(v) = u, state_u(well)))$.
21) state$_v(well) \equiv state_v(bad) \wedge (\neg N_{se}(v)) \wedge (N_w) \wedge (N_{nw}(v) = u, state_u(well))$.
22) state$_v(well) \equiv state_v(bad) \wedge (\neg N_{se}(v)) \wedge (N_w) \wedge (N_{nw}(v) = u, state_u(well)) \wedge ((N_e(v) = u1, state_{u1}(well)))$.
23) moveTo$_v(P_{N_{nw}}) \equiv state_v(top) \wedge state_v(bad) \wedge N_{nw}lastRound_v(LastRound) \wedge \neg hasN_{nw_v}(thisRound)$.
24) moveTo$_v(P_{N_{se}}) \equiv state_v(bottom) \wedge state_v(top) \wedge state_v(bad) \wedge N_{se}lastRound_v(LastRound) \wedge \neg hasN_{se_v}(thisRound)$.
25) moveAround$int_v(u, P_{se}) \equiv (\neg N_w(v) \wedge state_v(top) \wedge \neg N_{nw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{sw}(v) = u, state_u(int), state_u(top)) \wedge (\neg state_u(nper)) \wedge (((N_{se}(v) = u1, \neg N_e(u1) = u) \wedge N_{nw}(v)) \vee (N_{se}(v) = u1, N_e(u1) = u))$.
26) moveAround$well_v(u, P_{se}) \equiv (\neg N_w(v) \wedge state_v(top) \wedge \neg N_{nw}(v)) \wedge state_v(top) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{sw}(v) = u, state_u(well), state_u(top))$.
27) moveAround$int_v(u, P_e) \equiv (\neg N_w(v) \wedge state_v(top) \wedge \neg N_{nw}(v) \wedge \neg N_e(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{se}(v) = u, state_u(int), state_u(top)) \wedge (\neg state_u(nper))$.
28) moveAround$well_v(u, P_e) \equiv (\neg N_w(v) \wedge state_v(top) \wedge \neg N_{nw}(v) \wedge \neg N_e(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{se}(v) = u, state_u(well), state_u(top))$.
29) moveAround$well_v(u, P_{ne}) \equiv (\neg N_w(v) \wedge state_v(top) \wedge \neg N_{sw}(v) \wedge \neg N_{se}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_e(v) = u, state_u(well), state_u(top)) \wedge (\neg state_u(nper))$.
30) moveAround$int_v(u, P_{ne}) \equiv (\neg N_w(v) \wedge state_v(top) \wedge \neg N_{sw}(v) \wedge \neg N_{se}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_e(v) = u, state_u(int), state_u(top)) \wedge (\neg state_u(nper))$.
31) moveAround$well_v(u, P_{ne}) \equiv (\neg N_e(v) \wedge state_v(bottom) \wedge \neg N_{sw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{nw}(v) = u, state_u(well), state_u(bottom))$.
32) moveAround$well_v(u, P_{se}) \equiv (\neg N_w(v) \wedge state_v(bottom) \wedge \neg N_{sw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_e(v) = u, state_u(well), state_u(bottom), (\neg state_u(nper))$.
33) moveAround$well_v(u, P_e) \equiv (\neg N_w(v) \wedge state_v(bottom) \wedge \neg N_{sw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{ne}(v) = u, state_u(well), (\neg state_u(nper))$.
34) moveAround$int_v(u, P_{ne}) \equiv (\neg N_w(v) \wedge state_v(bottom) \wedge \neg N_{sw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{nw}(v) = u, state_u(int), state_u(bottom)) \wedge (N_{ne}(v) \vee N_{se}(v))$.
35) moveAround$int_v(u, P_e) \equiv (\neg N_w(v) \wedge state_v(bottom) \wedge \neg N_{sw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge (N_{ne}(v) = u, state_u(int), state_u(bottom), (\neg state_u(nper))$.
36) moveAround$well_v(u, P_e) \equiv (\neg N_w(v) \wedge state_v(bottom) \wedge \neg N_{sw}(v)) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad) \wedge state_v(bottom) \wedge (N_{ne}(v) = u, state_u(well), state_u(mnper), (\neg state_u(nper))$.

PAUC

Notice that, nodes with the state $well$ and nodes with state int together do not composites a square, it well be a square if all nodes having the state int have in the W direction a neighbor node, this neighbor node has the state $well$. Therefore, the wave of state changing to $well$ begins with predicates (19), (20), (21) and (22).

With the predicate (23) the nodes having the states top and bad descend towards the center of the chain. As well as, nodes having the state $bottom$ and bad mount towards the center of the chain with the predicates (24). That is, with predicate (23), the node that has the state top and bad takes the position of its former neighbor in the NW direction. With predicate (23) the node having the state $bottom$ and bad takes the position of its former neighbor in the SE direction (24). With predicates (6), (7) (8) and (9), the node cheeks if it had a neighbor in the SE or NW direction in the previous round.

With predicate (25)/(26), the node v that has the states top and bad moves around a node u having the states top and $int/well$, node u becomes a neighbor in SE direction relative to v. With predicate (27)/(28) the node v that has the states top and bad moves around a node u having the states top and $int/well$, node u becomes a neighbor in E direction relative to v. With predicate (29)/(30), the node v that has the states top and bad moves around a node u having the states $bottom$ and $well/int$, node u becomes a neighbor in NE direction relative to v. With predicate (31)/(32), the node v that has the states $bottom$ and bad moves around a node u having the states $bottom$ and $well/int$, node u becomes a neighbor in NE/SE direction relative to v. With predicate (33)/(34) the node v that has the states $bottom$ and bad moves around a node u having the states $bottom$ and $well/int$, node u becomes a neighbor in E/NE direction relative to v. And with (35)/(36), the node v that has the states $bottom$ and bad moves around a node u having the states $bottom$ and $int/well$, node u becomes a neighbor in E direction relative to v.

Theorem 5.1: If N is the network size, $n = \left\lfloor\sqrt{N}\right\rfloor \left\lfloor\sqrt{N}\right\rfloor$ and $\eta = \left\lceil\sqrt{N}\right\rceil \left\lceil\sqrt{N}\right\rceil$, then:

- if n is odd and $n = N$ then the highest number of movements will be $((n + \sqrt{n} - 2)/2)$
- if n is odd and $n < N$ then the highest number of movements will be $(\eta/2 - 1)$
- if n is even and $n = N$ then the highest number of movements will be $(n/2 - 1)$
- if n is even and $n < N$ then the highest number of movements will be $((\eta + \sqrt{\eta} - 2)/2)$

Example: Figure5 shows an example.

C. Predicting the number of movements for each node

In this section, we present how to make the algorithm energy-aware and robust by predicting the number of movements for each node. So each node knows the amount of energy that will consume. And, the node, by this predicting can make sure that is has correctly executed the protocol.

To predict the number of movements for each node we take a partitioning of nodes into two groups (A) and (B) :

- The size of the group (A) is $|(A)| = \frac{n+1}{2}$ if n is odd. Or $|(A)| = (n/2 - ((\sqrt{n}/2) - 1))$ if n is even.
- The size of the group (B) is $|(B)| = \frac{n-1}{2}$ if n is odd. Or $|(B)| = (n/2 + ((\sqrt{n}/2) - 1))$ if $n = N$ and n is even.

To apply the prediction functions, we partition the nodes into levels. Somme nodes will have a level FL_j, others will have a level SL_j and other nodes will have a level TL_j. The node takes only one level. Notice that, the partitioning procedures are always from top of the chain to bottom. In the following, $FL_x(i)$ means node i has the level FL_x, $TL_x(i)$ means node i has the level TL_x and $SL_x(i)$ means node i has the level SL_x.

The case n is odd:

Group (A):

The node $((n+1)/2)$ takes a special level $PL0$ and the node $((n+1)/2) - 1$ takes a special level $PL1$. The first $A = \sqrt{n}$ nodes take the first level FL_0. The next $B = (\sqrt{n} - 3)/2$ nodes take the first level SL_0. The next $C = (\sqrt{n} - 1)/2$ nodes take the level TL_0. After:

- The next $A = A - 2$ nodes take the level FL_j if $TL(A - 1)$.
- The next $B = B - 1$ nodes take the level SL_j if $FL(B - 1)$.
- The next $C = C - 1$ nodes take the level TL_j if $SL(C - 1)$.

Figure 6 shows an example of partitioning into levels for the group (A) with $n = 49$.

$$O_j = \begin{cases} (\sqrt{n} - 1)/2, if\, j = 0. \\ O_{j-1} - 1, otherwise. \end{cases} \tag{1}$$

$$W_j = \begin{cases} (3\sqrt{n} - 7)/2, if\, j = 0. \\ W_{j-1} - 3, otherwise. \end{cases} \tag{2}$$

$$V_{i,j} = \begin{cases} (n + 1/2) - ((\sqrt{n} + 1)/2), if\, FL_0(i). \\ 0, if\, PL0(i). \\ 1, if\, PL1(i) \\ V_{i-1} - W_{j-1}, if\, FL(i) \wedge TL(i-1). \\ V_{i-1} - 1, if\, SL(i) \wedge FL(i-1). \\ V_{i-1} - O_j, if\, TL(i) \wedge SL(i-1). \\ V_{i-1}, otherwise. \end{cases} \tag{3}$$

With $V_{i,j}$ is the number of movements of node i that has the level j, O_j and W_j are functions used to calculate $V_{i,j}$. To each level TL_j is associated a number O_j, and to each level FL_j is associated a number M_j. An example in figure 6.

Group (B):

The first two nodes take the first level FL_0. The next $A = 5$ nodes take the first level SL_0. The next $B = 2$ nodes take the level FL_1. The last $(\sqrt{n}+1)/2$ nodes take the last level SL_L After :

- The next $A = A + 3$ nodes take the level SL_j if $FL(A - 1)$.

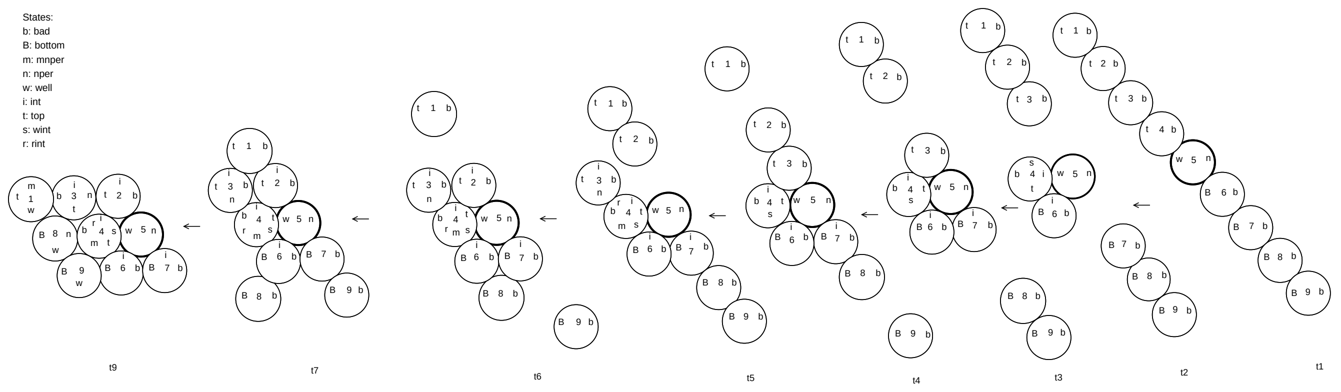

Figure 5. Represents an example of execution of PAUC with nine nodes, the initiator is the node 5

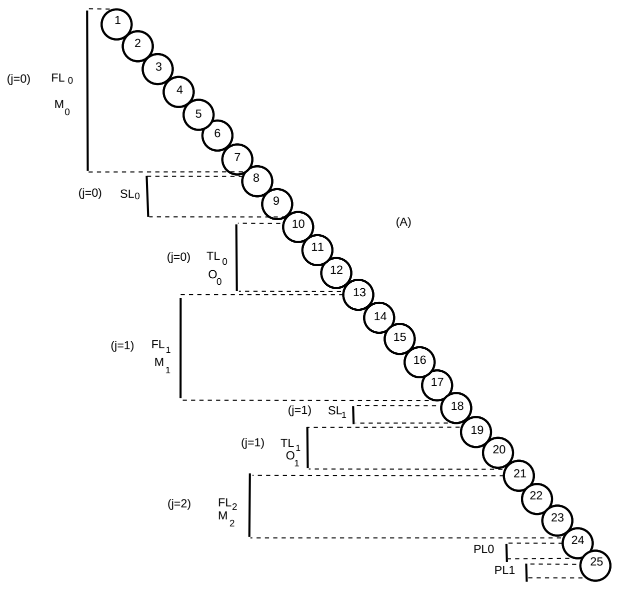

Figure 6. An example of nodes partitioning into levels for the group (A) in the case $n = 49$

- The next $B = B+1$ nodes take the level FL_j if $SL(B-1)$.

$$S_j = \begin{cases} 5, if\, j = 0. \\ S_{j-1} + 3, otherwise. \end{cases} \tag{4}$$

$$D_j = \begin{cases} 4, if\, j = 0. \\ D_{j-1} + 1, otherwise. \end{cases} \tag{5}$$

$$Z_{i,j} = \begin{cases} 1, if\, FL_0(i). \\ 5, if\, SL_0(i). \\ Z_{i-1} + S_{j-1}, if\, FL(i) \wedge SL(i-1). \\ Z_{i-1} + D_{j-1}, if\, SL(i) \wedge FL(i-1). \\ Z_{i-1}, otherwise. \end{cases} \tag{6}$$

With $Z_{i,j}$ is the number of movements of node i that has the level j, S_j and D_j are functions used to calculate $V_{i,j}$. To each level FL_j is associated a number S_j, and to each level SL_j is associated a number D_j.

The case n is even:

Group (A):

The node $(n/2 - ((\sqrt{n}/2) - 1))$ takes a special level $PL0$ and the node $(n/2 - ((\sqrt{n}/2) - 2))$ takes a special level $PL1$. The first $A = \sqrt{n}/2$ nodes take the fist level FL_0. The next $B = (\sqrt{n}/2) - 1$ nodes take the first level SL_0. The next $C = (\sqrt{n}/2) - 2$ nodes take the the level TL_0. After:

- The next $A = A-1$ nodes take the level FL_j if $TL(A-1)$.
- The next $B = B-2$ nodes take the level SL_j if $FL(B-1)$.
- The next $C = C-1$ nodes take the level TL_j if $SL(C-1)$.

With $FL_x(i)$ means node i that has the level FL_x, $SL_x(i)$ means node i that has the level SL_x and $TL_x(i)$ means node i that has the level TL_x

$$P_j = \begin{cases} (3\sqrt{n}/2) - 2, if\, j = 0. \\ P_{j-1} - 4, otherwise. \end{cases} \tag{7}$$

$$K_j = \begin{cases} (\sqrt{n}/2) - 1, if\, j = 0. \\ K_{j-1} - 1, otherwise. \end{cases} \tag{8}$$

$$H_{i,j} = \begin{cases} (n/2) - 1, if\, FL_0(i). \\ 0, if\, PL0(i). \\ 1, if\, PL1(i) \\ H_{i-1} - P_j, if\, SL(i) \wedge FL(i-1). \\ H_{i-1} - 1, if\, TL(i) \wedge SL(i-1). \\ H_{i-1} - K_{j-1}, if\, FL(i) \wedge TL(i-1). \\ H_{i-1}, otherwise. \end{cases} \tag{9}$$

With $H_{i,j}$ is the number of movements of node i that has the level j, P_j and K_j are functions used to calculate $V_{i,j}$. To each level TL_j is associated a number P_j, and to each level FL_j is associated a number K_j.

Group (B):

The first two nodes take the first level FL_0. The next $A = 5$

nodes take the first level SL_0. The next $B = 2$ nodes take the level FL_1. After:

- The next $A = A+3$ nodes take the level SL_j if $FL(A-1)$.
- The next $B = B+1$ nodes take the level FL_j if $SL(B-1)$.

$$U_{i,j} = \begin{cases} 1, if\, FL_0(i). \\ 5, if\, SL_0(i). \\ U_{i-1} + S_{j-1}, if\, FL(i) \wedge SL(i-1). \\ U_{i-1} + D_{j-1}, if\, SL(i) \wedge FL(i-1). \\ U_{i-1}, otherwise. \end{cases} \tag{10}$$

With $U_{i,j}$ is the number of movements of node i that has the level j, S_j and D_j are functions used to calculate $V_{i,j}$. To each level FL_j is associated a number S_j, and to each level SL_j is associated a number D_j.

D. Generalization of the algorithm

Presented algorithm PAUC is specific to a chain case where nodes form initially a straight line oriented toward SE-NW directions. In this section we describe how the algorithm can be generalized to any kind of initial chain with any direction. We start by explaining how the two end nodes are selected whatever are the directions of the straight chain. The node that has only one neighbor situated either in SW, SE or E direction is the first end node. The second end node has only one neighbor situated either in NW, NE or W. For the other nodes, every node in the chain can deduce the orientation of the chain by analyzing the orientation of its two neighbors, they use the orientation of their two neighbors to determine the orientation of the formed chain. Generally, every node after the detection of the chain orientation, noted D-$\overline{D}$, runs a variant of the PAUC algorithm depending of the orientation $D \in \{W, NW, NE\}$. The variant of PAUC algorithm, $PAUC^D$, represents an adaptation of the the original PAUC algorithm (corresponding to $PAUC^{NW}$) to the two other possible orientations with changing the directions in predicates.

VI. Simulation

We have done the simulation with the Dynamic Physical Rendering Simulator [27]. In our simulations the radius of the node is 1 mm. We simulated with a laptop with processor Intel(R) Core(Tm) i5, 2.53 GHz with 4 Gb of memory. We note in the figures of simulation, $PAUC1$ for the values odd of n with $t(\eta) = \eta/2 - 1$ and $p(n) = ((n + \sqrt{n} - 2)/2)$. And $PAUC2$ for the values even of n, with $v(n) = (n/2 - 1)$ and $s(\eta) = ((\eta + \sqrt{\eta} - 2)/2)$

The simulation results come to agree our theoretical results. Figure 7 represents the execution time in ticks by the number of nodes; this figure compares the execution time of the algorithm proposed in this paper to those given in [15] and [17]. Figure 8 presents the highest number of movements found in this paper compared to the one in [15]. Figure 9 compares also

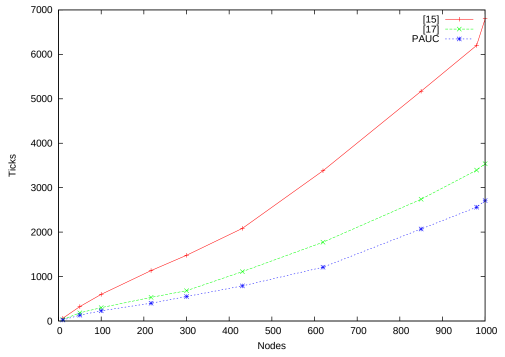

Figure 7. Execution time

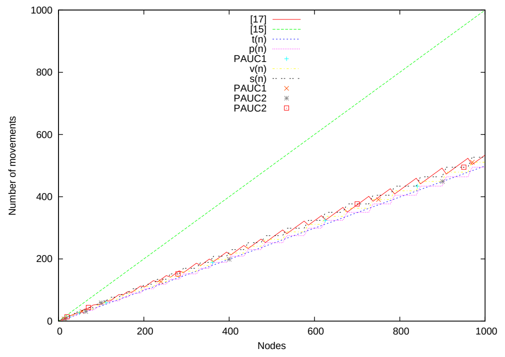

Figure 8. Highest number of movements

the sub-squares (we call *sub-squares* the intermediate squares generated before the final square) generated by the time with simulation on 1000 nodes. The effects of parallelism appear well in the curve representing the execution time of PAUC, [15] and [17], as in PAUC movements of microrobots are in parallel which optimizes the time of the algorithm and PAUC makes two rectangles in the same time that their union gives the target shape. An interesting thing to notice here is that optimizing the execution time of the algorithm will have a direct effect on messages therefore a gain on communication, and if the algorithm is fast then the critical information arrives early at the concerned node. Also, if the task is a heavy parallel computation, therefore if the algorithm is faster, the parallel computing will be fast and light on the nodes because the tasks are well distributed. In figure 7 we see that whenever the network size increases the difference increases dramatically. We remark in figure 8 that the number of movements in PAUC is much lower, which will increases the probability of lifetime of nodes, therefore the probability that the node continues its task (its movements), this is also improving the energy consumption. However, PAUC needs seven states per node and the algorithm in [15] needs three states per node, and the algorithm in [17] needs ten states. We see in figure 9 that the sub-squares are obtained early compared to the other protocols. This is explained with the fact that in the other solutions only leaf nodes that can move and they use a tree *(O(N) time)* to manage these leaf nodes (to ensure a snap-connectivity). Therefore, in these solutions the first sub-square is obtained

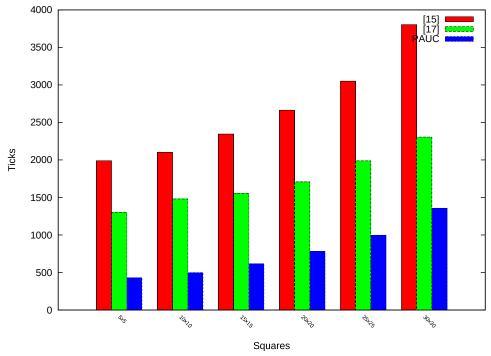

Figure 9. Sub-squares generated based on the time with simulation on 1000 nodes

after the construction of the tree *(O(N) time)* and a after arriving of the extremity node of the chain to the initiator *(O(N) time)* [15], and in [17] after the construction of the tree *(O(N) time)* and after arriving the two extreme nodes to the center of chain with *(O(N/2) time)*.

VII. Conclusion

In this paper, we presented a new method to complete the redeployment where the nodes do not know the fixed positions of the target shape but only the aimed shape. Compared to the literature works this algorithm is scalable because each node needs only seven state to achieve the redeployment. Nodes in our paper can perform the algorithm regardless the place where they are because the algorithm is independent of the map, that what we call portability. We have shown a robust parallel redeployment without predefined positions of the target shape. The proposed algorithm optimizes the execution time and the number of movements, each node needs seven states to help and collaborate with neighbors, its execution time and highest numbers of movements are much better than those in literature works. However, some open problems remain; we will study the fault tolerance on redeployment in microrobots networks. We will study of the effect of redeployment on the permutation routing [16] where the objective will be to optimize the path of a node to go to the correct position where it finds its correct data.

VIII. Acknowledgments

This work is funded by the Labex ACTION program (contract ANR-11-LABX-01-01), ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F) and ANR (contract ANR-2011-BS03-005). The authors wish to express their appreciation to the anonymous reviewers for their constructive comments.

References

[1] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. Pillai, *Meld: A Declarative Approach to Programming Ensembles,* In Proc. of the IEEE Int. Con. on Intelligent Robots and Systems, October, 2007.

[2] M. P. Ashley-Rollman, P. Lee, S. C. Goldstein, Padmanabhan Pillai, and Jason D. Campbell, *A Language for Large Ensembles of Independently Executing Nodes,* In Int. Con. on Logic Programming, 2009.

[3] 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, Spain, p 118–123, 2013.

[4] H. Bojinov, A. Casal, T. Hogg, *Emergent structures in modular self-reconfigurable robots,* Proc. of the IEEE Int. Con. on Robotics and Automation, vol. 2, pp. 1734-1741, Los Alamitos, 2000.

[5] Z. J. Butler, K. Kotay, D. Rus, K. Tomita, *Generic decentralized control for lattice-based self-reconfigurable robots*, International Journal of Robotics Research 23(9):919-937, 2004

[6] D. Dewey, S. S. Srinivasa, M. P. Ashley-Rollman, M. D. Rosa, P. Pillai, T. C. Mowry, J. D. Campbell, and S. C. Goldstein, *Generalizing Metamodules to Simplify Planning in Modular Robotic Systems,* In Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, September, 2008

[7] T. Ebefors, J.U. Mattsson, E. K. lvesten, and G. Stemme, *A walking a silicon microrobot*, in The 10th Int. Conf. on Solid-State Sensors and Actuators (Transducers '99), pages 1202-1205, Sendai, Japan, June 1999.

[8] S. Funiak, P. Pillai, M. P. Ashley-Rollman, J. D. Campbell, and S. C. Goldstein, *Distributed Localization of Modular Robot Ensembles,* In Proc. of Robotics: Science and Systems, June, 2008.

[9] C. Jones, M. J. Mataric, *From local to global behavior in intelligent self-assembly.In: Proceedings of the 2003 IEEE Int. Conf. on Robotics and Automation*,vol. 1, pp. 721-726, Los Alamitos, 2003.

[10] S. Jeon, C. Ji, *Randomized Distributed Configuration Management of Wireless Networks: Multi-layer Markov Random Fields and Near-Optimality* CoRR abs/0809.1916, 2008.

[11] S. Hollar, A. Flynn, C. Bellew, and K.S.J. Pister, *Solar powered 10mg silicon robot*,In MEMS, Kyoto, Japan, January 2003.

[12] M. E. Karagozler, A. Thaker, S. C. Goldstein, D. S. Ricketts, *Electrostatic Actuation and Control of Micro Robots Using a Post-Processed High-Voltage SOI CMOS Chip*,in IEEE Int. Symp. on Circuits and Systems, 2011.

[13] K. Katoy, D. Rus, M. Vona, and C. McGray, *The Self-reconfiguring Robotic Molecule*, in Proc. of the 1998 IEEE Int. Conf. on Robotics and Automation, Leuven, 1998.

[14] F. Kribi, P. Minet, A. Laouiti, *Redeploying mobile wireless sensor networks with virtual forces*, IFIP Wireless Days, Paris, France,2009.

[15] H. Lakhlef, H. Mabed, J. Bourgeois, *Distributed and Efficient Algorithm for Self-reconfiguration of MEMS Microrobots*, in the 28th ACM Symposium On Applied Computing, P. 560-566, Portugal, March 2013.

[16] H. Lakhlef, J. F. Myoupo, *Secure permutation routing protocol in multi-hop wiress sensor networks,* Int. Conf. on Security and Management (SAM'11), pp. 691-696, 2011.

[17] H. Lakhlef, H. Mabed, J. Bourgeois, *Parallel Self-reconfiguration for MEMS Microrobot*, in the 7-th IEEE Int. conf. on Computer as a Tool, Page(s): 283 - 290 Zagreb, Croatia, July 2013.

[18] H. Lakhlef, H. Mabed, J. Bourgeois, *Dynamicity to Save Energy in Microrobots Reconfiguration*, in 10th IEEE International Conference on Ubiquitous Intelligence and Computing , Page(s): 246-253,Vietri sul Mare, Italy, December 2013.

[19] M. Mamei, M. Vasirani, F. Zambonelli, *Experiments of Morphogenesis in Swarms of Simple Mobile Robots*, Journal of Applied Artificial Intelligence, 8(9-10):903-919, Oct. 2004.

[20] R. Ravichandran, G. Gordon, and S. C. Goldstein: *A Scalable Distributed Algorithm for Shape Transformation in Multi-Robot Systems,* In Proc. of the IEEE Int. Con. on Intelligent Robots and Systems, 2007.

[21] M. D. Rosa, S. C. Goldstein, P. Lee, J. D. Campbell, and P. Pillai, *Programming Modular Robots with Locally Distributed Predicates,* In Proc. of the IEEE Int. Con. on Robotics and Automation, 2008.

[22] D. Rus, M. Vona, *Crystalline robots: Self-reconfiguration with compressible unit modules*,Autonomous Robots 10(1), 107-124, 2001.

[23] E. Sahin. *Swarm robotics: from sources of inspiration to domains of application,*Swarm Robotics, SAB 2004 International Workshop (Revised Selected Papers) E. Sahin and W. M. Spear (Eds.), Lecture Notes in Computer Science 3342, Springer, 2005.

[24] K.Stoy, R.Nagpal, *Self-reconfiguration using Directed Growth*, 7th Int. Symp. on Distributed Autonomous Robotic Systems (DARs), France, June23-25, 2004.

[25] P. White, V. Zykov, J. C. Bongard, H. Lipson, *Three dimensional stochastic reconfiguration of modular robots* In: Proc. of Robotics Science and Systems, pp. 161-168. MIT Press, Cambridge , 2005

[26] F. Zambonelli, M.P. Gleizes, M. Mamei, R. Tolksdorf,*Spray Computers: Explorations in Self-Organization*, Journal of Pervasive and Mobile Computing, Elsevier, Vol. 1, p. 1-20, 2005.

[27] *http://www.pittsburgh.intel-research.net/dprweb*

[28] *http://smartblocks.univ-fcomte.fr/*