Energy-Aware Parallel Self-reconfiguration for Chains Microrobot Networks

Hicham Lakhlef[a,*], Julien Bourgeois[a], Hakim Mabed[a], Seth Copen Goldstein[b]

[a] *UFC/FEMTO-ST, UMR CNRS 6174, 1 cours Leprince-Ringuet, 25201 Montbeliard, France*
[b] *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA*

Abstract

MEMS microrobots are miniaturized electro-mechanical elements, made using the techniques of micro-fabrication. They have limited energy capacity and low memory space. Self-reconfiguration is required for MEMS microrobots to complete their mission and/or to optimize their communication. In this paper, we present a self-reconfiguration protocol from a straight chain to square organization, which deals with MEMS microrobots characteristics. In the proposed protocol, nodes do not have the map of their target positions which makes the protocol portable, standalone, and the memory complexity is bounded by a constant. This paper improves a former solution by using parallelism in the movements of microrobots to optimize the time and the number of movements and by making the algorithm energy-aware. So each node is aware of the amount of energy that it will spend, which will improve the energy consumption. Our algorithm is implemented in Meld, a declarative language, and executed in a real environment simulator called DPRSim.

Keywords: MEMS Microrobot; Distributed Algorithm; Parallel Algorithm; Self-reconfiguration; Logical Topology; Energy

1. Introduction

Micro electro mechanical system (MEMS) is a technology that enables the batch fabrication of miniature mechanical structures, devices, and 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 [1]. Their applications will require a massive deployment of nodes, thousands or even millions [2] which will give birth to the concept of Distributed Intelligent MEMS (DiMEMS) [3].
The size of MEMS nodes differs from well below one micron to few millimeters. A DiMEMS device is composed of typically hundreds of MEMS nodes. Some DiMEMS devices are composed of mobile MEMS nodes [4], some others are partially mobile [5] whereas others are not mobile at all [3]. Due to their small size and the batch-fabrication process, MEMS microrobots are potentially very cheap, particularly through their use in many areas in our lifetime [6].
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 [4, 7, 8, 9] each microrobot can only turn around its neighbor which introduce 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 [10].
MEMS microrobots topic is gaining an increasing attention since large-scale swarms of robots will be able to perform several missions and tasks in a wide range of applications such as odor localization, firefighting, medical service, surveillance, search, rescue, and security[1]. The self-reconfiguration for MEMS microrobots is necessary to do these tasks. In the literature, self-reconfiguration can be seen from two different points of view. On the one hand, 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 [11]. For example, if we have a connected chain of n microrobots then the complexity of message exchange if a node broadcasts a message to others will be $O(n)$ in the worst case. If we reconfigure the chain to a square the complexity will be $O(\sqrt{n})$ in the worst case. On the other hand, the self-reconfiguration is built from modules which are autonomously able to change the way they are connected, thus changing the overall shape of the logical network [8, 12]. This process is difficult to control, because it involves the distributed coordination of a large number of identical modules connected in time-varying ways. The range of exchanged information and the amount of displacement, determine the communication and energy complexity of the distributed algorithm. When the information exchange involves close neighbors,

*Corresponding author
Email addresses: hlakhlef@femto-st.fr(H. Lakhlef), julien.bourgeois@femto-st.fr(J. Bourgeois), hmabed@femto-st.fr (H. Mabed), seth@cs.cmu.edu(S. C. Goldstein)

the complexity is moderate and the resulting distributed self-reconfiguration scales gracefully with network size.
An open issue is whether distributed self-reconfiguration would result in an optimal configuration with a moderate complexity in message, execution time, number of movements and memory usage.
As said before, MEMS microrobots are low-power and low-memory capacity devices that can sense and act. A solution of self-reconfiguration should deal with MEMS microrobots characteristics. Self-reconfiguration with shared map does not scale. Because the map (predefined position of the target shape) consists of P positions and each node must have a memory capacity, at least, of P positions. Therefore, if P is very high, the self-reconfiguration will be not feasible. In this paper, we present an energy-aware parallel reconfiguration algorithm, without predefined positions of the target shape, which reduces memory usage to $O(1)$. This algorithm ensures the networks connectivity throughout all its execution time. 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.

2. Related Works

Many terms refer to the concept of self-reconfiguration. In several works on wireless networks the term used is $self-organization$. This term is also used to express the partitioning and clustering of ad-hoc networks or wireless networks to groups called cliques or clusters. Also, the self-organization term can be found in protocols for sensor networks to form a sphere or a polygon from a center node [13, 14, 15]. The term *redeployment* is also a new term to address self-reconfiguration for sensor networks [16, 17, 18]. For self-reconfiguration with robots or microrobots, there are the protocols [12, 19, 20] where the desired configuration is grown from an initial seed module. A generator uses a 3D model of the target configuration and outputs a set of overlapping blocks which represent this configuration. In the second step, this representation is combined with a control algorithm to produce the final self-reconfiguration algorithm. In [21], the authors propose map based distributed algorithm for self-reconfiguration of modular robots from arbitrary to straight chain configuration.
A growing number of research on self-reconfiguration for microrobots have used centralized algorithms, 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 [23]. The drawback of these methods is the centralized paradigm and the need for nodes identification. More distributed approaches that need the map of the target shape in [24, 25, 26, 27]. In simulation, the authors in [28, 19] have demonstrated algorithms for self-reconfiguration and directed growth of cubic units based on gradients and cellular automata. The authors in [29] 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.
In [30] the authors developed a centralized algorithm for reconfiguration (with predefined positions of the target chain) of an initial chain configuration into another chain configuration and then from a straight chain into an arbitrary goal that fulfills certain admissibility requirements [31]. The distributed version of this algorithm was given in [21]. Recent work in [32] demonstrated a time complexity of $O(n)$ for probabilistic reconfiguration of large systems of hexagonal metamorphic robots for single-move algorithms, in which at most one module can move in a time step.
Claytronics, is the name of a project led by Carnegie Mellon University and Intel corporation. In Claytronics, microrobots called catoms (Claytronics atoms) are assembled to form larger objects. The idea is to have hundreds of thousands of microrobots forming objects of any shape. Like the cells in a body or in a complex organism, each small member of the whole is committed to doing its own part and communication between microrobots helps in building the final shape.
Many works have already been done within the Claytronics project. In [33], 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. In [8], the authors present a scalable distributed reconfiguration algorithm with the Hierarchical Median Decomposition, to achieve arbitrary target configurations without a global communication. Another scalable algorithm has been presented in [9]. In [7], a scalable protocol for Catoms self-reconfiguration is proposed, written with the Meld language [4, 34] 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 [35]. However, this solution is not parallelized, is not energy-aware and takes longer to achieve the reconfiguration. In this former solution, the choice of the initiator node is simple, but this initiator is not the best one to get a good execution time for the self-reconfiguration protocol. Also, controlling the movements of nodes was simple compared to this new solution because only three states were required. Nevertheless, the solutions in [35, 37] take longer to achieve the reconfiguration protocol. Therefore, they are time-consuming, and do not maximize the lifetime of nodes. To cope with these disadvantages, we introduced the use of the parallelism in movements. Within this new solution, and because of the parallelism in movements, the control has become more complex, as new states are required to deal with a lot of cases to handle the parallelism in order to make it optimal. We presented in [38] an algorithm of reconfiguration from any starting physical

topology to a square, this algorithm does not ensure the connectivity of the network during the reconfiguration.

3. Contributions and comparison with literature works

In this paper, we propose a new distributed approach for parallelized self-reconfiguration of MEMS microrobots, where the target form is built incrementally and in parallel way (parallel movements). Each node in the current increment acts as a reference for other nodes to form the next increment, which will belong to the final form. In this paper each node predicts its future actions (movements), so it can compute the energy amount that will spend before the beginning of the algorithm. The prediction property makes the algorithm robust, because the node can make sure that it has correctly followed the algorithm.

We introduce a state model where each node can see the state of its physical neighbors to achieve the self-reconfiguration for distributed MEMS microrobots, using the states the nodes collaborate and help each other.

In the proposed algorithm, message exchanging is limited to the construction of the spanning tree and the selection of the initiator node (in the middle of the initial shape) in order to optimize the algorithm. The spanning tree is used to ensure the connectivity of the network and dynamically manage the nodes that can move. Contrary to existing works, in our algorithm each node has no information on the correct positions (predefined positions) of the target shape, and movements of microrobots are fully implemented. We propose here an efficient distributed and parallelized algorithm for nodes self-reconfiguration. Each node moves by rotation around their physical neighbors. For instance, we study the case of a self-reconfiguration from a chain of microrobots to a square. The performance of the self-organization algorithm is evaluated according to the number of rotations and the time taken. In this paper the MEMS network is organized initially as a chain. By choosing a straight chain as 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. The redeployment into a square organization allows to obtain the best messages broadcasting complexity with $O(\sqrt{n})$, instead of $O(n)$ in the chain.

To assess the distributed algorithm performance and the effect of the parallelization, we present our results of simulations compared to the results in [35], which are not parallelized. The simulations are made with the declarative language Meld [4] and the DPRSim simulator [39].

Outline of the paper. The rest of the paper is organized as follows: Section 4 discusses the model, tools and some definitions. Section 5 discuss the proposed algorithm, analyzes the number of sent messages and the number of movements, it discuss memory space required 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.

4. Model, definitions and tools

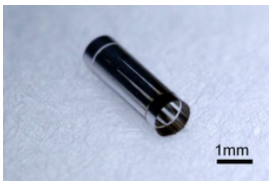

Figure 1: A millimetric Claytronics atom (Catom)

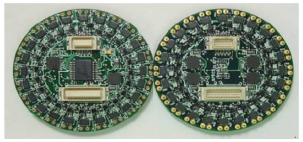

Figure 2: Two catoms

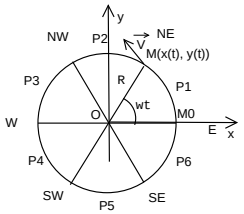

Figure 3: Node modeling, in each movement the node travels the same distance

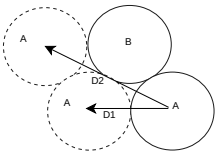

Figure 4: Traveled distance in one movement = 2R, the node A travels 2R in one movement

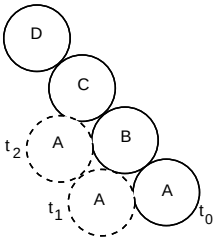

Figure 5: Message transmission, there will be message exchange if the node needs to know the state of a non-neighbor node

Within Claytronics, the Catoms can have two shapes, a sphere or a cylinder, a catom (See figure 1 and figure 2) that we call in this paper, a node, is modeled as a circle which can have at most six 2D-neighbors without overlapping (See figure 3). Each node 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)). In this work, the starting physical topology is a chain of n nodes linked together. A chain corresponds to a connected set of nodes where each node has two neighbors excepting the two extremities representing only one neighbor. 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 touch physically B (figure 4). Within Claytronics,

communications are only possible through contact, which means that only neighbors can have a direct communication.

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 $N(v) = \{u, (u, v) \in E\}$. Each node $v \in V$ knows the set of its neighbors in G, denoted $N(v)$. Each node regularly updates the set of neighbors $N(v)$ for each node v of the graph.

We define the following terminology:

Connectivity: A graph $G = (V, E)$ is connected iff $\forall v \in V$, $\forall u \in V, \exists C_{v,u} \subseteq$E: $C_{v,u} = (e_{v,-}, ..., e_{-,-}, ..., e_{-,u})$, with $e_{x,y}$ is an edge from x to y and $C_{v,u}$ represents a path from v to u.

Snap − Connectivity: let T be the total execution time of our distributed algorithm DA and $t_1 ..., t_m$ are the time slots of execution of DA. There is a Snap-Connectivity in DA with the dynamic graph $G_t(V_{ti}, E_{ti})$ the network state at the instant t_i, if $\forall t_i, i \in \{1, ..., m\}$, $G_{ti}(V_{ti}, E_{ti})$ maintains the connectivity.

Spanning tree: is a tree composed of all $v \in V$ without any cycle. In the spanning tree, a node is either a child or a parent. The leaf is a node without any children.

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

To calculate highest number of movements we define the following:

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 $D1 = 2R$. For example, if the node is in a position at a distance $D2$ (see figure 4) from the former position it has done two movements.

Consider figure 3 which represents a microrobot. The node perimeter corresponds to an angle of 360° that can be divided into six equal segments each one of 60°. The perimeter length of a segment with α degree is equal to $P_\alpha = \pi R\alpha/180$. According to figure 3 we prove that $P1 = P2 = P3 = P4 = P5 = P6 = \pi R/3$ and in each round the node travels the same distance, this also means that the node can have without overlapping at most six neighbors. In figure 3, the points $M0$ and M represent the contact point between the node in movement and the node around which it moves.

The movement performed by a node can be represented by the following Cartesian parametric equation:

$$\begin{cases} x(t) = Rcos(wt) \\ y(t) = Rsin(wt) \\ where\, wt \in [0..2\pi] \end{cases} \quad (1)$$

With wt is the angle of rotation w in a period t and $x(t)$ and $y(t)$ the coordinates of the M point (see figure 3).

The velocity vector is written:

$$\vec{V} = \begin{pmatrix} -Rsin\ wt \\ Rcos\ wt \end{pmatrix} \quad (2)$$

The arc length from M to $M0$ is equal $\int \| \vec{V} \|.dt = Rwt$, with $\| \vec{V} \|$ is the norm of the velocity vector. So in this paper in one round the microrobot of radius R travels Ra with a=60°.

In this paper, message exchange between physical neighbors is carried without complexity, because the node can see directly the state of its physical neighbor[1]. On the other side, if a node to decide needs to know the state of a nonphysical neighbor this is carried through exchange of message since the node will wait to decide, for example in figure 5:

- At t_0: the node A needs to know the state of B to move to the new position, this movement is done without message exchange.
- At t_2: if A is in the new position and it needs to know the state of D to move then D sends a message to C informing its state to C that forwards the message to A. So, in this case there is a message exchange and A should wait two rounds to decide [2].
- But if at t_0 or at t_1 a message has been sent from D to C, so A at t_2 can have the state of D with a simple consultation of C's state.

It is important to minimize the number of movements regarding the energy, the execution time and the memory space used, therefore the number of states per node.

Lemma 4.1. [3]

Let x be an integer number. It is well known that if x is odd\even, then x^2 is an odd\even number.

Proof As x is odd\even, we can write $x = 2n+1 \backslash x = 2n$. Therefore, $x^2 = (2n+1)^2 \backslash x^2 = (2n)^2$. So, $x^2 = 4n+4+1 = 2(2n^2+2)+1 \backslash x^2 = 2(2x^2)$; which is an odd \ even number.

Lemma 4.2. *Let x be a square number (x is an integer that is the square of an integer). If x is even, then $\sqrt{x}$ is an even number.*

Proof We proof this lemma by induction. As x is even and it is the square of an integer, we can write $x = n^2$. Therefore, $\sqrt{x} = \sqrt{n^2}$. Let us suppose that n is odd, so there is k with $n = 2k + 1$. Thus, $\sqrt{x} = \sqrt{(2k+1)^2}$. So, $x = 2(2k^2 + 2k) + 1$ which represents a contradiction, because this value is odd and our x is even. So, $\sqrt{x}$ is even.

[1] According to the simulation tools the node can see directly the state of its physical neighbor

[2] Node A has to wait for the state changing of B, firstly D will free C, after C will free A

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

Lemma 4.3. *Let x be a square number (x is an integer that is the square of an integer). If x is odd, then $\sqrt{x}$ is an odd number.*

Proof As x is odd, there is an even integer $2h$ with $x = 2h + 1$. Therefore, $\sqrt{x} = \sqrt{2h+1}$. Notice that, there is an integer η with $\sqrt{2h+1} = \sqrt{2\eta}+1$ where $h = \eta + \sqrt{2\eta}$. Therefore, the value $\sqrt{2h+1} = 4\sqrt{\eta}+1$ which is an odd number.

Theorem 4.4. *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 $y + 4\sqrt{y} + 4$*

Proof As y is an odd\even square number, we have $\sqrt{y} = \rho$, with ρ is odd\even (from lemma 4.1, lemma 4.2 and lemma 4.3) integer number. So, as ρ is odd\even, the next odd\even number is $r = \rho+2$, and because $r^2 = (\rho+2)^2 = \rho^2 + 4\rho + 4$ is odd\even (from lemma 4.1, lemma 4.2 and lemma 4.3), we find $r^2 = y + 4\sqrt{y} + 4$ □

Theorem 4.5. *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 $y + 2\sqrt{y} + 1$*

Proof As y is an odd\even square number, from lemma 4.1, lemma 4.2 and lemma 4.3 we have $\sqrt{y} = \rho$, with ρ is odd\even integer number. So, as ρ is odd\even, the next even\odd number is $r = \rho+1$, and because $r^2 = (\rho+1)^2 = \rho^2 + 2\rho + 1$ is even\odd (from lemma 4.1, lemma 4.2 and lemma 4.3), we find $r^2 = y + 2\sqrt{y} + 1$ □

5. Proposed Protocol

5.1. Parallel Algorithm with Safe Connectivity (PASC)

As mentioned before, in this algorithm, each node can move only around its physical neighbor. To ensure a snap-connectivity only nodes that keep the network disconnectivity can move around neighbors, for this purpose we introduce the use of the tree to dynamically manage the leaf nodes that can move.

To form the matrix of our square with $\sqrt{N} \times \sqrt{N}$ nodes,

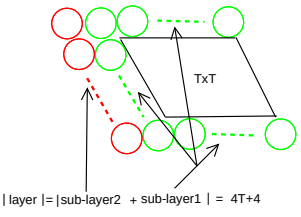

Figure 6: Represents how many nodes added to reach the next square

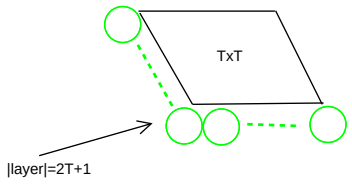

Figure 7: Represents how many nodes added in last layer to reach the last square when n is even

we begin with an incremental process with a correct square (for example 1x1). After, 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 6 and 7 show an example. The choice of the middle node depends on the optimality of parallelism. Let N be the network size, and $n = \left\lfloor \sqrt{N} \right\rfloor \left\lfloor \sqrt{N} \right\rfloor$, if n is odd the middle node will be $mi = \frac{n+1}{2}$, as the case in figure 9. If n is even the middle node will be $mi = \frac{n}{2} - (\frac{\sqrt{n}}{2} - 1)$, as the case in figure 8. The index of a node represents its rank into the initial chain (starting from the top).

The middle node mi can be found by knowing the size of the network, an end node of the chain initializes a counter to 1 and broadcasts it, each node receives this message increments the counter until it arrives to the concerned node mi.

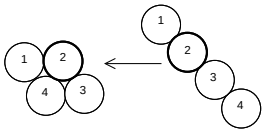

Figure 8: Represents an example of initiator finding when n is even, in this example the initiator is the node 2

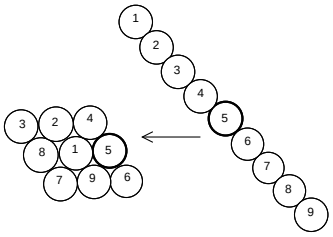

Figure 9: Represents an example of initiator finding when n odd, in this example the initiator is the node 5

5.2. Description and analysis

The algorithm PASC (presented hereafter) runs in rounds. At each round, the satisfied predicates are chosen to run. The distributed algorithm seeks the desired form by using an incremental process. In a completed increment, the nodes that build it belong already to the form; these nodes will help neighbor nodes and future neighbor nodes to get correct positions.

The middle node of the chain declares itself as the initiator with the predicate (1). The initiator which is the root of the spanning tree initializes the construction of tree and becomes a parent of itself (3). A node if it does not have a parent becomes a child of one the neighbor parents (8) and a node is a leaf if all its neighbors are parents (9). Nodes that are above the initiator take the state *top* with predicate (6), the other nodes that are under the initiator take the state *bottom* (7). Initially, all nodes are initialized with the state *bad* except the initiator (2), which takes the states *well* and *nper* with (4) and (5). 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 in term of numbers of movements and a correct square, the number of nodes having the state *top* to be in the same line as the initiator must be equal to the number of nodes having state *bottom* 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.

Variables and Predicates:

- initiator$_v()$: the node v that initializes the algorithm.
- state$_v(X)$: the node v takes the state X, with:
  $X \in \{well, bad, int, nper, mnper, top, bottom, \neg nper, \neg mnper, \neg int\}$, 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.
- parent(v, u): v is u's parent in the tree.
- isLeaf(v): v is a leaf.

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. parent$(v, v) \equiv initiator(v)$.
4. state$_v(well) \equiv initiator_v()$.
5. state$_v(nper) \equiv initiator_v()$.

Predicates checked in each round

6. state$_v(top) \equiv (N_{se}(v) = u, initiator_u()) \vee (N_{se}(v) = u, state_u(top))$.
7. state$_v(bottom) \equiv (N_{nw}(v) = u, initiator_u()) \vee (N_{nw}(v) = u, state_u(top))$.
8. parent$(v, u) \equiv (parent(w, v), u \neq w) \wedge (u \in N(v)) \wedge (state_u(bad)) \wedge (\nexists z \in N(v), parent(v, z))$.
9. isLeaf$(v) \equiv ((\forall u \in N(v), \neg parent(v, u)) \wedge \neg parent(v, v))$.
10. state$_v(mnper) \equiv (((N_{se}(v) = u, state_u(nper)) \vee (N_e(v) = u, state_u(nper))) \wedge initiator_u())$.
11. state$_v(mnper) \equiv (N_e(v) = u, state_u(nper)) \wedge (\neg state_v(nper)) \wedge (state_v(int) \vee state_v(well))$.
12. state$_v(nper) \equiv (N_e(v) = u, state_u(mnper)) \wedge (\neg state_v(mnper)) \wedge (N_{se}(v))$.
13. state$_v(int) \equiv ((N_e(v) = u, state_u(well)) \wedge (N_{nw}(v))) \vee ((N_{se}(v) = u, state_u(well)) \wedge (N_w(v))) \vee (N_e(v) = u1, N_{se}(v) = u2, state_{u1}(int), state_{u2}(int)) \vee ((N_{ne}(v) = u, state_u(well)) \wedge (N_{nw}(v))) \vee ((N_{ne}(v) = u, state_u(well)) \wedge (N_w(v)))$.
14. state$_v(well) \equiv ((N_e(v) = u, state_u(int)) \wedge (N_{nw}(v))) \vee ((N_e(v) = u, state_u(int)) \wedge (N_{se}(v)))$.
15. state$_v(well) \equiv (N_w(v) = u, state_u(well))$.
16. state$_v(well) \equiv state_v(bad) \wedge (N_{se}(v) = \oslash) \wedge (N_w) \wedge (N_{nw}(v) = u, state_u(well))$.
17. state$_v(well) \equiv state_v(bad) \wedge (N_{se}(v) = \oslash) \wedge (N_w) \wedge (N_{nw}(v) = u, state_u(well)) \wedge ((N_e(v) = u1, state_{u1}(well)))$.
18. moveAround$bad_v(u, P_e) \equiv canMove_v() \wedge (N_{se}(v) = u, state_u(bad), state_u(top))$.
19. moveAround$bad_v(u, P_{ne}) \equiv canMove_v() \wedge (N_e(v) = u, state_u(bad), state_u(top))$
20. moveAround$int_v(u, P_{se}) \equiv canMove_v() \wedge (N_{sw}(v) = u, state_u(int), state_u(top)) \wedge (\neg state_u(nper))$.
21. moveAround$well_v(u, P_{se}) \equiv canMove_v() \wedge (N_{sw}(v) = u, state_u(well), state_u(top))$.
22. moveAround$int_v(u, P_e) \equiv canMove_v() \wedge (N_{se}(v) = u, state_u(int), state_u(top))$.
23. moveAround$well_v(u, P_e) \equiv canMove_v() \wedge (N_{se}(v) = u, state_u(well), state_u(top))$.
24. moveAround$bad_v(u, P_{ne}) \equiv canMove_v() \wedge (N_{nw}(v) = u, state_u(bad)) \wedge state_u(bottom)$.
25. moveAround$bad_v(u, P_e) \equiv canMove_v() \wedge (N_{ne}(v) = u, state_u(bad), state_u(bottom))$.
26. moveAround$well_v(u, P_{ne}) \equiv canMove_v() \wedge (N_{nw}(v) = u, state_u(well), state_u(bottom))$.
27. moveAround$well_v(u, P_{se}) \equiv canMove_v() \wedge (N_e(v) = u, state_u(well), state_u(bottom), (\neg state_u(nper))$.
28. moveAround$well_v(u, P_e) \equiv canMove_v() \wedge (N_{ne}(v) = u, state_u(well), state_u(bottom)(\neg state_u(nper))$.
29. moveAround$int_v(u, P_{ne}) \equiv canMove_v() \wedge (N_{nw}(v) = u, state_u(int), state_u(bottom))$.
30. moveAround$int_v(u, P_e) \equiv canMove_v() \wedge (N_{ne}(v) = u, state_u(int), state_u(bottom), (\neg state_u(nper))$.
31. moveAround$well_v(u, P_e) \equiv canMove_v() \wedge state_v(bottom) \wedge (N_{ne}(v) = u, state_u(well), state_u(mnper), (\neg state_u(nper))$.
32. canMove$_v() \equiv isLeaf(v) \wedge (\neg state_v(well)) \wedge (\neg state_v(int)) \wedge state_v(bad)$.

The PASC algorithm

The initiator takes the state *nper* (5), by taking this state the initiator and each node has this state does not allow its neighbor to move around it in order to join the line of the initiator. This is done with the guard ($\neg state_v(nper)$). The state *mnper* is an intermediate state used to propagate the state *nper* to the other nodes, that will keep the parallelism optimal, as well the node that has a neighbor in the E direction having the state *nper* takes the state *mnper* (11). The node that has the initiator as neighbor node in the SE direction takes the state *mnper* (10). The other (next) nodes that will take the state *nper* are nodes having in the E direction a neighbor that has the state *mnper* (12). Therefore, the node having the state *nper* does not allow neighbor nodes to join the line of the initiator, as these nodes are checking the predicates (21), (22), (23), (26), (27), (28), (29) and (30).
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 (13). 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 with the state *well* and nodes with state *int* together do not form a square, it will be a square if all nodes having the state *int* have in the W direction a neighbor node, this neighbor nodes will have has the state *well*. Therefore, the wave of state changing to *well* begins from with predicates (15), (16) and (17).
With predicates (18) and (19) the leaf nodes having the state *top* descend to the center of the chain. As well as, leaf nodes having the state *bottom* mount to the center of the chain with the predicates (24) and (25). With predicate (20) / (21), leaf node v that has the *bad* state 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 (22) / (23), leaf node v that has the *bad* state moves around a node u having the states *top* and *int/well*, node u becomes v's neighbor in the E direction. With the predicate (26) / (27) / (28), leaf node v with *bad* state moves around a node u having *well* and *bottom* states, node u becomes v's neighbor in the NE / SE / E direction.

Theorem 5.1. *If N is the network size and $n = \left\lfloor \sqrt{N} \right\rfloor \left\lfloor \sqrt{N} \right\rfloor$ is odd, the highest number of movements will be $((n+1)/2) + N - n$.*

Theorem 5.2. *If N is the network size and $n = \left\lfloor \sqrt{N} \right\rfloor \left\lfloor \sqrt{N} \right\rfloor$ is even, with $N \geq 5$, the highest number of movements will be $(\sqrt{n}/2) + N - (n/2) - 1$.*

Example: Figure 10 shows an example with explanation, and figure 11 shows another example without explanation. In figure 10:

- At $t0$: with predicate (2) each node takes the state b (*bad*), with (6) the node 1 which is above the initiator (node 2) takes the state t (*top*), with (7) nodes (nodes 3 and 4) located under the initiator take the state B (*bottom*), with (4) the initiator takes the state w (*well*), and with (5) it takes state n (*nper*).

- To arrive at the next step $t1$, node 1 moves around node 2 using the predicate (18), and node 4 moves around node 3 using (26). The node 1 takes the state m with (10). The node 1 cannot move around the node 2 with (19) since the node 2 has the state n. Node 4 moves to the new position with (24) and it cannot get any other state in this step.

- To arrive at the next step $t2$, node 4 moves around node 3 using the predicate (25). After, node 4 takes the state i (*int*) with predicate (13).

- At $t3$, the target shape is obtained. Nodes 1 and 4 take the state i with (13).

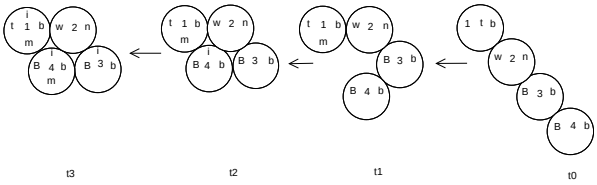

Figure 10: Represents an example of execution of PASC with four nodes, the initiator is node 2

5.3. The ten states minimum

In this section we prove that ten states are minimum and optimal to obtain the algorithm convergence. Obviously, with a single state, nodes have no way to distinguish whether they are in a good position or not and therefore if the node should move or not. Let us suppose a variant of PASC with two states *bad* and *well*, with these two states, we can say that the node that has *well* state is a steady node and is belonging to the target shape, and the node with *bad* state moves around nodes having *well* state, thereby with these two states, nodes collaborate between them to make a next layer and change the state from *bad* to *well*. Suppose a set S of nodes having the *well* state are correctly in the target shape. Depending on some conditions C the set B of nodes with *bad* state will change their state to *well* in order to make a new layer, however as we have only two states the other nodes that are B's neighbors have likewise these C conditions and they will change their states to *well* and became steady, although they are not even at the layer being built. So, PASC is executed and the target shape is lost.
Two additional states are required, the state *top* and *bottom*, these two states are indispensable to avoid deadlock in PASC. Indeed, in PASC there are predicates (18) and (19) executed by nodes to descend to the middle of the chain, and others executed to rise to the middle of the chain (24)

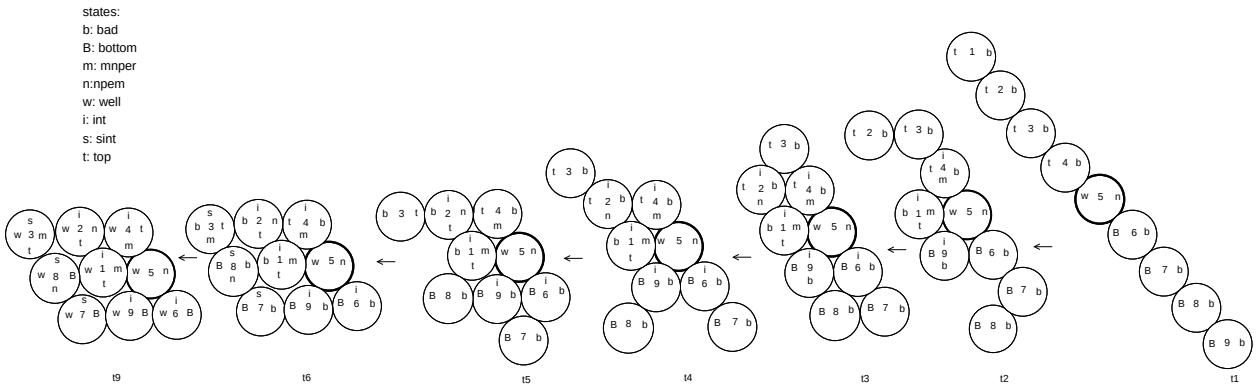

Figure 11: Represents an example of execution of PASC with nine nodes, the initiator is the node 5

and (25), if we remove the states *top* and *bottom* from (18), (19), (24) and (25) the nodes will remain in their position by running (18)/(25) after (25)/(18), or (19)/(24) and (24)/(19) cyclically. Therefore, PASC will not get finished. A variant of PASC with four states *bad* and *well* and *top* and *bottom* is impossible, the reasons are the same used to prove the impossibility with the two states *well* and *bad*.

The thing seen so far, is that we have to add an intermediate state *int* to separate neighboring nodes having the state *bad* and nodes having *well* state. By adding this state, the node that has *int* state can change the C conditions that will be C'. Such a way, B's neighbors cannot change their state to *well* with C', because they are not forming a new correct layer.

Let us suppose a variant of PASC with six states *bad*, *well*, *top*, *bottom*, *int*, and $\neg int$. With five states the deadlock is avoided, and the conditions to change the state to *well* are managed. However, the nodes having the state *int* are making a new layer adjacent to the current correct square $\sqrt{Z} * \sqrt{Z}$, the number of nodes having *int* added is $3\sqrt{Z} + 2$. Therefore, as $\sqrt{Z} * \sqrt{Z} + 3\sqrt{Z} + 2$ is not a square root (from Theorem 4.4 and Theorem 4.5), the shape is not a square. To become a square we have to add $\gamma = \sqrt{Z} + 2$ nodes, these nodes will be at the direction W relative to nodes having the state *int*. These γ can get the state *well* because the shape is a square or an intermediate square. With six states *bad*, *well*, *top*, *bottom*, *int*, $\neg int$, the parallelism will not be optimal and the energy consumption will not be well balanced between nodes. To make an optimal parallelism, PASC makes two rectangles in parallel where the union gives a square. Also, to propagate the state *nper* to the concerned nodes of the middle, we have to use another state *mnper*, the states $\neg nper$ and $\neg mnper$ are used to check if the neighbor node has the state *nper* and *mnper* respectively.

5.4. Complexity of sent messages

PASC needs only $O(N)$ message. That is, the messages of tree construction ($O(N/2)$) and the messages of middle node finding ($O(N/2)$). The most interesting action for message exchange in the algorithm is the one activated by state changing predicates, from the *int* and *bad* states to *well* with the predicates (15), (16) and (17). It is obvious that if node changes its state before it be sure of the good state of other nodes that have moved before it in the current layer, the process will completely go in the opposite direction of the desired objective and self-reconfiguration desired. The predicates (10), (11), 12), (13), (14), (15) and (16) ensure without exchanging of message that the node changes its state only if all nodes that have moved before it have changed their states. Therefore, the first node that begins the construction of the new layer does not need to wait for the message of the first node that began the previous layer. This is because the node that is currently checking the predicates (10), (11), (12), (13), (14), (15) and (16) can have this information by simply consulting (message) the state of its current neighbor. In other words, the message was being sent before the node needs to know the state of its sender, when the node needs to know it, it will find the message at its physical neighbor. So we do not need to transmit information from the node blocked necessarily in a good position with the *well* state to other nodes which are forming the new layer, which explains that throughout the algorithm in any case we do not need to transmit information between two non-neighboring nodes of the new layer. This efficiency is explained by the fact that synchronization in state changing is not required for nodes that are in the same layer. As consequence, PASC needs only the messages of tree construction and the messages of middle node finding.

5.5. PASC and Snap-Connectivity property

PASC maintains the snap-connectivity property since it uses the tree mechanism where only the leaves can move. Since only leaf nodes can move, moved nodes will not generate a hole between nodes connected through it. We can divide the move impact into two categories. A first case happens when no new neighbor is appearing after the movement. In this case there is no *ti* when the message cannot be sent because during the motion it was being always the neighbor of the node used it to move. The second

case appears when the moved node gets a new neighbor. In this case, before it let its neighbor it becomes a neighbor node with another node that is connected since the graph was connected at the beginning, so there is another route for the message of $C_{v,u}$ which will not be blocked for all ti, $i \in \{1, ..., n\}$.

5.6. 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 3 groups (A), (B) and (C) if $n = N$, or into 3 groups, (A), (B), (C') if $N > n$. As shown in figure 12. To apply the functions of prediction, each node will have a level (L), a special number, and eventually a special index. For each group we will give the composition of some functions to predict the number of movements for each node. Notice that, the partitioning procedures are always from the top of the chain to the bottom.

- The size of the group (A) is $|(A)| = \frac{n-\sqrt{n}}{2}$ if n is odd. Or $|(A)| = \frac{n}{2} - \sqrt{n} + 1$ if n is even.
- The size of the group (B) is $|(B)| = \sqrt{n}$.
- The size of the group (C) is $|(C)| = \frac{n-\sqrt{n}}{2}$, if $n = N$ and n is odd. Or $|(C)| = n - \frac{n}{2} - 1$, if $n = N$ and n is even.
- If $n > N$, there will be another group(C'), its size is $|(C')| = \frac{n-\sqrt{n}}{2} + N - n$, if n is odd. Or $|(C')| = \frac{n}{2} - 1 + N$, if n is even.

5.6.1. The case n is odd

For the group (A):

- The first group of nodes $g = 2\sqrt{n} - 3$ take the first level $L1$. The following group of nodes consisting of $g = g - 4$ take the next level $L2$, and the next group of $g = g - 4$ take the next level and so on by subtracting each time 4 from the last g. Figure 13 shows an example.
- The node $i = \left(\frac{n-2\sqrt{n}+3}{2}\right)$ is the first that takes the first index * (called $INDEX_*(i)$).
- The first node that takes the second index # (called $INDEX_\#(i)$) is the node $i = \left(\frac{n-4\sqrt{n}+7}{2}\right)$, the second node that takes the index # is the node $y = \left(\frac{n-4\sqrt{n}+7}{2}\right) - k$, with $k = 2\sqrt{n} - 6$, and the next node that takes the index # is the node $y = y - k$, with $k = k - 4$, and the next node that takes this index is the node $y = y - k$, with $k = k - 4$ and so on by subtracting each time 4 from the last k and subtracting this value from the last y.

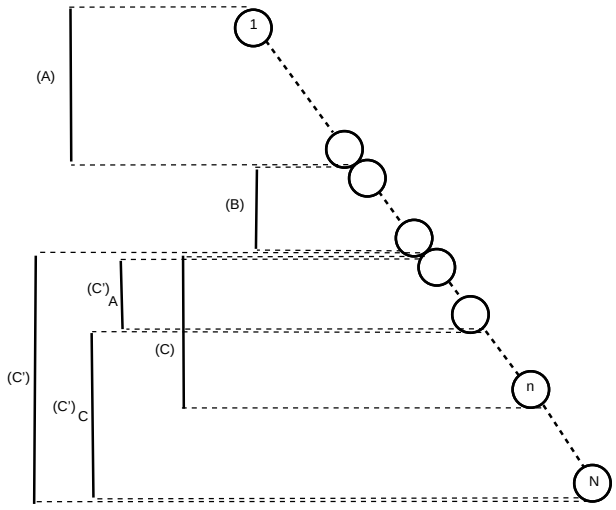

Figure 12: Models the partitioning procedure to calculate the number of movements

- The first node that takes the third index d (called $INDEX_d(i)$) is the node $x = \left(\frac{n-6\sqrt{n}+11}{2}\right)$, the second node that takes the index d is the node $x = \left(\frac{n-6\sqrt{n}+11}{2}\right) - p$, with $p = \left(\frac{4\sqrt{n}-2}{3}\right)$, and the next node that takes the index d is the node $x = x - p$, with $p = p - 4$, and the next node that takes the index d is $x = x - p$ with, $p = p - 4$, and so on by subtracting each time 4 from the last k and subtracting this value from the last x.
- To each level (except the last level) j is associated a number H_j and S_j.

$$H_j = \begin{cases} \frac{3\sqrt{n}-17}{2}, if\ j = 1. \\ H_{j-1} - 3, otherwise. \end{cases} \quad (3)$$

$$S_j = \begin{cases} \frac{\sqrt{n}-3}{2}, if\ j = 1. \\ S_{j-1} - 1, otherwise. \end{cases} \quad (4)$$

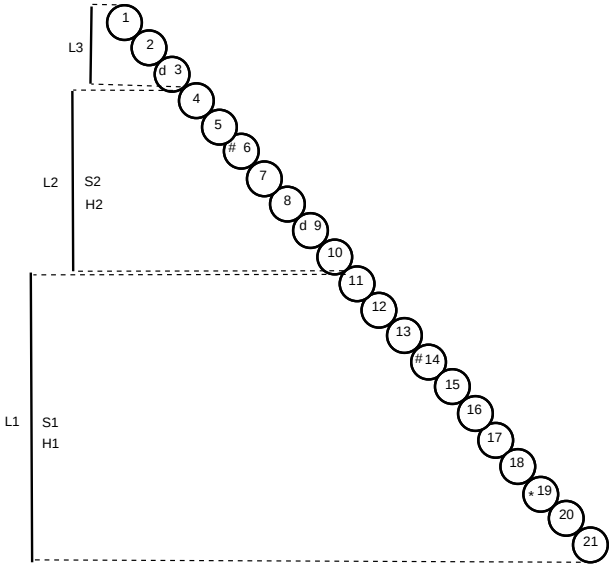

Figure 13: An example of partitioning into levels of nodes having the state *top* and numbers associated to levels with an example of 49 nodes

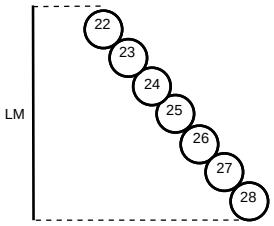

Figure 14: An example of partitioning into levels with the case n=49, the middle nodes have a special level LM, these nodes will not move, they will be of $\sqrt{n}$ size

$$U_{j,i} = \begin{cases} U_{j-1,i+1} - S_{j-1}, if\ L(i+1) \neq j. \\ U_{j,i+1} - H_{j-1}, if\ INDEX_d(i+1). \\ \sqrt{n} - 1,\ if\ i = \left(\frac{n-\sqrt{n}}{2}\right). \\ U_{j,i+1} + 1,\ if\ i+1 = \left(\frac{n-\sqrt{n}}{2}\right). \\ U_{j,i+1} - \left(\frac{3\sqrt{n}-11}{2}\right), if\ INDEX_*(i+1). \\ U_{j,i+1} + 1, if\ INDEX_\#(i+1). \\ U_{j,i+1} + 2, otherwise. \end{cases} \quad (5)$$

With $U_{j,i}$ is the number of movements of node i that has the level j.

For the group (B):

- The $\sqrt{n}$ nodes after the node $i = \frac{n-\sqrt{n}}{2}$ take the middle level called LM. The nodes having this level will not move, their number of movements is 0, (see figure 14).

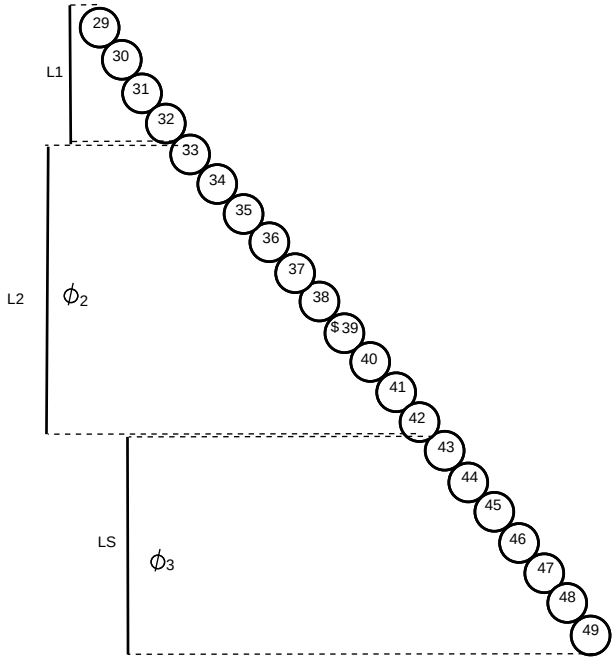

Figure 15: An example of partitioning into levels of nodes having the state *bottom* (group (C)) and numbers associated to levels with an example of 49 nodes

For the group (C):

- The last 7 nodes take the level LS. The first $\left(\frac{\sqrt{n}+1}{2}\right)$ nodes take the first level $L1$. The next $a = 2\sqrt{n} - 4$ nodes take the next level $L2$, and the next $a = a - 4$ take the next level $L3$ and so on, figure 15 presents an example.

- The first node that takes the index \$ is the node $c = \left(\frac{5\sqrt{n}-5}{2}\right)$, the following node that takes the index \$ is the node $c = c + p$, with $p = (2\sqrt{n} - 7)$ and the next is the node $c = c + (p - 4)$ and the next is the node $c = c + (p - 8)$ and so on, figure 15 shows an example.

$$\lambda_j = \begin{cases} \frac{\sqrt{n}-5}{2}, if j = 2. \\ \lambda_j = \lambda_{j-1} - 1, otherwise. \end{cases} \quad (6)$$

$$\phi_j = \begin{cases} \frac{\sqrt{n}+5}{2}, if j = 2. \\ \phi_j = \phi_{j-1} - 3, otherwise. \end{cases} \quad (7)$$

$$Z_{j,i} = \begin{cases} \sqrt{n}, if LM(i-1). \\ Z_{j,i-1} - \phi_j, if L(i-1) \neq Li. \\ Z_{j,i-1} + \lambda_j, if INDEX_D(i). \\ Z_{j,i-1} + 1, if\ i - 1 = n - 1. \\ Z_{j,i-1} + 2, otherwise. \end{cases} \quad (8)$$

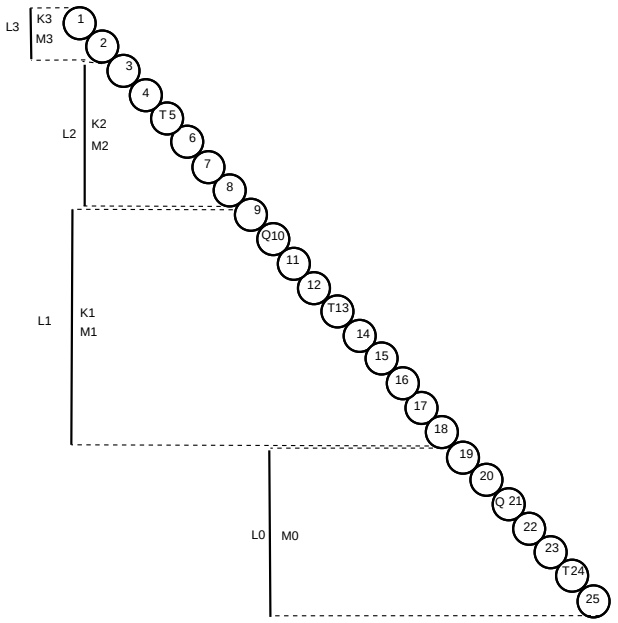

Figure 16: An example of partitioning into levels of the group (A), and numbers associated to levels with an example of $n = 64$ nodes

With $Z_{j,i}$ is the number of movements of node i that has the level j.

For the group (C'):

We partition the group (C') to two subgroups $(C')_A$, $(C')_C$. The group $(C')_A$ contains the first $N - n$ nodes. The following $\frac{n-\sqrt{n}}{2}$ nodes are in the group $(C')_C$. In the following: $(C')_A(i)$ means the node i belongs to the group $(C')_A$.

For the group $(C')_A$:

$$\nu_i = \begin{cases} 2\sqrt{n} - (N-n), if LM(i-1). \\ \nu_{i-1} + 2, otherwise. \end{cases} \quad (9)$$

With ν_i is the number of movements of node i.

For the group $(C')_C$:
The number of movements are predicted with the same instructions (numbers, levels, and indexes) of the group C, with this new function $Z_{j,i}$.

$$Z_{j,i} = \begin{cases} \sqrt{n} + (N-n), if (C')_A(i-1). \\ Z_{j,i-1} - \phi_j, if L(i-1) \neq Li. \\ Z_{j,i-1} + \lambda_j, if INDEX_D(i). \\ Z_{j,i-1} + 1, if, i-1 = n-1. \\ Z_{j,i-1} + 2, otherwise. \end{cases} \quad (10)$$

5.6.2. The case n is even

For the group (A):

- The first $\sqrt{n} - 1$ nodes take the root level (L0).
- The following $a = 2\sqrt{n} - 6$ nodes take the first level (L1).
- The following $a = a - 4$ nodes take following level (L3). And the following $a = a - 4$ take the following level and so on.
- The first node that takes the index Q ($INDEX_Q(i)$) is the node $c = \frac{n}{2} - \frac{3\sqrt{n}}{2} + 1$, the next node that takes the index Q is the node $c = c - p$ with $p = (2\sqrt{n} - 5)$. And the next node that takes the index Q is the node $c = c - p$ with $p = p - 4$ and so on.
- The first node that takes the index T ($INDEX_T(i)$) is the node $b = \frac{n}{2} - \sqrt{n}$, the next node that takes the index T is the node $b = b - e$ with $e = (2\sqrt{n} - 5)$. And the next node that takes the index T is the node $b = b - e$ with $e = e - 4$ and so on. The figure 16 shows an example.

$$K_j = \begin{cases} \frac{3\sqrt{n}}{2} - 7, if j = 1. \\ K_j = K_{j-1} - 3, otherwise. \end{cases} \quad (11)$$

$$M_j = \begin{cases} \frac{\sqrt{n}}{2} - 1, if j = 0. \\ M_j = M_{j-1} - 1, otherwise. \end{cases} \quad (12)$$

$$\chi_{j,i} = \begin{cases} \sqrt{n} - 1, if, i = \frac{n}{2} - \sqrt{n} + 1. \\ \chi_{j,i+1} - K_j, if L(i+1) \neq j. \\ \chi_{j,i+1} - M_j, if, INDEX_Q(i). \\ \chi_{j,i+1} + 1, if, INDEX_T(i). \\ \chi_{j,i+1} + 2, otherwise. \end{cases} \quad (13)$$

With $\chi_{j,i}$ is the number of movements of node i that has the level j.

For the group (B):

- The $\sqrt{n}$ nodes after the node $i = \frac{n}{2} - \sqrt{n} + 1$ take the middle level called LM. The nodes having this level will not move, their number of movements is 0, (see figure 14).

For the group (C):

- The last 7 nodes take a special level LS. The first $w = 2\sqrt{n} - 2$ nodes take the first level $L1$, and the next $w = w - 4$ take the next level $L2$ and so on. To each level is associated an number B_j and O_j.
- The first node that takes the index H (called $INDEX_H(i)$) is the node $c = n - 10$, and the next node that takes this index is $c = c - p$ with $p = 11$, and the next node that takes the index H is the node $c = c - p$ with $p = p + 4$ and so on by adding each time 4 to p, (see figure 17).

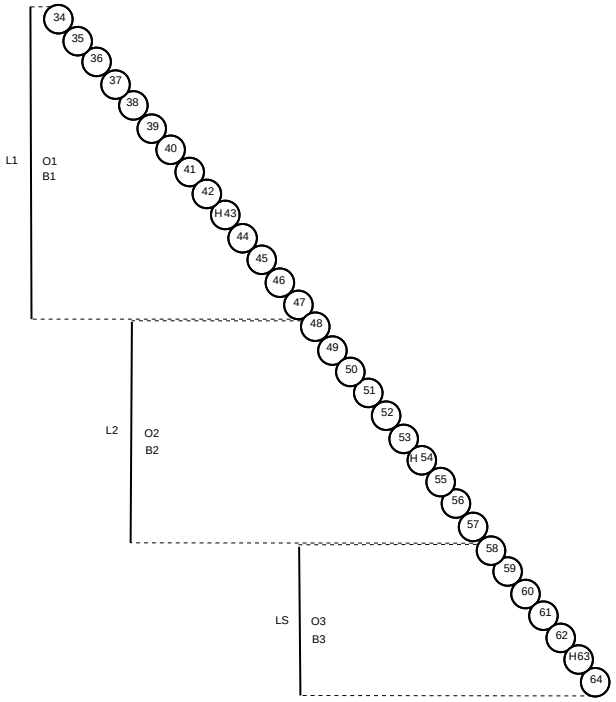

Figure 17: An example of partitioning into levels of nodes of the group (C), and numbers associated to levels with an example of $n = 64$ nodes

$$B_j = \begin{cases} \dfrac{\sqrt{n}}{2} - 2, if j = 1. \\ B_j = B_{j+1} - 1, otherwise. \end{cases} \quad (14)$$

$$O_j = \begin{cases} \dfrac{3\sqrt{n}}{2} - 5, if j = 1. \\ O_j = B_{j+1} - 3, otherwise. \end{cases} \quad (15)$$

$$\xi_{j,i} = \begin{cases} 2, LM(i-1). \\ \xi_{j,i-1} + 1, if i = n. \\ \xi_{j,i-1} - B_j, if INDEX_H(i-1). \\ \xi_{j,i-1} - O_j, if L(i-1) \neq j. \\ \xi_{j,i-1} + 2, otherwise. \end{cases} \quad (16)$$

With $\xi_{j,i}$ is the number of movements of node i that has the level j. For the group (C'):

We partition the group (C') to two subgroups $(C')_A$, $(C')_C$. The group $(C')_A$ contains the first $N - n$ nodes. The following $n - \frac{n}{2} - 1$ nodes are in the group $(C')_C$.

For the group $(C')_A$:

$$\kappa_i = \begin{cases} 2\sqrt{n} + 1 - (N - n), if LM(i-1). \\ \kappa_{i-1} + 2, otherwise. \end{cases} \quad (17)$$

For the group $(C')_C$:
the number of movements are predicted with the same instructions (numbers, and levels, and indexes) in the group (C), with this new function $\xi_{j,i}$

$$\xi_{j,i} = \begin{cases} 2 + (N - n), if (C')_A(i-1). \\ \xi_{j,i-1} + 1, if i = n. \\ \xi_{j,i-1} - B_j, if INDEX_H(i-1). \\ \xi_{j,i-1} - O_j, if L(i-1) \neq j. \\ \xi_{j,i-1} + 2, otherwise. \end{cases} \quad (18)$$

5.7. Generalization of the algorithm

Presented algorithm PASC is specific to a chain case where nodes form initially a straight chain 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 as shown in figure 18. We start by explaining how the initiator nodes is selected whatever is the direction of the chain. For this end, the node with only one neighbor situated either in SW, SE or E direction is chosen as an initiator node. For the other nodes, every node in the chain can deduce the orientation of the chain (one of the three cases represented in figure 18) by analyzing the orientation of its neighbors. For example, if a node corresponds to an extremity node (one neighbor) where the direct neighbor is on the E side, the node deduces that the straight line is oriented E-W. The same thing is happened on the middle nodes, which uses 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 PASC algorithm depending of the orientation $D \in \{W, NW, NE\}$. The variant of PASC algorithm, $PASC^D$, represents an adaptation of the original PASC algorithm (corresponding to $PASC^{NW}$) to the two other possible orientations with changing the directions in predicates. For instance, if the initial chain is oriented NE-SW, the algorithm $PASC^{NE}$ is called, and the square form is realized using moves of type $moveAroundbad_v(u, P_w)$, $moveAroundwell_v(u, P_w)$ and $moveAroundwell_v(u, P_{nw})$. The usage of these three predicates is described in figure 19 that presents an example with nodes having the state *bottom*.

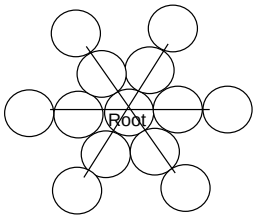

Figure 18: The three possible cases of a straight chain

6. Simulation and comparison

We have done the simulation with the declarative language Meld that uses the DPRSim simulator. In our sim-

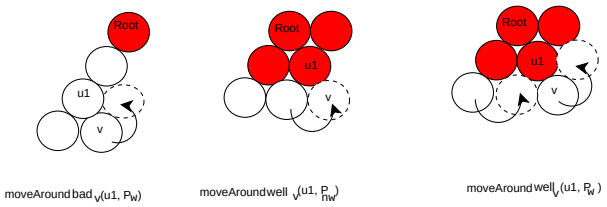

Figure 19: Moves adaptation in the case of NE-SW chain: dark nodes are nodes having the state *well*

ulations the radius of the node is 1 mm[4]. We simulated with a laptop with processor Intel(R) Core(Tm) i5, 2.53 Ghz with 4 GB of memory. The figures 20 and 21 represent an example of execution of PASC.

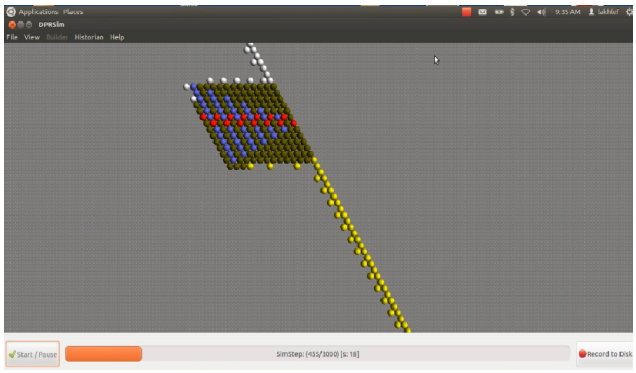

Figure 20: An instance example of execution of PASC, the nodes white-colored are the nodes having the state *top*, the nodes yellow-colored have the state *bottom*, the nodes blue-colored are the nodes having the state *well*, the nodes green-colored have the state *int*, and the nodes red-colored have the state *nper*. We do not show all nodes colored with *well*, otherwise all nodes will have the same color.

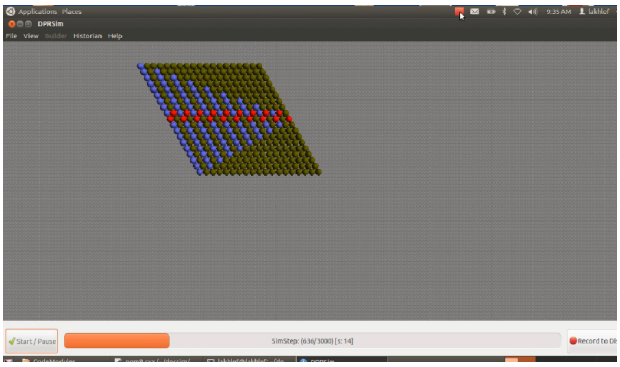

Figure 21: An example of the final execution of PASC

We denote in the figures of simulation: $PASC1$ for the values odd of $n = \left\lfloor \sqrt{N} \right\rfloor \left\lfloor \sqrt{N} \right\rfloor$, and $PASC2$ for the values even of n, with N is the network size. Figure 26 compares also the sub-squares (we call *sub-squares* the intermediate squares generated before the final square) generated based

[4]The time of one movement round depends on the size (the diameter) of the microrobot, as shown in section 4.

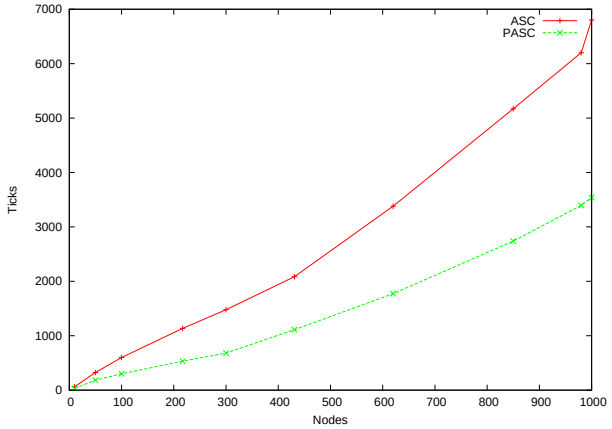

Figure 22: Execution time

on the time with simulation on 1000 nodes. The simulation results come to agree our theoretical results. The nodes applied the procedure of nodes partitioning to levels and predicted with the sequences the number of movements for each node, at the end of the algorithm each node compares the results of prediction (with previous functions) to the results calculated in execution. The figure 22 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 the one given in [35]. Figure 23 presents the highest number of movements found in this paper compared to the one in [35]. With, $g(N) = (\frac{\sqrt{n}}{2}) + N - (\frac{n}{2}) - 1$ and $f(N) = (\frac{n+1}{2}) + N - n$, where $n = \left\lfloor \sqrt{N} \right\rfloor \left\lfloor \sqrt{N} \right\rfloor$. The figure 24 represents the overall number of movements in the networks corresponds to

$$(\sum Z_{i,j}) + (\sum U_{i,j}) \tag{19}$$

or to

$$(\sum \chi_{j,i}) + (\sum \xi_{j,i}) \tag{20}$$

The figure 25 represents the average of the overall number of movements corresponds to

$$\frac{(\sum Z_{i,j}) + (\sum U_{i,j})}{n} \tag{21}$$

or to

$$\frac{(\sum \chi_{j,i}) + (\sum \xi_{j,i})}{n} \tag{22}$$

The effects of parallelism appear well in the curve representing the execution time of PASC compared to the algorithm ASC in [35], as PASC makes two rectangles in the same time. We see that whenever the network size increases the difference increases dramatically. We remark in figure 23 that the number of movements in PASC is much lower, which will increases the probability of lifetime of nodes, therefore, the probability that the node continues its task (its movements). The parallelism has improved

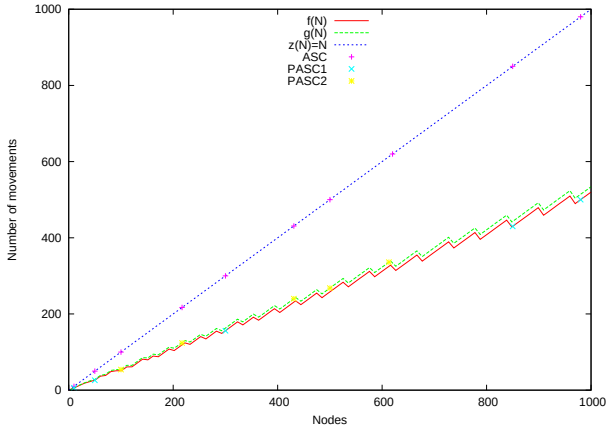

Figure 23: Highest number of movements

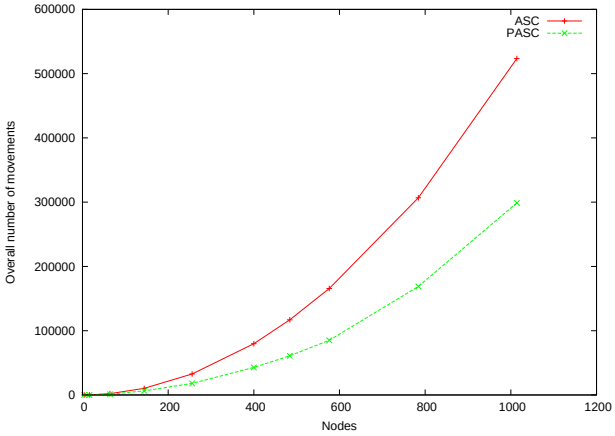

Figure 24: Overall number of movements in the network

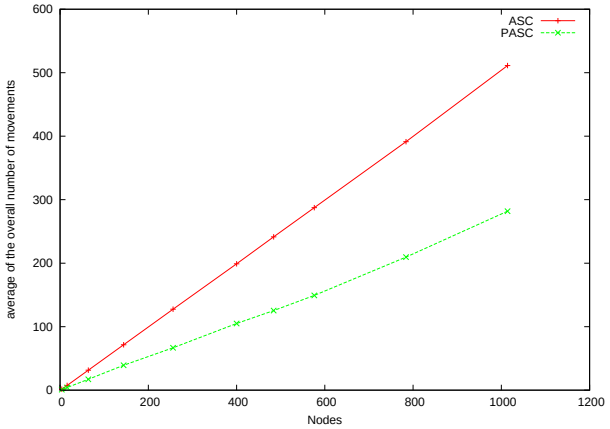

Figure 25: Average of the overall number of movements in the network

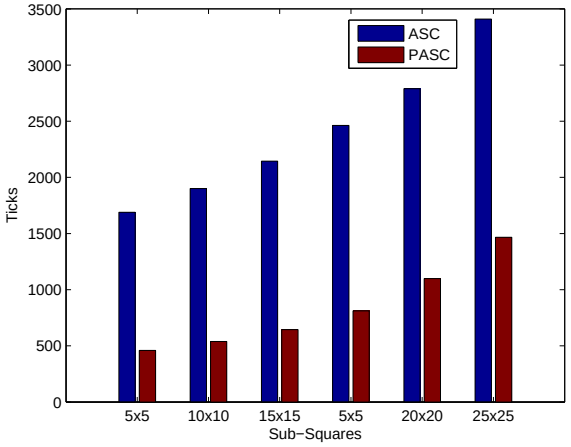

Figure 26: Sub-squares generated based on the time with simulation on 1000 nodes

the overall number of movements in the network, therefore the average of the overall number of movements in the network. However, PASC needs ten states per node and the algorithm *ASC* in [35] needs three states per node, but the complexity remains bounded by a constant in both solutions.
We see in figure 26 that the sub-squares are obtained early compared to the other protocol. This is explained with the fact that in PASC the shape construction is in parallel, so the algorithm takes less time to build each sub-square.

7. Conclusion

In this paper, we proposed an energy-aware parallel self-reconfiguration in MEMS microrobot networks. We have shown the self-reconfiguration parallelized possibility without predefined positions of the target shape, and we presented an algorithm where nodes help each other to achieve the self-reconfiguration using an incrementally process. Our algorithm ensures the connectivity of the network throughout the execution time of the algorithm. Furthermore, each node needs ten states to help and collaborate with neighbors; the algorithm is portable and standalone because it is independent of the map of the target shape. Therefore, it needs a constant complexity of memory, its execution time and highest numbers of movements are much better than that proposed in the previous solutions.

However, some open problems remain. We are studying the conception of an energy-efficient algorithm when the starting form may be any connected shape, in which we predict the loss of these previous characteristics described in this paper (time, number of movements, complexity of memory and message). Another questions remain, the derivation of a fault tolerant algorithm for faulty MEMS nodes is to be investigated. Another problem is to study the effect of self-reconfiguration on the permutation routing [36], where the objective will be to optimize the path of a node to go to the correct position where it finds its correct data.

8. ACKNOWLEDGMENTS

This work is supported 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] S. Hollar, A. Flynn, C. Bellew, and K.S.J. Pister, *Solar powered 10mg silicon robot*, In MEMS, Japan, January 2003.

[2] B. Warneke, M. Last, B. Leibowitz, and K.S.J Pister,K.S.J., 2001, *Smart Dust: Communicating with a Cubic-Millimeter Computer*, Computer Magazine, pp. 44-51, 2001.

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

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

[6] *http://today.duke.edu/2008/06/microrobots.html.*

[7] 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 Proc. of the International Conference on Logic Programming,July, 2009.

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

[9] R. Ravichandran, G. Gordon, and S. C. Goldstein: *A Scalable Distributed Algorithm for Shape Transformation in Multi-Robot Systems,* In Proceedings of the IEEE International Conference on Intelligent Robots and Systems IROS '07, October, 2007.

[10] 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*, IEEE International Symposium on Circuits and Systems, May 2011.

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

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

[13] M. Mamei, A. Roli, F. Zambonelli, *Emergence and Control of Macro Spatial Structures in Perturbed Cellular Automata, and Implications for Pervasive Computing Systems*, IEEE Transactions on Systems, Man, and Cybernetics, 36(5), May 2005.

[14] 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.

[15] 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.

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

[17] E. Juergen and L. Hermann and D. Falko and F. Hannes, *On the Feasibility of Mass-Spring-Relaxation for Simple Self-Deployment*, 8th IEEE/ACM International Conference on Distributed Computing in Sensor Systems,203-208, China,2012.

[18] R. Soua, L. Saidane, P. Minet, *Sensors deployment enhancement by a mobile robot in wireless sensor networks*, IEEE ICN 2010, Les Menuires, France, April 2010.

[19] K. Stoy, R. Nagpal, *Self-Repair Through Scale Independent Self-Reconfiguration*, Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robotsn and systems, japan, 2004.

[20] K. Kotay, D. Rus, M. Vona, and C. McGray, *The Self-reconfiguring Robotic Molecule*, in Proceedings of IEEE International Conference on Robotics and Automation, Leuven, 1998.*

[21] S. Wong and J. Walter, *Deterministic Distributed Algorithm for Self-Reconfiguration of Modular Robots from Arbitrary to Straight Chain Configurations*, IEEE International Conference on Robotics and Automation, Germany, May 2013

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

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

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

[25] 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

[26] W. Shen, P. Will and A. Galstyan, Hormone-inspired self-organization and distributed control of robotic swarms. Autonomous Robots 17(1), 93-105, 2004.

[27] H. Mabed, H. Lakhlef, J. Bourgeois *Fully Distributed Redeployement Algorithm for Multi-Robot System.In: 6th Int. Conf. on NETwork Games, COntrol and OPtimization*, NetGCooP'12. IEEE Computer Society, Avignon, France, 2012.

[28] K. Stoy, *Using cellular automata and gradients to control self-reconfiguration*, Robotics and Autonomous Systems 54(2), 135-141, 2006.

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

[30] J. Walter, J. Welch, and N. Amato, *Distributed reconfiguration of metamorphic robot chains,* Springer-Verlag Journal on Distributed Computing, vol. 17, pp. 171-189, 2004.

[31] J. Walter, B. Tsai, and N. Amato, *Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots,* IEEE Transactions on Robotics, vol. 21, no. 4, pp. 621-631, 2005.

[32] T. Larkworthy and S. Ramamoorthy, *An efficient algorithm for self-reconfiguration planning in a modular robot,* IEEE Intl. Conf. Robotics and Automation, 2010, pp. 5139-5146.

[33] 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 2008 International Conference on Intelligent Robots and Systems, September, 2008

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

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

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

[37] Lakhlef H, et al. *An Energy and Memory-Efficient Distributed Self-reconfiguration for Modular Sensor/Robot Network*, Journal of Supercomputing (2014), DOI 10.1007/s11227-014-1196-8

[38] Lakhlef H, et al. *Optimization of the logical topology for mobile MEMS networks*, journal of Network and Computer Applications (2014), shttp://dx.doi.org/10.1016/j.jnca.2014.02.014

[39] *The physical rendering simulator (dprsim), http://www.pittsburgh. intel-research.net/dprweb.*