DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch129

Splittable Metamorphic Carrier Robots

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

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

***Abstract*—Metamorphic modular robots are versatile systems composed of a set of independent modules. These modules are able to deliberately change their overall topology in order to adapt to new circumstances, perform new tasks, or recover from damage. The modules considered in this paper are cubic shapes, and we assume that each of them has a separate computational resources and it is equipped with specialized sensors to perceive the environment. In this paper, we demonstrate the ability of these robots to evolve the topology of the whole structure in order to achieve, surround and transport target objects dispersed in the environment. While performing its task, the robot may be split up in order to cope with environmental variations. Our work integrates a simplified model of biological hormone system to generate inputs for a finite-state machine (FSM) that controls the evolution process.**

I. Introduction

Metamorphic self-reconfiguring modular robots are versatile systems that can change their overall shape with the intention of adapting to the task at hand. These robots are composed of a number of independent modules usually called atoms. These atoms are able to connect, disconnect one from each another, or even push/pull or exchange information and energy with the neighbor modules in order to form various structures/patterns dynamically [2,10,5]. Depending on the atoms degrees of freedom and the basic actions that can be performed in a coordinated way, several modules can perform elementary movements from position to an other position across their neighbors by changing the topology of the modules connectivity network [4,6,8,11], this action is called *robots reconfiguration*. For example, a self-configured modular robot can reform itself into a thin-linear pattern to cross a tunnel, reform into an emergency structure such as dam, shield, bridge, or even surrounding, carrying or manipulating objects.

Compared with conventional robotic systems, self-reconfigurable robots are believed to be more robust and more adaptive under dynamic environments, because on the one hand they are able to reconfigure modules to form more suited pattern with respect to the task at hand or the current situation. This property means that such robots can be survived and self repaired thanks to its ability to expel faulty modules outside the body [2,3], and in the other hand, the gathering of these modules forms a distributed system with no central controller since they are computationally independent, this property is crucial to make this kind of systems invulnerable to the failure situations or malfunction of robot modules.

Due to their many degrees of freedom, developing effective control system for modular robots is recognized as one of the major challenges in the development of self-reconfigurable modular robots. These challenges attract several researchers to investigate the feasibility of providing effective solutions using the existing approaches and mechanisms, or even propose others. Some of them focused on issues of modular-robots self-reconfiguration, while others focused on issues of modular-robots locomotion. A co-evolution of both configuration and control has been also the subject of many interesting research. For example, we can cite the Karl Sims model as the first major work evolving virtual robots [1], in this work Sims used a neural network to control a morphology generated by a graph-based genotype-phenotype map so that the controller was coevolved with the morphology generator. Later, Komosinski has used this strategy again with L-System morphology generator [15] to produce artificial robots. Evert Haasdijk has used HyperNEAT to develop a reactive quadruped modular robot [16], the controllers of the individual robots act autonomously and with only local exchange of information. However, the morphology of the organism is predefined by the user. More recently, artificial Gene Regulatory Networks seem to be able to generate complex morphologies when they control a developmental system [9,12,13,14] or even to generate oscillations that give artificial creatures a mechanism to move. However, few of these works have been designed to actually take advantage of the computational power of the individual modules.

The objective of our work is to evolve the structure of a metamorphic self-reconfiguring robot to perform the task at hand by taking advantage of the computational power of the individual modules. The research presented in this paper is grounded in our previous work [7] in which we demonstrated how simple local sensing, local communication and control rules achieve useful emergent behaviors of crystalline metamorphic robots. In particular in this work, we are interested in evolving the configuration of metamorphic modular robot to transport sliding objects from their current positions to specific target positions, this process can be subdivided in three successive steps: (1) evolve dynamically the structure configuration to find and surround the objects, we use morphogen gradients to locate these objects, (2) evolve the current structure configuration to be able to transport the surrounded objects, (3) release the sliding objects whenever the final position is achieved. These steps are coded in a simple finite-state machine (FSM) that denotes all the states in which every unit of the system may be, and the possible transitions can be performed according to the required conditions, a basic

hormone system is used to control the inputs of the FSM.

II. The Control Model

A. The Modular Robot and its Environment

To reduce the simulation's complexity, we consider a static environment modeled with a lattice (*grid*) of 2D cells, where each of these cells may be in one of the following states: empty, occupied with a single module, occupied with an obstacle. Basically, all the modules are the same size. Each of them is controlled by the evolutionary approach shown in figure 1, perceives the environment thanks to its specialized sensors and communicates with its nearby modules [7].

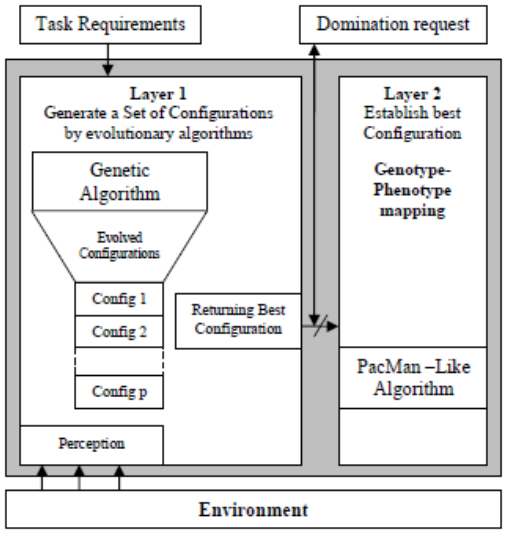

Fig. 1: Diagram of the evolutionary approach used to evolve the modular robot structure

The genetic algorithm used in this approach evolves a population of genomes that represent feasible configurations. Each of them is encoded as illustrated by figure 2.

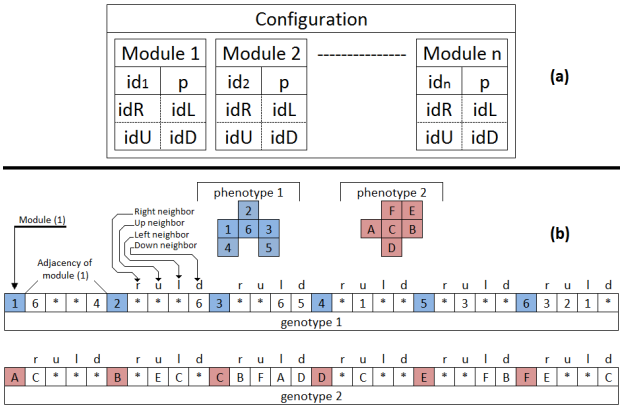

Fig. 2: Encoding a solution, (a): a genome that encod a configuration, (b): genotype-phenotype mapping

To encode an arbitrary configuration we used the adjacency matrix to denote which modules are adjacent to which other modules. Since we use Von Neumann neighborhood, a single module may have at least one adjacent module (zero adjacent is excluded) and at most four adjacent modules (respectively modules A,C in figure 2b). With this idea in mind, the adjacency matrix can be transformed into $5n$ size array (n is the number of modules) as illustrated in figure 3.

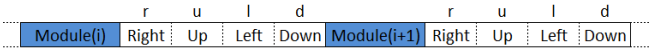

Fig. 3: simplified form of adjacency matrix

The *crossover* operation is applied to two different genomes that represent feasible configurations. Commonly, a point called crossover site along their length is selected, and the information after the crossover site of the two parent strings are swapped. As a result, two new children are created. However, this operation doesn't work directly with the genomes shown in figure 2. The next four additional actions are required:

1) From the first parent G_1, eliminate $(n-m)$ modules so that $m < n$ and $\forall M_i \in G_1$ / $Adjacency(M_i) \neq \phi$.
2) From the second parent G_2, eliminate $(n-k)$ modules so that $n = m + k$ and $\forall M_i \in G_2$ / $Adjacency(M_i) \neq \phi$.
3) From G_1 (respectively G_2), sellect two modules x_1, x_2 that have not a full adjacency list $\{\forall genome_i$ $\exists$ module $x_j \in genome_i$ / $\| Adjacency(x_j) \| < 4\}$ so that the translation of every module of the remaining part in G_2 by the vector $\overrightarrow{p_{x_1}p_{x_2}}$ (p_x: denotes the position of the module (x), figure 2.a) should respect the next constraint: $\forall x_j \in SubGenome_2$ $\nexists x_i \in SubGenome_1$ / $p_{x_i} = Translate_{\overrightarrow{p_{x_1}p_{x_2}}}(p_{x_j})$ Where, $SubGenome_i$ is the remaining part of $Genome_i$.
4) Translate all the modules (update the p vectors) of $SubGenome_2$ by the vector $\overrightarrow{p_{x_1}p_{x_2}}$, then update the adjacency list of (x_1, x_2) and integrate them together into new children genome as shown in figure 4.

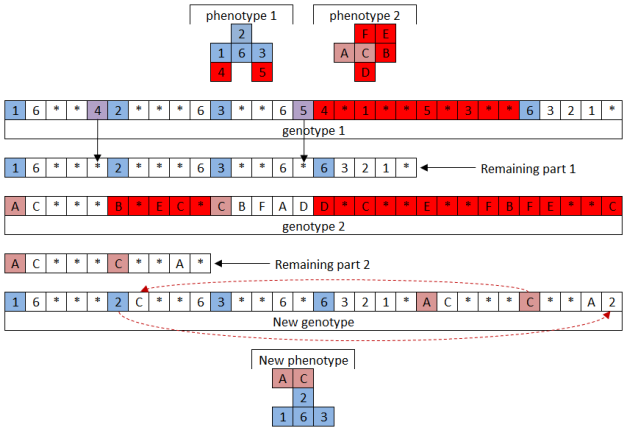

Fig. 4: The crossover operation

The *mutation* operation is applied to one genome from which a gene is chosen to be mutated. Except the part that encodes the module position, the remaining parts of the

selected gene should not be mutated. This operation proceeds as following:

1) Randomly select a genome G from the population.
2) Randomly select two genes (g_1, g_2) from the genome G so that g_2 has not a full adjacency list and the translation of the module specified by g_1 to fill a random free neighbor of the module specified by g_2 should not produce a fragmented phenotype.
3) Perform the translation and update the neighbor lists of (g_1, g_2) as shown in figure 5.

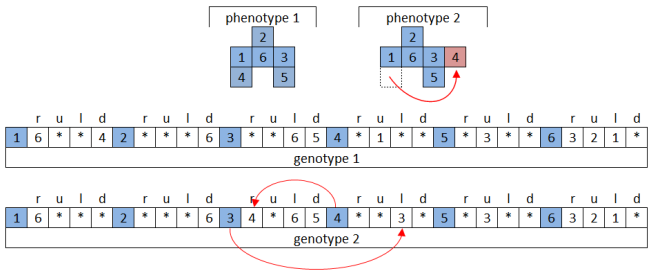

Fig. 5: The mutation operation

In this work we use the same software architecture discussed in [7], with a slight change in the sensing system (figure 6). Actually the cell-robot has no idea about the positions of the target objects, however it can sense some particular informations called *morphogens* diffused by these objects.

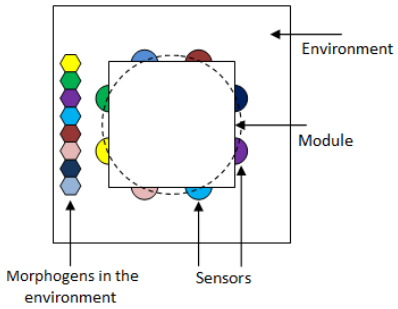

Fig. 6: Scheme of a cell-robot in an artificial environment. It has a ring of sensors (half-circles) to perceive the morphogens concentrations (hexagons)

We assume that the environment contains different morphogens. They are gradually spreading on the grid (figure 7) so that the variation on their concentrations between the neighbor cells emerges a guidance system that gives an implicit information about directions that should be followed to reach the source of these morphogens. This change improves the control model for taking advantages of not using global information and makes the system more realistic.

Our model integrates a basic diffusion algorithm to spread morphogens on each cell in the environment with respect to the following rules:

1) Each of the morphogens has a unique identifier.
2) Each of the target objects is considered as a morphogen source, it diffuses a unique morphogen on the environment.

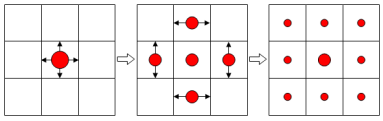

Fig. 7: Spreading a morphogen on the environment, the size of the circle represents the quantity of the morphogen, while the arrows represent the direction of spreading

3) The concentration of $morphogen_i$ is maximal at the position of its source (the biggest circle in figure 7).
4) The concentration of $morphogen_i$ changes across the grid, it decreases as we move away from its source.

Algorithm 1 Spread $Morphogen_i$

$List_A : list\ of\ empty\ cells.$
$List_B : list\ of\ marked\ cells.$
$List_A \leftarrow List_B \leftarrow \emptyset$
Set to (-1) all C_{Mor_i} for each of the empty cells.
Insert the target position cell into $List_A$ and set C_{Mor_i} to $MaxValue$.
while $List_A \neq \emptyset$ **do**
  if $List_A.First.C_{Mor} \geq \triangle_{Mor}$ **then**
    $Foreach(e \in Neighbor(List_A.First))\ do$
    if $e \notin List_B$ **then**
      $e.C_{Mor} \leftarrow max(e.C_{Mor}, List_A.First.C_{Mor} - \triangle_{Mor})$.
    end if
    if $e \notin List_A$ **then**
      $Insert\ e\ into\ List_A$.
    end if
    $EndForeach$
  end if
  $InsertList_A.First\ into\ List_B$.
  $Delete(List_A.First)$.
end while

In this algorithm:
C_{Mor_i}: denotes the concentration of $morphogen_i$
C_{Mor}: denotes the overall morphogens concentrations.
$\triangle_{Mor}$: denotes the change in morphogen concentration through neighbor cells.
$Neighbor(X)$: returns the list of Von Neumann neighbor cells of $Cell_X$.

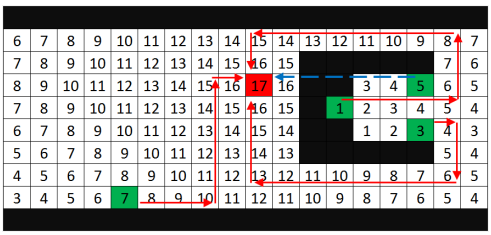

Fig. 8: Result of applying Spread $Morphogen_i$ algorithm, black squares represent obstacles, red square represents a target object, green squares represent mobile robots, values from 1 to 17 represent the concentrations of $Morphogen_i$ across the empty cells.

Figure 8 shows the result of applying $Spread\ Morphogen$ algorithm to spread a single morphogen across all the empty cells in the environment. The green squares represent mobile robots that have a desire to reach the position of the morphogen source while the red square represents a morphogen source, so in this position the concentration of the corresponding morphogen is set to its maximum value (17 in this example). The algorithm ensures a gradient diffusion of the morphogen across each empty cell. So to find the source position, the robot will just have to follow any path that increases the morphogen concentration (red paths in figure 8).
In fact, we developed this algorithm in hope to improve our previous model [7] in which we used euclidean distance to locate target objects and to drive the system evolution (blue dashed path in figure 8). In such a model, all the units are supposed to know the exact position of all the target objects as a global information. Besides that, using euclidean distance makes the metamorphic robot unable to get out from some situations, in particular avoiding obstacles that are parabolic in shape (figure 8). To resolve this problem, we introduce f_{moving} fitness (equation 1) that should be performed by the robot to acquire morphogens as much as possible. Maximizing f_{moving}, the robot will track the morphogens concentrations from low to high level of concentration.

$$f_{moving} = \sum_{i=1}^{m} \sum_{j=1}^{n} Morphogen_i(m_j) \tag{1}$$

In this equation, $Morphogen_i(m_j)$ denotes the concentration of $Morphogen_i$ perceived by the module m_j and n denote the number of the modules, m denotes the number of the morphogens sources.

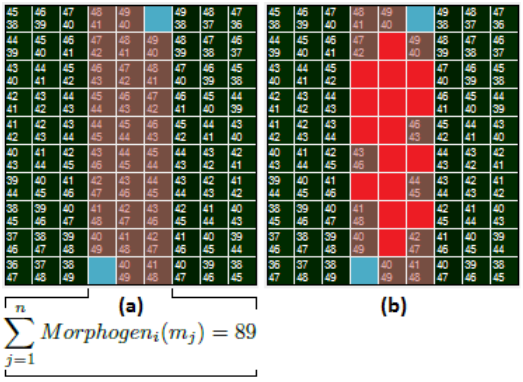

Fig. 9: The most concentrated cells ≡ equipotential area, the dark red squares represent the modular robot, the blue squares represent the target objects, values from 36 to 49 represent morphogens concentrations

Using f_{moving} as a fitness, our GA may have a tendency to converge towards local optima in which it is not defined how to sacrifice short-term fitness to gain longer-term fitness. As a result, the robot gets stuck in an equipotential area from which it can not get out anymore. This particular circumstance strongly depends on the shape of f_{moving} landscape that depends itself on the way of spreading morphogens.
In order to alleviate this problem, we introduced an activator/inhibitor coefficient (δ_i) to control the fitness f_{moving} as illustrated in equation 2.

$$F_{Moving} = \sum_{i=1}^{m} \delta_i \sum_{j=1}^{n} Morphogen_i(m_j) \tag{2}$$

In this equation δ_i is used to activate or inhibit $Morphogen_i$. Using F_{Moving}, the robot can performe the task at hand either sequentially by activating a single morphogen at a time, or in parallel by dividing the whole structure into m parts, where $m > 1$ is the number of morphogens sources (target objects). To divide the whole structure, we used the following three rules:

1) Cluster the modules into m classes, using their perceived morphogens as an input data (refer to section 3 for more details), and assign each of the modules the appropriate class-identifier.
2) Each of the modules keeps the link with the same-class modules and disconnect from the others.
3) For each class C_i, δ_i gets value as shown in the following equation 3, where id_{C_i} is the identifier of C_i.

$$\delta_i = \begin{cases} 1 & if\ i = id_{C_i} \\ 0 & if\ i \neq id_{C_i} \end{cases} \tag{3}$$

Once the structure divides, each part behaves as an entirely autonomous modular robot in which only one morphogen is activated where the others are inhibited. In such a case, no equipotential area appears and each part can successfully track and surround a unique morphogen source.

B. Generate Cyclic Locomotions

The GA used in this work is basically designed for evolving the structure of modular robots as discussed in [7], it gives only the next configuration that improves the fitness at hand. However, it can be used to develop facilities for generating locomotions.
Actually, the modules can generate various motions as a combination of each module micro-movement. In particular, they are able to generate an earthworm-like locomotion as a loop of simple cyclic locomotion.

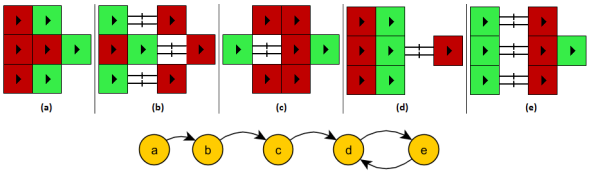

Fig. 10: Cyclic locomotion, green cells: moving modules, red cells: stopped modules

Figure 10, shows a loop of simple cyclic locomotion that can be generated using the folowing rules:
(1): Define a direction for the movement.
(2): Arrange the modules so that each of them can disconnect from its neighbor modules that are perpendicular to its direction without leaving any isolated module.
(3): The module is able to move one step position (green modules in figure 10) once the folowing conditions are satisfied:
(3.1): The neighbor cell to the direction of movement is empty.

(3.2): The disconnection from neighbor modules that are perpendicular to the direction of its movement should not leave any isolated module.
(3.3): All the module faces are contracted.
In this work, the direction of movement is defined by using the variation of morphogen concentration around the space occupied by the modules, while the modules arrangement is defined by using GA since it is a particular configuration.

C. Encapsulation into the modules

As mentioned in the introduction, we are interesting to evolve the configuration of metamorphic modular robot to transport sliding objects from their current positions to specific target positions. This process can be subdivided in three successive steps:
$Step_1$ Track and surround the objects: the modules run a GA for evolving the whole structure in order to acquire morphogens as much as possible (using F_{Moving} as a fitness where $\delta_i = 1 \ \forall i = 1..m$). This evolution drives the modular robot towards an equipotential area in which it gets stuck and can not completely converge towards any of the target objects. (at this moment $\triangle_{F_{Moving}} = 0$).
Each module on the system can either produce two artificial hormones H^1 and H^2 (equations 4,6) or diffuse an amount of them (H^1_r, H^2_r) to control their desires to switch between the three steps.
Once the modular robot reaches an equipotential area, each of the modules starts to lose H^1 since $\triangle_{F_{Moving}} = 0$ according to equation 4. An intermodular compensation is fired (equation 7) to ensure the homogeneity of H^1 through all the modules and while $\triangle_{F_{Moving}} = 0$, H^1 keeps decreasing until it reaches a lower threshold. At this moment, a clustering method is used for determining a division scheme using the informations perceived by the modules as an input data. The whole structure is then divided into several parts where each part will be attracted by only one target object that matches with its class identifier.
$Step_2$ Transport the objects: As a result of step (1), each target object is surrounded by several modules of the same class. Again, F_{Moving} achieves a maximum level and $\triangle_{F_{Moving}}$ converges towards 0, at this moment, every module of the class starts to lose H^2. The modules that are in interaction with the target object lose H^2 faster than the others (φ in equation 6). Once H^2 gets lower, the modules define the direction of movement using the variation in concentration of the morphogen that denotes the final position, then a GA is called to evolve a structure that can generate a cyclic locomotion towards the defined direction.
Each substructure can push ahead the sliding object or pull it from the back while the sliding object moves from low concentration level to high concentration level of morphogen that denotes the final position. Otherwise, the modules in interaction with the sliding object stop the movement and diffuse a hormone H^3 to switch to $Step_2$ and redefine a new direction.
$Step_3$ Release the objects: Once the sliding object arrives at the final position, it is expelled as if an obstacle or a failed module.
The dynamic of these steps can be modeled by a finite state machine as shown in figure 11, where:
$Start$: denotes the initial state of the system.

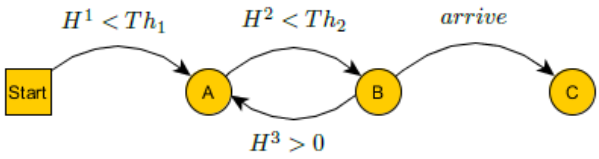

Fig. 11: FSM modeling the global task

States (A, B, C): denote respectivelly steps 1,2,3.
(Th_1, Th_2): denote respectively thresholds of hormones (H^1, H^2).
In this work we integrate a highly simplified model of biological hormone system to generate inputs for FSM.
The dynamics of the hormones H^1 and H^2 are modeled as in equations 4–6:

$$H^1 = H^1_r + \alpha_1 \triangle_{fit} - \beta_1 f(\triangle_{fit}) \tag{4}$$

$$f(x) = \begin{cases} 0 & if x \neq 0 \\ 1 & if x = 0 \end{cases} \tag{5}$$

$$H^2 = H^2_r + \alpha_2 \triangle_{fit} - \varphi\beta_2 f(\triangle_{fit}) \tag{6}$$

Where, $\triangle_{fit} = |F(t) - F(t-1)|$: denotes the change of the fitness over the time. (α_1, α_2): are two coefficients used to accelerate the production of H^1 and H^2. (H^1_r, H^2_r): represent the received hormones. (β_1, β_2): are two coefficients used to decelerate the production of H^1 and H^2. φ: is a coefficient used to enhance the deceleration of producing H^2.
The function $D^i_{x,y}(t)$ models the diffusion of the hormone H^i at time t as described in the following equation 7, where d_i represents the diffusion coefficient of H^i:

$$D^i_{x,y}(t) = d_i |x - y| H^i(t) \tag{7}$$

III. Clustering The Modules

Clustering is the task of finding natural groupings among objects in such a way that objects in the same group (called a cluster) share similar values [18]. In our work we use clustering for partitioning modules into several groups so that the whole structure of the modular robot will be split up in order to cope with environmental variations. From this point of view, the perceived information of each module is considered as a data point.
In this section we discuss three clustering methods in the aim of choosing the more efficient of them for our study. These methods are applied to the same data inputs.

A. K-means Clustering

K-means is a well known algorithm commonly used to solve clustering problems, it is based on minimizing the overall sum of the squared errors between each pattern and the corresponding cluster center. This can be written as minimization of the following objective function:

$$E = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - m_i\|^2 \tag{8}$$

K-means clustering proceeds as shown in algorithm-2, where m_i represents the center of the cluster C_i. This algorithm converges when there is no further change in assignment of

Algorithm 2 K-means

while $\exists \triangle m_i \neq 0$ **do**
(1): Initialize the k cluster centers.
(2): Assign each input data point to one of the existing clusters according to the closest Euclidean distance from the clusters.
(3): Compute the mean of each cluster to update its center.
end while

instances to clusters. In such situation, all the centers m_i are stable in values ($\triangle m_i = 0$).
Depending on the first initialization of the clusters, k-means algorithm converges in 3–8 iterations, and gives a good data-classification quality (a low overall error).

B. GA-Based Clustering

The goal of using GA to solve clustering problem is to enhance the quality of clusters. In fact the GA works with a population of feasible solutions called genomes, each genome encodes a solution and is assigned a fitness value to describe how good solution it represents.
Starting from a random initial population, the GA stochastically selects a set of individuals (based on their fitness) and modifies their genetic codes by means of operators: mutation and crossover to form a next evolved population.
The genome that encodes the clusters is shown in figure 12, where id represents the data point identifier and c represents its corresponding cluster.

id	1	2	3	4	5	6	7	8	----------	399	400
c	3	5	2	2	3	4	1	1	----------	4	5

Fig. 12: Genome that encodes the clusters

A feasible solution assigns each element of the data point to a cluster. The center of each cluster is determined and the overall error E of the solution is calculated using equation 8. The fitness should be inversely proportional to the overall error of the corresponding genome, in this paper we adopt a basic form shown in equation 9.

$$Fitness = \frac{1}{\sqrt{E}} \tag{9}$$

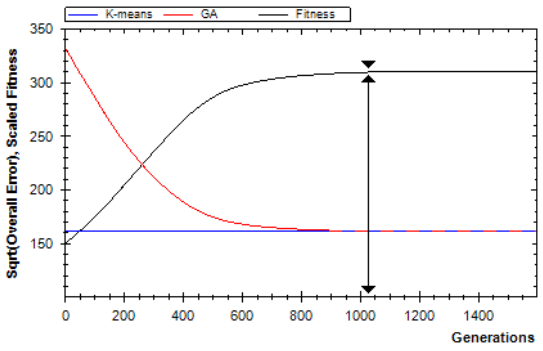

Fig. 13: GA-Clustering

As shown in figure 13, the GA improves the quality of the clusters in each generation. However it seems to be of little interest for our work since it needs an average of 1025 generations to converge around k-means solution.

C. SOM-Based Clustering

Kohonen Self Organizing Map, or SOM, is an artificial neural network based on an issue of unsupervised learning with the aim of mapping a high-dimensional data to a low-dimensional space (figure 14) which is formed by arranging the computational neurons into a grid [17,19].

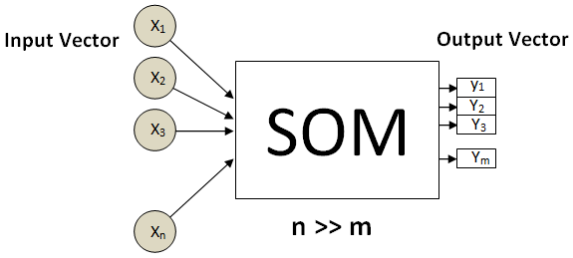

Fig. 14: Global sheme for SOM clustering

The way that SOM goes about organizing itself describes a self organization process, during this process a competition between neurons is invoked, and each neuron is allowed to change its weight vector to become more like samples in hopes to win the next competition. The general steps of the learning process are described in algorithm 3:

Algorithm 3 SOM

(1)-Initialization:
Randomly initialize the weight vectors of the map
(2)-Begin training:
while $t \leq 1$ **do**
An input weight vector is applied to the map
The node most like the input is selected (Best Matching Unit).
The weight vectors of BMU and surrounding neighborhood nodes are scaled towards the input.
The Learning rate and the ray of neighborhood are decreased.
end while
(3)-Apply input vector:
The BMU now is the node most like that provided.

In first stage, the best matching unit (BMU) should be selected, usually as the unit matching to the shortest euclidean distance (equation 10) between the input vector and the units of the map.

$$D_{(v_2,v_1)} = \sqrt{\sum_{i=1}^{n}(v_{2i} - v_{1i})^2} \tag{10}$$

Next, the neighboring weights should be scaled, so firstly we should select the units considered as neighbors to the winner unit, then we should determine how much each weight can become more like the input vector. The neighbors of a winning unit can be determined using a number of different methods.

In this paper we opted to use a gaussian function (equation 11) where every point with a value above zero is considered as a neighbor.

$$G(x,y) = \alpha e^{-\frac{x^2+y^2}{\beta}} \tag{11}$$

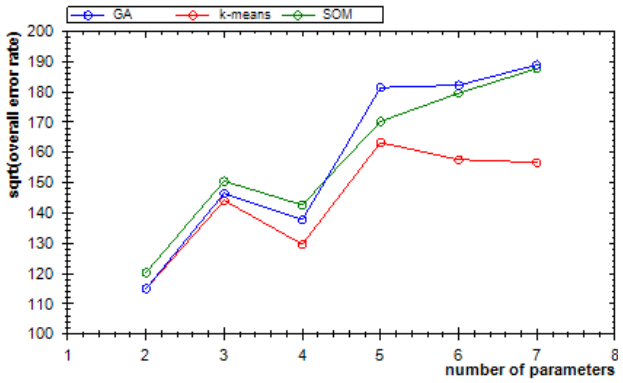

Fig. 15: Overall error of k-means, SOM, GA clustering methods

To compare these clustering methods, a data set is captured from a random robot configuration. The captured informations represent the morphogen diffusion across the cells occupied by the robot modules.
As shown in figure 15, we note that k-means is not just the easiest algorithm to be realized, but also our natural choice since it gives a high quality of data clustering (for the purpose of our work) and has low computation cost (converges in 3 to 8 iterations).

IV. Results

In this experiment, the environment is modeled as a 2D grid composed of 25x10 cells as shown in figure 17, where, the shaded cells represent obstacles, the blue cells represent sliding objects, and the 16 unit modular robot is represented by the red cells. Each of the sliding objects produces a unique morphogen that is spreaded gradually on the environment. The mission should be performed by the modular robot is to track the sliding objects that are randomly dispersed in the environment and transport them into a predefined final position (we assume that the final position diffuse a unique morphogen called $Morph_{Final}$).

The following parameters are used to set up the simulation: Max morphogen concentration = 50, hormone accelerators (α_1, α_2) = (0.8,0.8), hormone decelerator (β_1, β_2) = (0.3,0.3), hormone diffusion coefficient d_i=0.5, the cumulation of any hormon H^i can not exceed 10 (otherwise the additional value is ignored except for the non divided structure). This setup has been empirically determined, through a set of tests.

During the convergence of the genetic algorithm, it is interesting to observe the evolution of the structure towards the best solution. As it is shown in figure 17(a.b), the modular robot starts form its initial position and evolves its structure to aquire morphogens as much as possible. As a result, the modular robot converges more and more towards the most concentrated cells (red-brick cells in figure 17(a.b.c)), at this

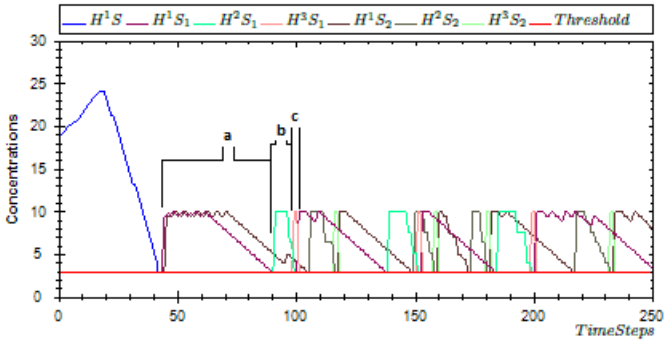

Fig. 16: Hormones concentrations during the time of simulation, (a,b,c): parts of the global task

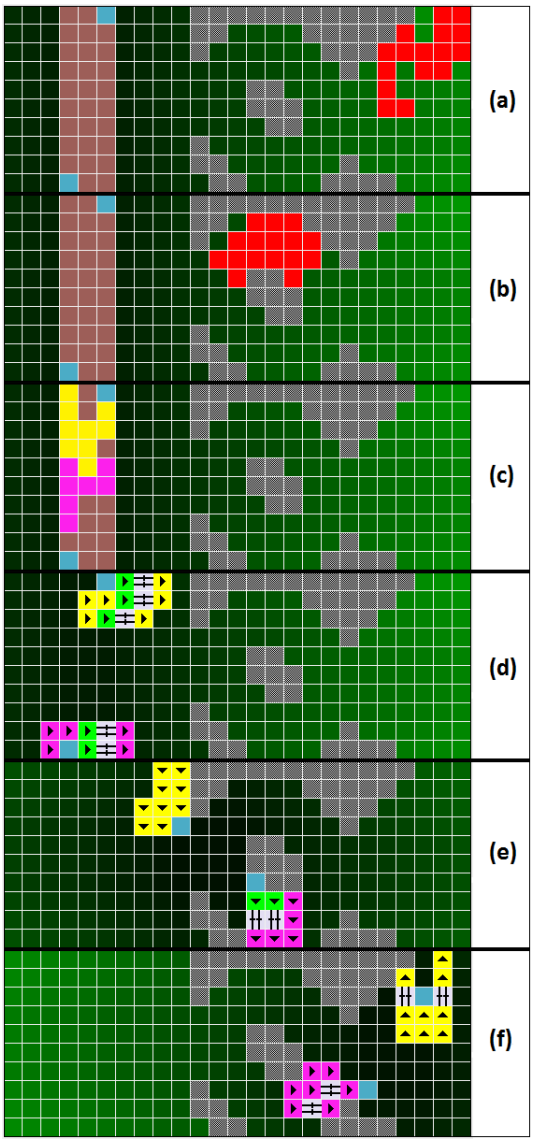

Fig. 17: The modular robot during the evolution

moment, the structure is not yet divided, and only H^1 is produced. Once the structure reaches the equipotential area (figure 17c), $\triangle_{F_{Moving}}$ approaches to 0 and H^1 starts to decrease (blue curve in figure 16).
When the H^1 concentration gets down under the threshold 2.5 (empirically determined), the structure is believed to be steady for a long time and is ready to be divided into several parts. A k-means algorithm is called to cluster the dataset (informations perceived by the modules) and to assign each module to a class preparing to the separation process. Next, each module tests its neighbors and disconnects from those that have not the same class identifier. As a result, the modules are separated in two classes and the structure is divided in two parts (figure 17c), where these parts should not be divided anymore and each of them converges only to the object attracted with, then it surrounds this object and evolves its configuration to be able to perform a cyclic locomotion to the direction by which the concentation of $Morph_{Final}$ is being increased (subsript b in figure 16), otherwise, a hormone H^3 is diffused to evolve an other configuration to move toward a new direction. As a result, the sliding object gets closer to the final position.
The global system dynamic is already modeled by a FSM that is encapsulated in every module to switch between steps (parts of the global task) a,b,c in figure 16 while the hormon system produces H^1, H^2, H^3 to generate inputs to this FSM. Coupling between the hormone system and the FSM is illustrated as following:
$(State = Start \wedge H^1 < 0.3) \mapsto (State \leftarrow A, H^2 \leftarrow Max)$
$(State = A \wedge H^2 < 0.3) \mapsto (State \leftarrow B)$
$(State = B \wedge H^3 > 0) \mapsto (State \leftarrow A, H^2 \leftarrow Max)$
Observing these rules, we notice that the second and the third rules, create such an interesting cycle that can be used to create a generalized process for more complex tasks.

V. Conclusion

In this paper, we presented a decentralized approach that evolves the ability of metamorphic robots to perform the task at hand. This approach is based on our previous work [7] in which we used a GA coupled with a PackMan-like algorithm for evolving the structure of a modular robot. In this study the genetic algorithm is used to generate the next better configuration of the modular robot, while the PackMan-like algorithm is used to drive the self-reconfiguration process. Switching between generating better configuration and reforming to the new configuration emerges an adaptive locomotion for the modular robot.
A more complex task is considered in this work, and a new evolutionary approach is presented.

The first improvement we can talk about is the ability of our approach to drive the modular robot into the target objects without being stuck by obstacles that are parabolic in shape (figure 8). In fact, using a gradient of morphogens instead of euclidean distance is not just a natural choice since the modules are not supposed to have global informations but it gets also a significant improvement to the quality of objects-tracking in our system.

The experiment presented in this paper shows the capacity of the artificial hormone system to control the finite state machine (FSM) that schedules the steps to perform the global task. To do that, the global system task dynamic is modeled by a FSM, and then an artificial hormone system is used to control transitions between the FSM states. As a result, the robot's behavior is controlled by the hormone system.
An other interesting property of our approch is the ability to generate a separation strategy for the modules according to some circumstances. It should be interesting to investigate the feasibility of biological cell division techniques as well as the multi-objective techniques for generating such strategy.

References

[1] K. Sims. Evolving 3d morphology and behavior by competition. Artificial Life IV, pages 28-39, 1994.

[2] 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.

[3] D. Christensen. Experiments on Fault-Tolerant Self-Reconfiguration and Emergent Self-Repair. In IEEE Symposium on Artificial Life (ALIFE'07), pages 355-361, 2007.

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

[5] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. IEEE Robotics and Automation Magazine, 14(1):43-52, 2007.

[6] S. Vassilvitskii, J. Kubica, E. Rieffel, J. Suh, M. Yim. On the General Reconfiguration Problem for Expanding Cube Style Modular Robots. In IEEE Intl. Conf. on Robotics and Automation (ICRA), 2002.

[7] T. Ababsa, N. Djedi, Y. Duthen, S. Cussat-Blanc. Dencentralized Approach to Evolve the Structure of Metamorphic Robots (regular paper). In : IEEE Symposium on Artificial Life (IEEE-ALife 2013), april 2013.

[8] M. Vona, D. Rus. A Physical Implementation of the Self-reconfigurable Crystalline Robot. In Proc. of the IEEE Intl Conf. on Robotics and Automation 2000, San Francisco, CA, Apr. 24-28, 2000, p. 1726-1733.

[9] J. Bongard and R. Pfeifer. Evolving complete agents using artificial ontogeny. Morpho-functional machines: The new species (designing embodied intelligence), pages 237-258, 2003.

[10] 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.

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

[12] L. Schramm, Y. Jin, and B. Sendho. Emerged coupling of motor control and morphological development in evolution of multi-cellular animats. Advances in Artificial Life: Darwin Meets von Neumann, pages 27-34, 2011.

[13] Sylvain Cussat-Blanc, Jordan Pollack. A Cell-based Developmental Model to Generate Robot Morphologies (regular paper). In : Genetic and Evolutionary Computation COnference (GECCO 2013).

[14] Meng, Y, Zheng, Y and Jin, Y. A Morphogenetic Approach to Self-Reconfigurable Modular Robots using a Hybrid Hierarchical Gene Regulatory Network. On Artificial Life XII, 2010.

[15] Komosinski M, Rotaru-Varga A. From directed to openended evolution in a complex simulation model. In: Bedau MA, McCaskill JS, Packard NH, Rasmussen S, editors. Artificial Life 7. Cambridge, MA: The MIT Press, 2000. p. 2939.

[16] Haasdijk, E., Rusu, A.A., Eiben, A.: Hyperneat for locomotion control in modular robots. In: 9th International Conference on Evolvable Systems (ICES 2010). pp. 169180 (2010)

[17] Balakrishnan, P. V., M. C. Cooper, V.S. Jacob, P.A. Lewis. A Study of the Classification Capabilities of Neural Networks using Unsupervised Learning, Psychometrika 59(4): 509-525 (1994).

[18] Jain, A.K., M.N. Murty, P. Flynn. Data Clustering: A review. ACM Computing Surveys 31(3): 264-323. (1999)

[19] K.Mehrotra, C. Mohan, and S. Ranka. Elements of Artificial Neural Networks. MIT Press, (1996)