Automated Synthesis of Locomotion Controllers for Self-Reconfigurable Modular Robots

Fernando Torres and Juan Cristóbal Zagal

Department of Mechanical Engineering, University of Chile.
Beaufchef 850, 8370448, Santiago, Chile.

{fertorre, jczagal} @ ing.uchile.cl

Abstract. Previous studies on control of self-reconfiguring modular robots have shown how complex group behavior can be obtained from simple low level interactions. In this study we explore the power of Genetic Algorithms and NEAT to automatically produce group behavior such as locomotion with obstacles. We study the invariance of resulting rule set controllers with respect to different scenarios, scales, and initial robot configurations. Resulting GA controllers performed 17.88% better than NEAT controllers. The use of sequential mode of cell activation was critical for the evolvability of robot controllers.

Keywords: NEAT, Genetic Algorithm, Self-Reconfiguring Modular Robots, Programmable Matter.

1 Introduction

Self-reconfigurable modular robotic systems have the potential of being more versatile and robust than current robots. A group of modules may reconfigure itself into a tool shape, suitable to accomplish an unforeseen task. Eventually, members of the same group may replace damaged or lost components, giving the overall system the capability of self-repair [9].

Achieving unsupervised adaptive self-organization is fundamental on a wide variety of applications, and this type of systems promises to bring new light into the field of robotics and automation. However, there are a number of problems that must be addressed in the way of producing practical low cost modular self-reconfiguring machines.

Planning and control have been recognized amid the great challenges in modular self-reconfigurable robotics [9]. There is a need of algorithms for locomotion of modular systems under environments with obstacles. The later is of utmost relevance, since locomotion is a sub-task of reconfiguration and self-repair.

Various studies on generic locomotion algorithms for self-reconfigurable robots have been carried by Rus and colleagues [1,2]. They focused on methods inspired by cellular automata. The idea is that a set of geometric rules is used to control module actions by looking at their neighborhood. Modules operate in a lattice environment composed either by free space, obstacles or other modules. They are assumed to be able to sense the type of elements on their surroundings, and thus, they can decide where to move by applying a shared geometric rule set.

Although the rule set only defines low level interactions, it has been shown how complex global behavior arises as a consequence of these local interactions [5]. Unfortunately, there is no clear model of how local rules can be designed or tuned in order to achieve a desired high level behavior. So far, researchers have concentrated on testing manually designed rule sets with a potential for solving locomotion or self-repair tasks.

In [1] rule sets for locomotion with and without obstacles were analyzed. Tests were performed on a 2D lattice simulation environment. A set of five rules was sufficient to achieve locomotion without obstacles. Locomotion over certain type of obstacles was achieved with a set of eight rules. Three different rule evaluation models were tested.

At first, cells were evaluated in a cyclical order (D_0 model). Then, cells were evaluated at random, without allowing for repetition along an evaluation cycle (D_1 model). Finally, cells were evaluated completely at random, allowing for repetition of activation on a single cycle (D_∞ model). They observed how the later, most flexible model, requires more complex rule sets, although it allows for more variability and potential use on distributed systems.

Despite of their success at generating basic locomotion controllers, it was clear that the manual generation of such rules is an increasingly difficult task, especially as the complexity of environment increases and more challenging sophisticated behaviors are required. This encourages the development of automated methods to generate rule sets for self-reconfigurable modular robots.

Stoy [8] also uses cellular automation to guide the reconfiguration of a modular ensemble so as to reach a desired geometrical configuration in three dimensions. The local behavior is automatically generated over the basis of a gradient function constructed with a CAD model of the target geometrical shape. He provides successful examples on the autonomous construction of a configuration resembling a chair.

Promising results on evolving cellular automata rules were obtained by Koza and colleagues in another context [3]. Their rules were evolved for solving the Majority Problem, and achieved a performance better than the best known human-written solution (Gacs-Kurdyumov-Levin rule).

Our present work focuses on the automatic generation of rule sets for the locomotion of self-reconfigurable modular robots. We will explore the use of Genetic Algorithms (GA) [4] and Neuroevolution of Augmenting Topologies (NEAT) [7].

The remainder of this paper is as follows: In section 2 we present our experimental setup. Results of the adaptation process are shown on section 3. In sections 4 and 5 we explore the robustness of our solutions by varying the initial robot shape and scale of simulation. Finally, in section 6 we present the conclusions of our work.

2 Experimental Setup

2.1 Simulation

We used a matrix representation of a 2D lattice environment containing robot cells, background and obstacles. Figure 1 shows various elements of our simulation. An example of a robot and environment is shown in (a). A mask of identifiers centered on an activated cell is shown in (b). A rule input condition is shown in (c). Possible displacements are presented in (d). An example of an entire simulation is presented in (e). Obstacles are represented by "2", background by "1" and robot cells by numbers greater or equal to "3". A neural network used as an alternative mapping between geometric conditions and cell displacement is shown in (f).

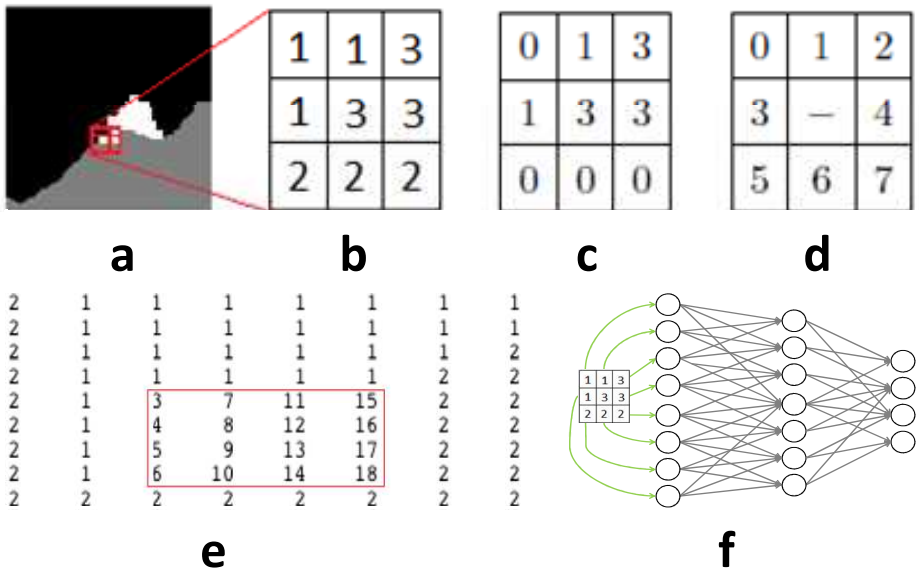

Fig. 1. **a.** Example of a robot (white) moving on an environment containing obstacles (grey) and free space (black). The red square represents a mask centered on an activated cell. **b.** The mask is used to evaluate if a rule input condition is satisfied. **c.** Example of a rule input condition where robot cells ("3") and background elements ("1") are relevant. In this example the input condition is met and a displacement is triggered. **d.** Eight possible displacements are represented by numbers (0-7). **e.** Matrix representation of a small simulation environment. Obstacles are represented by "2", background by "1" and robot cells by numbers greater or equal to "3". **f.** Neural network used as an alternative mapping between geometric condition and displacement.

2.2 Training and Test Environments

We produced complex training and test scenarios by segmenting natural images. Figure 2 shows our training (a) and test scenario (b). The later scenario is particularly challenging, since it usually forces a moving robot to split in two pieces. Simulation environments of different sizes were produced out of these images.

The idea was to test the robustness of resulting behaviors to changes in the scale of simulation. The complete learning process was carried on the training scenario, while the final experiments were done in both training and testing scenarios.

Fig. 2. Training and test scenarios used for our experiments. The darker area represents the background, while the gray area represents the obstacles. **a.** Training scenario. **b.** Test scenario.

2.3 Encoding

In the case of GA, we represented a locomotion controller by an individual binary string genome. The genome is implemented as a string containing pairs of input rule geometric conditions and action displacements. Each rule was represented by a total of 21 bits, where 18 bits are used encoding the input rule and 3 bits are used for representing the displacement. We considered a total of 30 rules for our representation.

In the case of NEAT, we used the numerical representation provided by the simulation itself. The eight outer values of the cell centered mask were fed into a neural network, as shown in Figure 1f. The four output neurons were considered as binary values and paired to represent the horizontal and vertical displacement separately.

2.4 Objective function

The objective function was designed to maximize overall robot locomotion toward the right most portions of the scenarios while maintaining, at the same time, the connectivity of the robot. We defined the travelled distance d as the averaged position of each robot cell at the end of the simulation. We also defined the connectivity c as the (normalized) size of the largest 4-connected cell group averaged through every simu-

lation step. Both quantities were bounded in the interval [0,1]. Then the fitness f for each candidate individual was defined as

$$f = d \cdot c \tag{1}$$

2.5 Testing parameters

The training was executed with robots of 4 by 4 cells size. The training scenario was chosen to be 30 (height) by 50 (width) cells. Five learning trials were performed for GA and NEAT. The best performing individual was selected for further characterization on each method.

Further tests consisted on studying the invariance of resulting controllers with respect to changes on robot's initial shape and scale of the simulation. These tests were executed on both, training and test scenarios. Table 1 defines the configurations for initial shape invariance study and Table 2 defines the settings for simulation scale invariance analysis.

Table 1. Initial shape invariance settings. The scenario size was fixed in 30x50 cells.

Shape Index	Robot Height	Robot Width
I	4	4
II	3	5
III	5	3
IV	2	8
V	8	2

Table 2. Scale invariance settings. Each case corresponds to 1x, 2x, 5x and 10x the size of the initial learning configuration.

Scale Index	Scenario Height	Scenario Width	Robot Height	Robot Width
I	30	50	4	4
II	60	100	8	8
III	150	250	20	20
IV	300	500	40	40

3 Results

Figure 3 shows the learning curves for Sequential GA, Random GA and NEAT in terms of number of evaluations. We can observe that Random GA shows signs of premature converegence. NEAT shows the smaller standar deviation. Despite of this, its final fitness is lower than that reached by GA.

The final average fitness is 0.862 for sequential GA, 0.860 for random GA and 0.709 for NEAT. The overall standar deviation is 0.179 for sequential GA, 0.118 for random GA and 0.019 for NEAT. Averages were taken by considering five independent learning trials.

The maximum fitness obtained with GA was 0.936. On the other hand, the best NEAT individual‘s fitness reached 0.73. These were the individuals selected for further characterization.

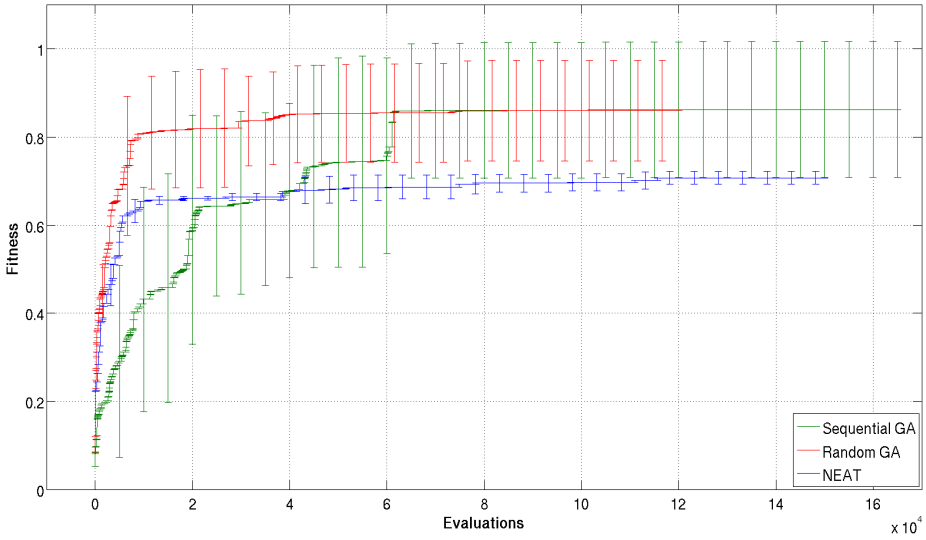

Fig. 3. Resulting learning curves obtained for sequential GA, random GA and NEAT. The final average fitness is 0.862 for sequential GA, 0.860 for random GA and 0.709 for NEAT. The overall standar deviation is 0.179 for sequential GA, 0.118 for random GA and 0.019 for NEAT. Averages were taken by considering five independent learning trials.

4 Initial Shape Invariance

Figure 4 shows the results obtained when varying the initial robot shape. For the case of GA, the fitness obtained was close to the one obtained during training, for all initial configurations and scenarios. On the other hand, the results shown for NEAT didn‘t show a clear relationship between initial shape and fitness.

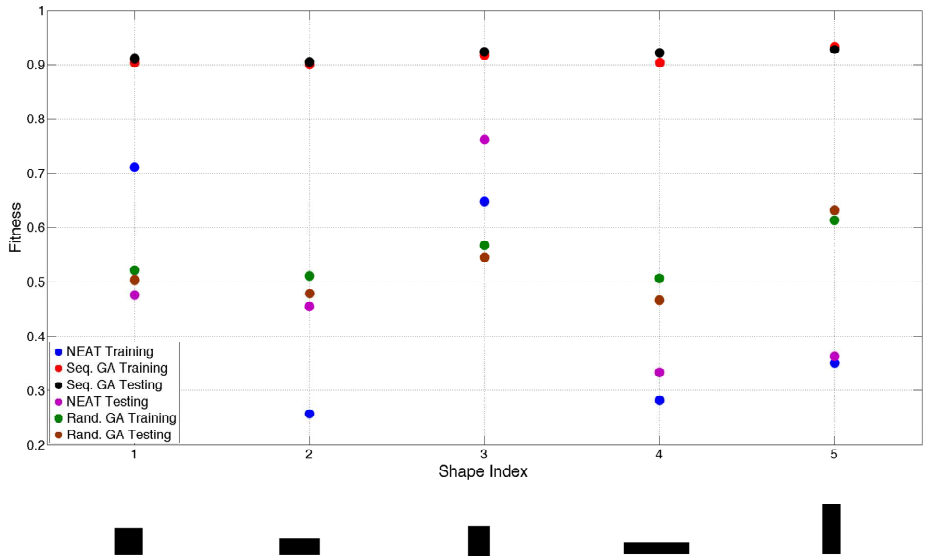

Fig. 4. Summary of fitness values obtained for different methods, scenarios and initial robot shape. Black and red dots are representing the level of fitness obtained when using Sequential GA. There is a clear similarity of results obtained under test and training scenarios. Results show how Random GA always gives lower results under testing, even during training configuration.

5 Scale Invariance

Figure 5 shows the fitness obtained with different methods and scenarios while varying the simulation scale. The resulting NEAT behavior showed a higher invariace between scenarios than those obtained with GA, although they don't show a clear relation between configurations. In the GA case, it can be seen an inverse relationship between the fitness obtained and the simulation scale. This might be due to the number of simulation steps required to cross the scenario. The later implies a disconnected locomotion to reconnect at the end of the scenario (See Figure 6).

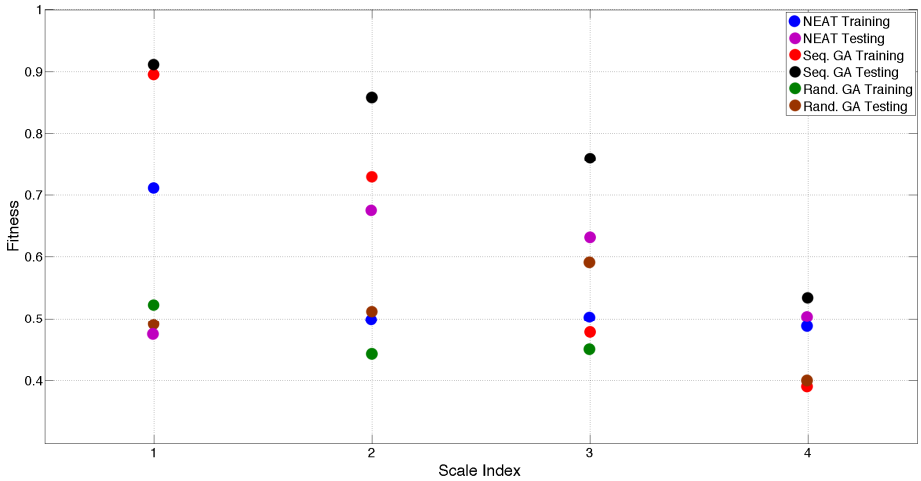

Fig. 5. Fitness values obtained with each evolutionary method in both scenarios and while varying the simulation scale index.

6 Conclusions

We have studied three methodologies to generate decentralized controllers to achieve locomotion of self-reconfigurable modular robots. We have tested the resulting controllers by varying the initial robot's shape and the simulation scale, using different scenarios.

We have observed the convenience of using secuential module activation, so each individual evolved could be characterized by one fitness value. In other hand, random activation resulted in noisy learning curves, where the resulting fitness were not even close to those resulting during the learning process.

Sequential GA results showed invariance to the initial shape and an inverse relation to the scale of simulation. This behavior seems to be independent of the simulation scenario. It was also observed that GA generated controllers with slight better performance in vertical initial robot shapes. On the other hand, NEAT showed no clear relationship through the configurations being tested.

The cellular automata representation used with GA is able to condense several situations into a single rule, while the neural network treats every local scenario as unique. This is true unless some operator is introduced to disable input nodes on the neural network, whereby a more global behavior could be achieved.

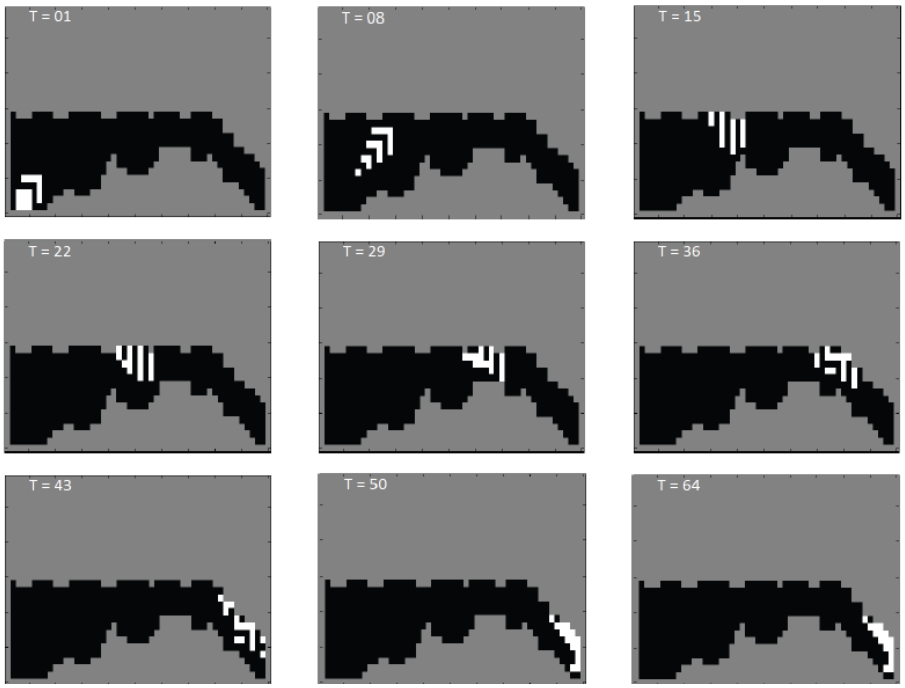

Fig. 6. Snapshots of the best performing individual generated with sequential GA in the training scenario. The gray zone corresponds to the obstacles while white dots are robot cells.

The amount of simulation steps explains the decrease in fitness shown on the scale invariance experiments. This variable was somehow uncontrollable, since there was no clear way to stablish stangnation due to the small local movements that disabled the connectivity in the fitness function. Robots were able to increase their scored connectivity index by travelling with a slight separation from each other, but gathering closely at the end.

We note that for all experiments, the desired direction of locomotion was achieved. This raises GA and NEAT as tools to synthesize decentralized controllers in even more complex tasks than those developed so far (including native 3D displacements and reconfigurations).

In a future work we plan to provide comparisons with previous hand made controllers. A fair direct comparison is not yet possible, since this entails adapting the rule domain to the exact settings used on other studies.

Aknowledgements

This research was funded by Fondecyt project number 11110353. We thank the thorough revision provided by anonymous reviewers.

References

1. Z. Butler, K. Kotay, D. Rus and K. Tomita, "Generic decentralized control for a class of self-reconfigurable robots," in *Proc. of IEEE International Conference on Robotics and Automation*, ICRA, pp. 809-816, May 2002.
2. Z. Butler, K. Kotay, D. Rus and K. Tomita, "Generic decentralized control for lattice-based self-reconfigurable robots," *International Journal of Robotics Research*, vol 23, no 9, pp. 919-937, 2004.
3. A. David, F.H. Bennett, J.R. Koza, "Evolution of intricate long-distance communication signals in cellular automata using genetic programming," in *Proc. of the Fifth Int. Conference on the Synthesis and Simulation of Living Systems*, ALIFE, pp. 513-520, 1996.
4. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., 1989.
5. J. Kubica and A. Casal, "Complex behaviors from local rules in modular self-reconfigurable robots," in *Proc. of IEEE International Conference on Robotics and Automation*, ICRA, pp. 360-367, 2001.
6. K. Kotay and D. Rus, "Generic distributed assembly and repair algorithms for self-reconfiguring robots," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2362-2369, February 2005.
7. K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies.," *Evolutionary computation*, vol. 10, no. 2, pp. 99-127, Jan. 2002.
8. K. Stoy, "Using cellular automata and gradients to control self-reconfiguration," *Robotics and Autonomous Systems*, vol. 54, no. 2. pp. 135-141.
9. M. Yim et al., "Modular Self-Reconfigurable Robot Systems," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 43-52, 2007.