# Decentralized Approach to Evolve the Structure of Metamorphic Robots

Tarek ABABSA, Noureddine DJEDI
LESIA Team, Computer Science Department
University Mohamed Khider
Biskra, ALGERIA
ababsatarek@yahoo.fr , n.djedi@univ-biskra.dz

Yves DUTHEN, Sylvain CUSSAT BLANC
Vortex Team (IRIT)
University Toulouse 1-Capitole
Toulouse, FRANCE
Yves.Duthen@univ-tlse1.fr , sylvain.cussat-blanc@irit.fr

*Abstract*—**Metamorphic robots are robots that can change their shape by reorganizing the connectivity of their modules to adapt to new environments, perform new tasks, or recover from damages. In this paper we present a decentralized method for structural evolving of a class of lattice-based simulated metamorphic robots in a static environment. These robots are considered as a set of crystalline (compressible) modules that are able to connect or disconnect one from each another or even exchange information and energy with the neighbor modules in order to form various structures/patterns dynamically. Our approach is spited in two layers: in the first layer a genetic algorithm is used to generate a number of well suited target configurations based on current information perceived from environment, while in the second layer a PacMan-like algorithm is used to make a plan for modules movement to transform the robot from its current pattern to the target pattern emerged in first layer.**

***Keywords— Metamorphic Robots; Multi-Cellular Structures; Self-Configuration; Genetic Algorithm; PacMan Algorithm.***

## I. INTRODUCTION

A metamorphic robot is a robot that consists of a large number of autonomous units able to self-organize into an entire structure (morphology) better suited to the environment in which the robot is deployed, or to a specified task that is better accomplished by a specified morphology. The units are a set of mechatronic modules that are able to connect or disconnect one from each another or even exchange information and energy with the neighbor modules.

The locomotion of several modules from position to position over their neighbors changes the whole shape of the robot [1,2,8,11,12]. For example, a self-configured modular robot may reconfigure itself into a thin, linear shape to facilitate passage through a narrow tunnel, transform into an emergency structure such as dam, shield, bridge, or even surrounding and carrying objects.

Metamorphic robots are usually constituted from identical units that have the same physical structure and able to do computational and communication functionalities. These units must have also enough degrees of freedom to be able to move over neighbor units so that gives the whole robot the ability of self-reconfiguration and the opportunity to self-repairing by replacing damaged units with operational ones [2].

What makes metamorphic robots more attractive in the research field of artificial life than the robots with fixed morphology is that in addition to the perception, the actuation and the control abilities of the robots with fixed morphology, a metamorphic robot is also able to adapt its shape through units reorganization and the updating of their connectivity in order to survive in new situations, supports a new mechanical operations or even recover from failure. Some potential applications of metamorphic robots include:

- Navigation and obstacle avoidance in a massive constrained environments or strongly unstructured environments.
- Studying evolutionary structures that consist of autonomous modules.
- Object surrounding and isolation for microscopic manipulations.

Modular self-reconfigurable robotic systems that are composed of many modules are expected to have the following properties:

a) Versatility: metamorphic self-reconfigurable robots are potentially more adaptive to the environment than conventional robots thanks to the ability to reconfigure modules which allows the robot to change its morphology according to its tasks or its current special situation.

b) Robustness: since each unit of the robot is able to move over neighbor units, the whole robot can also autonomously expel faulty modules outside the robot structure leading to what we call self-repairing.

c) Low Cost: Self-reconfigurable robotic systems may substantially reduce robot cost thanks to the self-reconfiguration that can make a range of complex machines from one set of modules.

The versatility in potential capabilities of self-reconfigurable modular robot is influenced with degrees of

freedom added to its modules, but also increases mechanical and computational complexities.

The advantages of metamorphic self-reconfigurable robots have not yet been fully realized, thus their interest is not only when we consider multiple tasks that require passing from several morphologies or when the working environment is not fully determined. The field of metamorphic robots addresses the design, the manufacturing, the motion-planning and the control of autonomous machines with variable morphologies, this type of machines is much desired to do operations in a not fully-known environments.

The objective of our work is to develop a decentralized method that can evolve the structure of a crystalline based metamorphic robot using the computing capacity of each module of the system. Experimenting this method within a metamorphic robot whose goal is to surround a known target object dropped in the environment, the task requires a set of reconfiguration to the robot shape in order to emerge water-flow like behavior to go through a tunnel that separates the target object position and the initial position of the robot. Our approach is spited in two layers: the first layer uses a genetic algorithm in each module of the system to randomize and evolve a population of the system reconfigurations while the second layer is charged to broadcast the best configuration over the system modules and next uses a PacMan-like Algorithm to establish the reconfiguration.

## II. RECONFIGURATION APPROACHES OF MODULAR ROBOTS

Robot assembly and metamorphosis is part of a robotic challenge defined in the FP7 project Replicator where the main problem in this type of systems is the motion planning of the modules and their reorganization, in other words how to transform a given configuration (X) into a target configuration (Y) only using legal moves for each module of the system [10,11,12,13, 17].

Several centralized and decentralized researches had been the subject to deal with this kind of problems; the most of these researches may be classified on the two axes described thereafter.

### *A. Centralized and decentralized complexity of self-reconfiguration*

In the centralized self-configuration approach, the sequence of legal reconfiguration moves for each module is determined by a computational model that is performed by only a central processing unit; these moves are used to transform the initial configuration of the robot into a target configuration.

According to [1], the experimentation of self-configuration with this strategy requires $O(n^2)$ computation time, while the amount of computation performed by each module in a distributed implementations is $O(n)$. In the distributed self-configuration, the computation is not performed by a central processing unit, because all the units have the same description and every unit is involved in the computation process.

### *B. Transition Graph Based Reconfiguration (TGBR)*

Robot metamorphosis is presented at the most abstract level as a reconfiguration problem using a graph rewriting grammar to generate the robot shapes (configurations). The idea of this centralized approach is to build a transition graph where the nodes represent possible configurations of the whole robot and the edges represent the elementary moves required to transform the global structure from configuration ($C_a$) to the configuration ($C_b$) where {a,b} are two states of the system [8,11,12, 17]. The temporal dynamics of the system are well described by this approach; however its implementation is very expensive in memory and time computation when used to the systems that have a large number of modules (leads to a combinatorial explosion).

n = number of modules.
C(n) = number of possible configurations.
G=(V,E) / E={a,b,c,d,e,f,g,h,i,j,k}

V= {
(a,k),(a,j),(a,b),(a,d),
(b,h),(b,i),(b,e),(b,a),
(c,f),(c,e),(c,h),(c,i),
(d,i),(d,e),(d,h),(d,a),
(e,d),(e,g),(e,k),(e,b),(e,j),(e,c),
(f,c),(f,k),(f,j),(f,g),
(g,i),(g,e),(g,f),(g,h),
(h,g),(h,k),(h,c),(h,b),(h,j),(h,d),
(i,g),(i,b),(i,c),(i,d),
(j,a),(j,h),(j,f),(j,e),
(k,f),(k,h),(k,a),(k,e)}

| n | C(n) |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 11 |
| 4 | 44 |
| 5 | 186 |
| 6 | 814 |
| 7 | 3.652 |
| 8 | 16.689 |
| 9 | 77.359 |
| 10 | 362.671 |
| 11 | 1.716.033 |
| 12 | 8.182.213 |

Fig. 1. Left: TGBR of three hexagonal modules (n=3), Right: Number of possible configurations C(n) depending on the number of modules (n).

For example, if we consider the case of the hexagonal modules detailed in [11], the number of possible configurations is given in the table of figure 1. Within this approach, we can see that the size of the graph is exponential to the number of the modules that makes the robot (see the table of figure 1). The left part of the figure 1 shows the TGBR formal definition of three hexagonal modules, where the vertices {a,b,...,k} are the possible states (possible configurations shown in figure 2) of the system and the legs are the system transitions from a state to another.

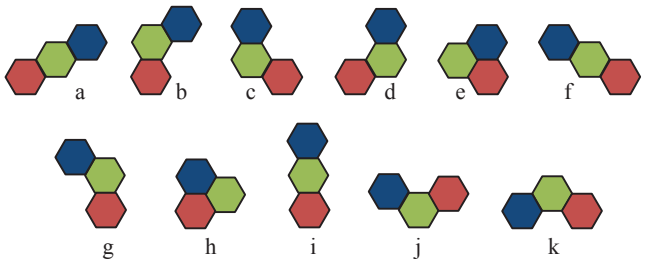


Fig. 2. Possible configurations using three hexagonal modules. The corresponding TGBR is shown in figure 1 (left).

An improvement of TGBR approach is done later by Ulrik P. Schultz in [8]: the author used an approach based on distributed finite state automata to represent all possible configurations. A copy of the automata is distributed to all

modules of the robot, but each module runs only the states it has to address. The role of the finite state automata is to switch the control over modules thanks to a communication protocol used to share active states and the identity of the modules selected to run these states.

### C. Heuristics Based Reconfiguration

Murata et al. built a self-configurable machine [13]; in fact this machine is a real reconfigurable modular robot. The reconfiguration of this machine is done by random locals locomotion of its modules. The implementation of this approach is easy because it doesn't require much parameters, however the convergence is slow because it uses a random strategy for motion planning, therefore this approach is more convenient and can't be used for a variety of complex problems.

Gregory S. Chirikjian [2] tried out self-reconfigurable robots in hexagonal lattice environment where each cell of the lattice has only one of the three states (1) empty, (2) occupied by a single module, (3) occupied by an obstacle. The author defined a geometric quantity (d) to measure the distance between two cells of the hexagonal lattice and it is used to calculate the rate of modules attraction toward the target. This quantity is calculated as following:

1. $d(A,B)>0$
2. if A=B then $d(A,B) = 0$
3. $d(A,B) = d(B,A)$
4. $d(A,B) + d(B,C) >= d(A,C)$

The evolution of structure configuration is based on the motion planning to move modules from one position to another by the fact that the target acts as an attractor of the modules. The modules in this experimentation are influenced by an artificial potential field that makes every module feeling artificial force to move, this force is inversely proportional to the distance between the module and the target [2].

A. Pamecha et al. used Chirikjian's model to develop a reconfiguration algorithm based on simulated annealing in order to improve the quality of reconfiguration by increasing the degree of similarity between initial and target configurations, the higher and lower bounds of steps required for the reconfiguration are also formulated [14].

D. Rus and M. Vona [15] developed a centralized planning algorithm called "the melt-grow planner" to control self-reconfiguration of metamorphic robots by using a muscle-like actuation mechanism. This algorithm was developed for systems with unit-compressible modules, such as the crystalline robots.

Zhang et al. [16] developed several heuristic algorithms to solve planning of parallel locomotion of modules for modular robots.

Z. Butler and S. Byrnes introduced PacMan algorithm in [9] as an efficient approach to plan and fulfill self-reconfiguration of crystalline robots. The efficiency of this approach is due to its ability to be parallelized. In fact the PacMan algorithm gives to each module of the system the opportunity to perform computation and planning requirements to make parallel self-reconfiguration from initial configuration to the target one using only local communications between modules.

## III. DISCOVERING THE TOPOLOGY OF LATTICE-BASED METAMORPHIC ROBOTS

The structure topology of metamorphic robots is defined by both the spatial location of each module and the physical links between modules.

In the first stage of our approach, we need to discover the initial robot topology by distributing the perceived information over all the units to emerge a global vision of the whole structure and initialize the morphology generator (see section IV) that will evolve this structure into a more suitable one.

The structural discovering must only uses local communications; therefore the modules have to keep the whole structure in a connected state (no fragmentation must be occurred) during all the time steps of the simulation.

In this work we consider the crystalline units to form our metamorphic robot where the number of units is known in advance.

Each module (i) has an adjacency matrix ($M_i$) with a size of (n*n) where (n) is the number of modules forming the robot. To discover the robot topology, modules use message propagation mechanism based on two primitives of asynchronous communication "Send($M_i$, Destination)" and "Receive()" (figure 3) to achieve the next constraints:

- The local vision (perception) of the module (k) is encoded in an adjacency sub-matrix that encodes the next three values : $M_k[i,j] = 0$ if cell (i,j) is empty, $M_k[i,j] = 1$ if cell (i,j) is occupied by a module and finally $M_k[i,j] = 2$ if cell (i,j) is occupied by obstacle.
- Each module perceives the environment to get local vision about both environment and robot morphology by extending free arms of the module. If the extension is done without physical contact then the first cell in the same direction is empty, otherwise the physical contact identify if there is a module or an obstacle.
- If sub-matrix $M_k$ that encodes the local vision of the module (k) is updated then the module should send the updated matrix to neighbor modules.

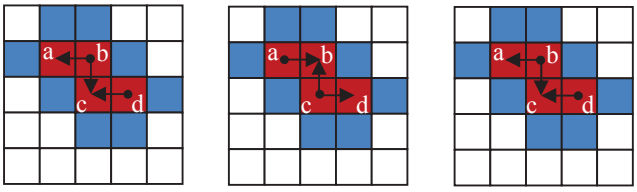


Fig. 3. Propagation of local perception over modules of the metamorphic robot. The red cells represent 4 connected modules, while the blue cells represent empty cells perceived by the modules and the white cells represent the unperceived cells in the environment.

| | | |
|---|---|---|
| b: Send({a,b,c}, {a,c}) | a: Send({a,b,c}, {b}) | b: Send({a,b,c,d}, {a,c}) |
| d: Send({d,c}, {c}) | c: Send({a,b,c,d}, {b,d}) | d: Send({a,b,c,d}, {c}) |
| Update(Ma,Mc) | Update(Mb,Md) | Update(Ma) |
| a{a,b,c} | a(a,b,c) | a(a,b,c,d) |
| b{a,b,c} | b(a,b,c,d) | b(a,b,c,d) |
| c{a,b,c,d} | c(a,b,c,d) | c(a,b,c,d) |
| d{c,d} | d(a,b,c,d) | d(a,b,c,d) |

The next algorithm shows the implementation of these constraints to ensure the propagation of local perceptions over the modules.

The bordered code segments must be executed in mutual exclusion because of the shared variables.

**ALGORITHM Discover_Topology**

**Foreach** module (i) in the system **do**

Create new process (**$P_1$**) with the next segment code:

$Update_i \leftarrow$false;

**While** true **do**

$M \leftarrow$Receive();

**if** $M_i - M \neq 0$ **then**

$Mi \leftarrow M_i + M$;

$Update_i \leftarrow$true;

**End if**

**End While**

**End Foreach**

**Foreach** module (i) in the system **do**

Create new process (**$P_2$**) with the next segment code:

Fill the matrix $M_i$ with neighbor modules;

$Update_i \leftarrow$true;

**While** $M_i$ is not fulfilled **do**

**if** $Update_i$ = true **then**

Send($M_i$, list of neighbor modules);

$Update_i \leftarrow$false;

**End if**

**End While**

**End Foreach**

This algorithm explains the dynamic of the information propagation over the modules, in fact the role of the process **$P_1$** injected in module (i) is to receive local information "adjacency sub-matrix" sent by the process **$P_2$** injected in module (j ; i≠j) and cumulates this information to have a richer vision about the whole robot. While the role of the process **$P_2$** is to detect changes in the local vision established by the process **$P_1$**, and if there are changes in local vision then the module must inform all its neighbors.

From this mechanism emerges a global vision of the metamorphic robot and gives each module of the system the ability to have a full description of the robot's topology and its surrounding environment using only a set of simple local communications.

## IV. DECENTRALIZED APPROACH TO EVOLVE METAMORPHIC ROBOT STRUCTURE

In this paper we have adopted a decentralized approach to evolve the structure of metamorphic robots. It consists of a set of autonomous units that have the same properties, so the nature of the system does not imply the existence of a supervisor module: all modules have the same functional level and they can all participate in the evolution of the whole structure.

In the second stage of our work, we propose to use an "online" evolutionary search of the next configuration using a genetic algorithm and the robotic modules as parallel computers. Looking to the exploitation of the computing capacity of each module of the system, we attempt to parallelize the genetic algorithm in order to reduce the computation time. The genetic algorithms tend to be easy to parallelize thanks to their structure.

Different version of parallel genetic algorithms had been the subject of several studies thanks to the appearing of parallel machines and computer networks.

Mainly there are two models of parallel genetic algorithms, master-slave model and island model [4]. In our work the results are obtained using our proper framework developed in JAVA initially to parallelize Ray-Tracer Algorithm. This framework makes a heterogeneous computer network (where each machine runs java virtual machine) looked as a supercomputer with a shared memory and multiple processing units that can run a set of parallel algorithms (the efficacy of our framework is not the subject of this work).

A copy of the same genetic algorithm is implemented in each unit of the system to perform its distribution following the island model, where the initial population is divided into several sub-populations that are processed separately by a set of interconnected computing units.

The master process creates (n) occurrences of the same process that performs a copy of the genetic algorithm, progressing in separate machines within the required parameters as it is shown in figure 4. Each process evolves its local population independently until it decides (according to a predetermined criterions) to migrate some best genome (proportional to the size of the local population) into a selected process. The receiver process adds the new genome to its population and eliminates the worst ones (the size of local population must be respected).

This strategy of parallelization makes it easy to perform a genetic algorithm within a large population in size, and gives result in a reasonable duration.

It is observed that a proper size for sub population must be correctly selected because a subdivision in sub-populations of a too small size leads to non-reliable genetic algorithms. Indeed a population must also contain enough diversified genomes so that the search space can be well explored and the result returned is more interesting.

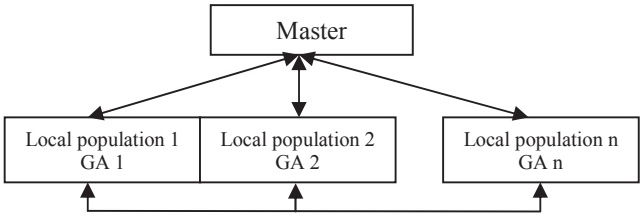


Fig. 4. Island architecture to parallelize Genetic Algorithm, $\sum_{i=1} LocalPopulation_i = T$

To reduce the system complexity, we consider a static environment modeled by a lattice of 2D cells, where each cell has the interior architecture shown in figure 5 and may be in one of the following states:

1) ***Empty***: that can be filled by a single module as it goes along.

2) ***Occupied by a module***: and it become empty if the inside module moves into one of the neighbor cells.

3) ***Occupied by an obstacle***: while considering static environment, this state remains unchanged during all time steps of the simulation.

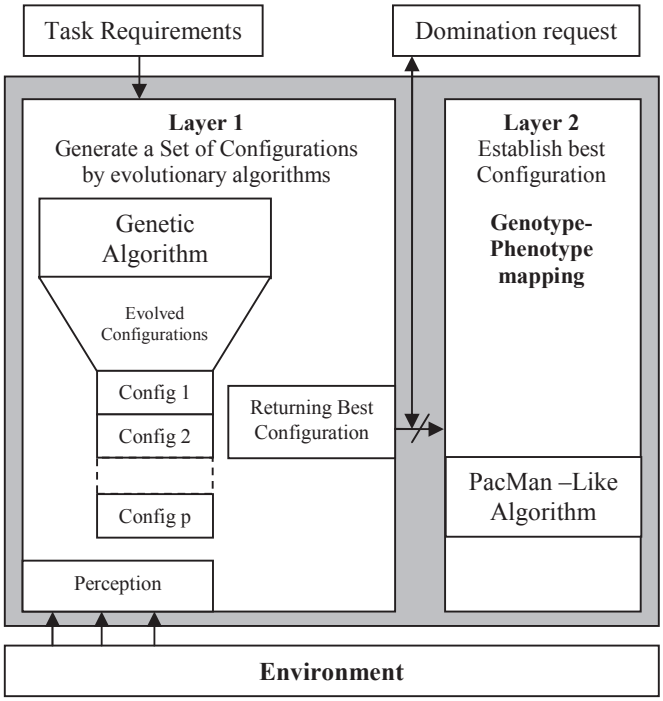


Fig. 5. The diagram of our evolutionary approach.

In the present work, the proposed approach is spited in next three stages: evolving modules configuration, domination of new structural information and reconfiguration to the new pattern.

*A. Evolving modules configuration using Genetic Algorithm*

In this stage, each module uses its computational capacity to run a distributed genetic algorithm to search the next configurations better adapted to the environment where the best one is the configuration that maximizes the fitness function (in this work the fitness function is defined as the euclidean distance between the center of gravity of the robot and the target object that has to be surrounded).

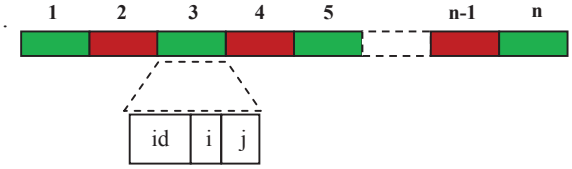


Fig. 6. Genome structure that represents a configuration of (n) modules.

Initially, each module of the system must have the following genetic information: initial population where the genomes encode both the links between modules and the position of each module in the environment. A fitness function is defined to calculate the importance of each genome. A model of the genome is shown in the figure 6.

Considering a metamorphic robot of (n) modules, a set of genomes with a size of (n) genes are randomly created and diversified as much as possible. Each genome should also be well-formed (it must encode unfragmented robot structure) where each gene of the genome contains the next two integer fields:

- Identifier field (id): to identify each module of the system.
- Discret coordination field (i,j): to encode the discret positions in the lattice based environment.

The links between modules are deduced by using neighbor rules (all the neighboring modules are linked together).

The genetic operations (crossover, mutation) are applied only for discrete coordination field, while the identifier field remains always unchanged.

*B. The domination of new structural information*

After a number of iterations, each module of the system retrieves the best genome of its local population, this genome encodes the current best configuration evolved by this module. A message of domination request that contains this genome is created and sent to the neighbor modules that will process this message using the fitness function to measure the importance of the genome encapsulated, and in particular they must deal with the following five situations:

1) If the extracted genome has a higher fitness than the best one in the local population then, first suspend the genetic algorithm progressing inside the module, next add this genome to the local population and diffuse the same message to neighbor modules that will do the same operation. At this moment each sender module must wait for the acknowledgement answer from the modules contacted in order to answer positively to the domination request of the initiator.

2) If the fitness is smaller, ignore the received message and create a message of "genome migration", in which the best genome of the local population is encapsulated. Next, respond negatively to the domination request with this

message. This operation is executed without suspending the progress of the genetic algorithm.

3) If the answer to the domination request is negative, then extract the encapsulated genome from the received message (this genome is the best of the local population at the moment of the request repercussion), add the genome to the local population, and send the acknowledgment message to the initiator module (module initially requests the domination).

4) If (n-1) acknowledgment messages are gathered within a particular module then the module must report the domination of the new configuration using propagation of the dominated message to allow the modules startingthe reconfiguration stage.

If a module receives a message of domination (this message contains the geometric description of the conventional best configuration) then it cancels the processing of other messages, sends this message to the neighbor modules, and starts the reconfiguration process.

### *C. Reconfiguration to the target pattern*

The reconfiguration to the new pattern is the last stage of our approach. At this stage, we used a PacMan-Like algorithm [9] as it is a parallel planning algorithm used to reconfigure crystalline robots.

In fact, the PacMan algorithm was inspired by the video game of the same name; this algorithm is parallelized and reused by Zack Bulter et al [9]. It uses a specific data structures called "pellets" as a way of marking the path that each module should follow to perform its part of reconfiguration. An example of application of the PacMan algorithm is illustrated on figure 7. Once the pellets are distributed, the modules start their asynchronous virtual locomotion where each module switches its identifier with the neighbor module found in the direction along its path of the reconfiguration. The moving module must also consume pellets of its identifier as shown in the next PacMan algorithm.

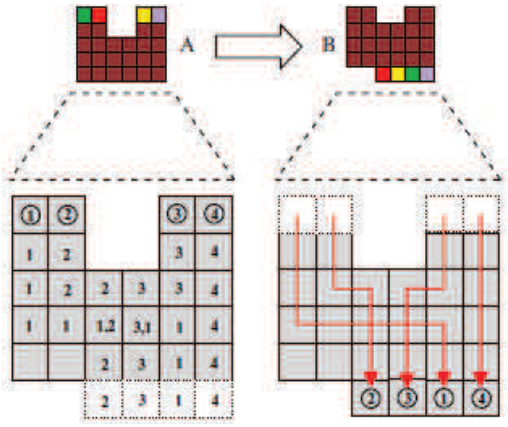


Fig. 7. An example of using PacMan algorithm to transform the configuration (A) to the configuration (B).

The figure 6 shows that the PacMan approach lets several modules to do simultaneous moves, which prevents the occurrence of a deadlock or structure fragmentation. To recover this problem, the following constraints must be applied:

- The switching of identifiers between modules must be executed in mutual exclusion.
- At the end of its displacement, each module must diffuse a message that indicates the end of PacMan algorithm to all the modules, to inform them that it successfully reaches its final destination.

Each module of the system starts again stage A (Evolving configuration by GA) if the number of the received messages that indicate the end of the PacMan algorithm equals to the total number of the modules.

**ALGORITHM** PackMan

**Foreach** module (i) of the system **do**

**if exist** neighbor(i) that have the same identifier pellet **then**

Select this module

**if** module (i) is contracted **then**

- Switch identifier of module (i) with the selected module.

**else**

- Contract module (i) toward the selected module.
- Switch identifier of module (i) with the selected module.

**End if**

**End if**

**if** module (i) is contracted **then**

**if** module (i) is in final position or it has just crossed a module (j) **then**

- Extract module (i) toward the direct of the module (j).

**End if**

**End if**

**if** module (i) reaches the final position **then**

- Diffuse the "end of PacMan Algorithm" message to all modules.

**End if**

**End Foreach**

The bordered code segments must be executed in mutual exclusion to avoid deadlock. The following algorithm shows a method for spreading pellets over the modules of the system.

**ALGORITHM** Pellets-Spreading

**Foreach** module (i) of the system **do**

**if** module (i) is not in the target configuration **then**

- Establish the locomotion path of the module (i) to move from current position to the proper position in the target configuration using Dijkstra algorithm to

perform the shortest path in order to reduce the cost of the reconfiguration.

- Each path is modeled using representation by successors and the models are pushed on a stack where each field has two sub-fields (the identifier of the moving module and its successor that represents the direction of pellets spreading).
- Put a unitary quantity of pellets coupled with the identifier of the stack-owner module removed from the top of the stack, next send the remaining stack to the successor module.

**End if**

**End Foreach**

## V. RESULTS AND DISCUSSION

Initially, the crystalline units are gathered in an entire connected structure (no fragment must be occurred), where each module has to perform a genetic algorithm in order to randomize and evolve the whole structure.

To ensure the autonomy of modules, we used a grid of 4x4 computers to perform the computation of the parallel genetic algorithm, where each machine acts as the computing unit of a module. A mechanism of communication inter-modules is implemented using the SOCKETs of BERKLEY.

The following parameters are used to get the results shown in figure 8:

- **Selection**: Tournament selection.
- **Mutation rate**: 5%.
- **Crossover rate**: 60%.
- **Sub-population size**: 60 genomes.
- **Migration**: 5% from the size of sub-population.

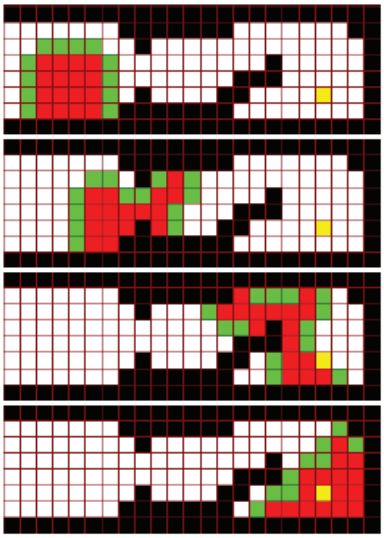

Fig. 8. The evolution of a metamorphic self-reconfigurable robot during its movement in a tunnel from left to right to surround the yellow square.

To demonstrate our approach, we assumed that the robot should surround an object on the environment.

Knowing the object position we define the fitness function as the distance (D) between the object and the center of gravity of the robot. So the quantity (D) must be minimized as much as possible to ensure the best surrounding of the object.

The environment is simulated as a 2D lattice matrix composed from 23x8 cells as shown in figure 8, where 23 cells represent obstacles (black cells) and the metamorphic robot is represented by 4x4 red cells. The green calls represent the perceived cells while the yellow cell represents the object to be surrounded.

Figure 8 shows instances of simulation of the metamorphic robot evolution that moves from left to right while going through a tight tunnel to surround the object represented by the yellow square.

During this simulation, we can clearly observe in all time steps the changes in the morphology of the robot while doing its evolution so that it can adapt to the environment to achieve its goal. This evolution is emerged by the co-operation of all the modules that evolve population of possible configurations and make an agreement (using domination request) to apply the best one.

We can also observe a kind of directed locomotion behavior of the whole structure emerges from the successive configurations of the metamorphic robot. This locomotion is expected because the search space of the genetic algorithm integrates and disintegrates dynamically the perceived cells for each configuration which create not only a local vision for each module (the green cells represent the vision field of the modules) but also a local vision of the whole evolutionary structure(by propagating information) thus the emerging behavior is well-adapted to the environment because the evolved structure is strongly influenced by the information perceived from the environment.

## VI. CONCLUSION AND PERSPECTIVES

In this paper, we presented a decentralized approach that evolves the configuration of metamorphic robots composed from crystalline modules.

This approach mainly uses PacMan-like algorithm coupled with a genetic algorithm.

In fact the local information propagation over modules emerges a full-description of the structural topology of the metamorphic robot in each module. This description is indispensable to perform a distributed genetic algorithm using modules as computational units, because each unit of the system is an autonomous module that has a computational capacity though limited but it is sufficient to perform genetic operations “crossover, selection, mutation” because these operations are relatively simple.

This first stage invokes an evolutionary process that is distributed over the modules of the robot using the information perceived from environment and shared between these modules. The second stage uses a reconfiguration algorithm

(PacMan algorithm) to change the robot structure from the current configuration to the one defined in the first stage.

The evolutionary algorithm used in this work is decentralized, which ensures theoretically the continuity of the evolutionary process even if there are faulty modules.

A directed movement behavior emerges from the different reconfigurations of the metamorphic robot: this behavior can be studied in a future work to get the most possible hidden potential of evolutionary structures.

## REFERENCES


[1] G. Aloupis & al, "Linear Reconfiguration of Cube-Style Modular Robots". ISAAC'07 Proceedings of the 18th international conference on Algorithms and computation, pp. 208-219, 2007.

[2] G. S. Chirikjian, "Kinematics of a metamorphic robotic system". In IEEE International Conference on Robotics and Automation Proceedings, Volume 1, pp. 449-455, 1994.

[3] S. Cussat-Blanc, H. Luga & Y. Duthen, "Artificial Embryogeny and Grid Computing", Genetic and Evolutionary Computation (GECCO 2008), Atlanta, July 2008.

[4] A. Bertoni & M. Dorigo. "Implicit parallelism in genetic algorithms". Artificial Intelligence, 61(2), pp. 307-314, 1993.

[5] A. Bethke, "Comparison of Genetic Algorithms and Gradient-based Optimizers on Parallel Processors": Efficiency of Use of Processing Capacity. The University of Michigan, College of Literature, Science, and the Arts, Computer and Communication Sciences Dept, 1976.

[6] J. Grefenstette, "Parallel adaptive algorithms for function optimization". Vanderbilt University, Nashville, TN, Tech. Rep. CS-81-19 (1981).

[7] R. Hauser & R. Manner, "Implementation of standard genetic algorithm on MIMD machines". Parallel Problem Solving fron Nature, PPSN 3, pp. 504-513, 1994.

[8] U. P. Schultz, M. Bordignon & K. Stoy, "Robust and Reversible Self-Reconfiguration", The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, USA, October 2009.

[9] Z. Butler, S. Byrnes & D. Rus. "Distributed motion planning for modular robots with unit-compressible modules". In Proc. Of the Int'l Conf. on Intelligent Robots and Systems, 2001.

[10] A. Abrams & R. Ghrist. "State complexes for metamorphic robot systems". Intl. J. of Robotics Research, 23(7) , pp. 809-824, 2004.

[11] G. Chirikjian & A. P. Chirikjian, "Bounds for self-reconfiguration of metamorphic robots", in Proceedings IEEE ICRA, vol.2, pp. 1452-1457, 1996.

[12] D. Christensen & al, "A Unified Simulator for Self-Reconfigurable Robots", In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, pp. 22-26, 2008.

[13] S. Murata et al., "Self-Assembling Machine", in Proceedings of the IEEE International Conference of Robotics & Automation., San Diego, California, pp. 441- 448, 1994.

[14] A. Pamecha & al., "Design and Implementation of Metamorphic Robots", in Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, Aug 1996.

[15] D. Rus & M. Vona, "Self-reconfiguration Planning with Compressible Unit Modules" in Proceedings of the 1999 IEEE International Conference on Robotics & Automation, Detroit, Michigan, vol.4, pp. 2513 - 2520, 1999.

[16] Y. Zhang & al., "Distributed Control for 3D Shape Metamorphosis", Autonomous Robots Journal, Vol. 10, Issue 1, pp. 41-56, 2001.

[17] A.C. Van Rossum & J. H. Van Den Herik, "Designing Robotic Metamorphosis". In Proc. of the 22nd Benelux Conference on Artificial Intelligence (BNAIC-2010), Luxembourg, 2010.