Controlling Cooperative and Conflicting Continuous Actions with a Gene Regulatory Network

Sylvain Cussat-Blanc[1,2], Stéphane Sanchez[1], Yves Duthen[1]
[1] University of Toulouse - IRIT; 2 rue du Doyen Gabriel Marty 31042 Toulouse, France
[2] DEMO Lab, Volen National Center for Complex System;
Brandeis University MS018, 415 South street, Waltham, MA 02454, USA
cussat@brandeis.edu; sanchez@irit.fr; duthen@irit.fr

***Abstract*—Artificial Gene Regulatory Networks (GRN) usually simulate cell behavior in developmental models. However, since 2003, GRN based controllers have been applied to robots to solve problems with few sensors and actuators. In this paper, we present our first steps toward an effective GRN-based controller for intelligent agents in video games. We will also introduce an experiment, the Radbot, where a robot has to handle and manage simultaneously four conflicting and cooperative continuous actions. Finally, we will show how a GRN-based controller can be evolved to solve the Radbot experiment.**

I. Introduction

Artificial Gene Regulatory Networks are usually used in developmental models to perform cell behavior such as migration, division or differentiation. In such models, a GRN can determine, and regulate, cell behaviors through the production of internal and external proteins. Over the past ten to fifteen years, regulatory networks have been proved efficient as a central system for cell based developmental models such as [1], [2]. Since 2003, regulatory networks have been more and more used to control robots [3], [4], [5], [6]. From our past experiences using GRN [7] and based on Nicolau's works, we will try to show that a GRN can make a decent adaptive behavioral controller. More specifically, we think that a GRN can successfully learn how to handle simultaneously several continuous actions to fulfill a specific task. By "continuous actions", we mean parametrized actions that stay active for a while in the simulation and that only evolves through their parameter changes.

To prove this, we have to evaluate how a GRN performs in learning how to keep several competitive and/or cooperative actions active at the same time, and how it can switch from one to another to fulfill its assigned task. To that end, we propose a new experiment, the Radbot. This experiment is designed to evaluate the learning of the management of four continuous actions and, thanks to its easy parametrization, it allows a correct study of how a known GRN [8] converges while learning. If successful, this GRN will be used to produce controllers for agents that need to coordinate several actions, to prioritize one of them or to maintain the best compromise amongst them according to the current situation. Examples of such trained agents can be found in many works related to artificial intelligence in video games as different as car racing [9], first person shooter [10], [11] or real time strategy games [12]. It is however important to note that in most of these works, actions or behaviors are played sequentially and at full intensity while our intend is to obtain, using machine learning, a controller that can play simultaneously different actions and/or mitigate their effects if necessary instead of systematically stopping them.

The following section of this paper will present the Radbot experiment. Section III proposes a method based on a gene regulatory network to solve the Radbot problem. Section IV shows the results of three main experiments with different environments (discrete, continuous, random) and different complexities. The paper concludes with a discussion of obtained results and many perspectives as well for the benchmark as for regulatory networks.

II. The Radbot problem

The Radbot problem is designed as a benchmark to evaluate performance in learning how to manage and coordinate multiple continuous actions (or behaviors) in a dynamic world. More specifically, we are interested in problems where an agent must maintain and manage more than one action to correctly perform in its environment. These actions share critical resources or/and they benefit from each other influence. In such cases, the agent might need to simultaneously perform at least two tasks or it might need to switch continuously from one task to another. The Radbot problem puts an agent in such a situation: the agent will have to handle accurately four parametrized actions that are either conflicting or cooperative to solve the problem. Obviously, any mistake in the management of the four continuous actions will lead to a more or less quick failure.

The problem consists in a robot, the Radbot, which have to go through a linear environment while managing two energy accumulators, A and B, and its internal temperature T. Within this environment, two different radiations, $R1$ and $R2$, increases the temperature of the robot according to their respective intensity.

The Radbot can gradually activate two shields, $S1$ and $S2$, to protect itself from respectively radiations $R1$ and $R2$. The more it activates a shield, the more it absorbs corresponding radiation and the more it prevents the robot from heating up. If temperature T reaches its maximum, the Radbot overheats and is destroyed. The shields share and consume the energy

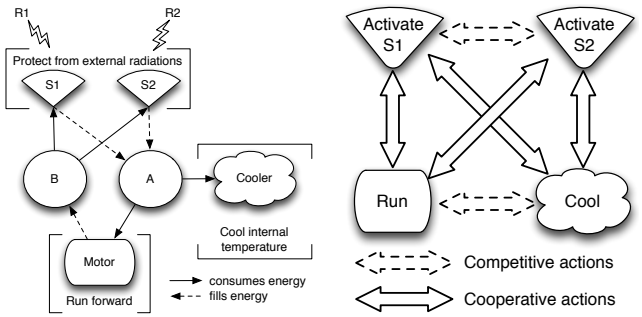


Fig. 1. The Radbot's actions and their relations

from accumulator B. Each shield converts absorbed radiation in energy stored in accumulator A. The Radbot consumes energy from accumulator A to run through the environment, but it produces energy to fill accumulator B doing so. The Radbot can cool down its internal temperature T consuming exclusively energy A. Finally, all actions cannot be fully applied. The sum of activation of the actions cannot exceed 100%. In other words, if the Radbot decides to trigger only one action at 100%, all other actions will be disabled.

Figure 1 summarizes the possible actions of the Radbot and their relations. These actions can be formalized by table I. In this table, each column corresponds to the modification of each parameter of the Radbot's state (lines). All the equation are applied at every time step of the simulation. δ_* can take values between 0 and 1. They correspond to the degree of activation of the corresponding action. The sum of δ_* must be equal to one. It means that all the resources are distributed to all the actions. a_*, b_*, S_{cool} and M_{sp} are positive values set up during the definition of the Radbot. According to their values, they will directly influence the complexity of the problem. Finally, rad_{R1} (respectively rad_{R2}) is the quantity of radiation $R1$ (respectively $R2$) emitted by the environment at the considered time step. For example, the activation of shield $S1$ at δ_{S1} will increase (+) the level of energy A of $a_{S1} * \delta_{S1}$, where a_{S1} is a parameter given during the definition of the Radbot. The run behavior will calibrate the speed of the Radbot with the value $M_{sp} * \delta_{run}$ where M_{sp} is a predefined parameter that expresses the maximum possible speed of the Radbot.

The difficulty of this problem is to coordinate and manage the four actions (run, cool, activate shield $S1$, activate shield $S2$) to go through the maximum distance in the environment: all the actions cannot be fully applied at the same time; the robot has a maximum behavioral load and has to share it out among the different actions. For example, the robot can allocate 25% of its load to each action and, if the robot wants

Level	Activate S1 at δ_{S1}	Activate S2 at δ_{S2}	Cool down at δ_{cool}	Run at δ_{run}
A	$+a_{S1} * \delta_{S1}$	$+a_{S2} * \delta_{S2}$	$-a_{cool} * \delta_{cool}$	$-a_{run} * \delta_{run}$
B	$-b_{S1} * \delta_{S1}$	$-b_{S2} * \delta_d$	–	$+b_{run} * \delta_{run}$
Temp	$+rad_{R1} * \delta_{S1}$	$+rad_{R2} * \delta_{S2}$	$-S_{cool} * \delta_{cool}$	–
Speed	–	–	–	$M_{sp} * \delta_{run}$

TABLE I
EFFECT OF THE DIFFERENT BEHAVIORS ON THE RADBOT'S STATE

to increase the intensity of a specific action, it has to decrease the intensity of another one. Besides, as shown in figure 1, the actions are either conflicting or cooperative: Activate $S1$ and Activate $S2$ are conflicting because they consume the same energy resource B and the activation of one of them competes directly with the activation of the other (preventing the Radbot to protect itself from one of the two radiations). Cool down and Run are conflicting for they share of energy A and cooling down the Radbot prevents to move forward at maximum speed (and vice versa). Activate $S1$ (or $S2$) cooperates with cool down for it fills accumulator A that cooling down needs, and cool down can compensate a partial shielding from activation of $S1$ (or $S2$). At last, Activate $S1$ (or $S2$) cooperates with Run for they produce needed energy of each other and an effective shielding allows a slower run while a faster run allows a more partial shielding.

There are many ways to configure the Radbot: we can configure the gain and the loss of each component for each action; the changes of intensity of both radiations in the environment (discrete values within discrete areas as represented in figure 2, continuous values along the environment by the use of functions such as in figure 3 or random values) and the maximum behavior load of the Radbot. Playing with these parameters allows to implement easy to complex problems.

To generate the necessary inputs to make chose an action in reaction to its environment and its internal variables, the Radbot can use a maximum of six sensors:

- $energy_A$ (0 to 100): current energy level stored in A,
- $energy_B$ (0 to 100): current energy level stored in B,
- $temp_T$ (0 to 100): current internal temperature (0: cold to 100: overheated),
- rad_{R1}^{norm} (0.0 to 1.0): normalized level of radiation $R1$ in current position,
- rad_{R2}^{norm} (0.0 to 1.0): normalized level of radiation $R2$ in current position,
- $distance$: distance to the next switch of maximum intensity of radiations $R1$ and $R2$ (see figures 2 and 3).

In this paper, we propose a first approach to solve the Radbot problem. This approach is based upon an artificial gene regulatory network that regulates the activity of the different continuous actions of the Radbot. This method seems to be well-adapted to this problem. Indeed, the expression of the different genes involved in the regulatory network will represent the expression of the behavior in the Radbot and will be regulated by the network. The next section presents the Banzhaf's regulatory network used in this paper.

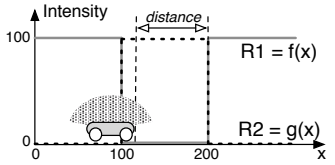


Fig. 2. The Radbot in a discrete environment. *distance* is a sensor of the RadBot that represents the distance to the next radiation switch.

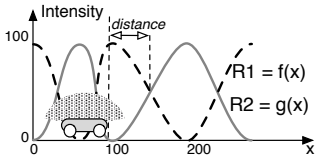


Fig. 3. The Radbot in a continuous environment. *distance* is a sensor of the RadBot that represents the distance to the next radiation switch.

III. The gene regulatory network

A. Background on artificial regulatory networks

Many current developmental models rely on an artificial GRN to control the cells. This system is more or less inspired by gene regulation systems of living beings. In living beings, the cells of organisms have several functions. They are described in the organism genome and their expressions are controlled by the regulatory network [13]. Cells use external signals from their environment to activate or inhibit the transcription of genes. The cells collect external signals through protein sensors localized on the cell membrane. Then, gene expression within a cell determines its behavior.

Eggenberger [14] was one of the first to use a GRN to generate a 3-D organisms able to move by modifying its morphology. In [15], Reil proposed a biologically plausible model, with a genome defined as a vector of numbers. In this model, each gene starts with a particular sequence (0101), named the "promoter". Then, a graph is used to visualize the gene activations and inhibitions over time with randomly generated networks. Observations revealed the existence of several patterns such as gene activation sequencing, chaotic expressions or cyclic expressions. The author also pointed out that the system was able to display self-repairing pattern after random genome deteriorations. Banzahf also described an artificial GRN model strongly inspired by real-world gene regulation [8]. This model is detailed in the next section.

Starting from these two seminal models, various extensions and variations have been explored, for addressing various concerns and applications. Several works addressed Artificial Embryogeny problems with models of GRN ranging from cellular automaton modeling [16] to stripped-down version of GRN combined with complex developmental systems [1], [2], [17]. Some other works have also addressed sequential control problems: using GRN as a control function to map a virtual robot's sensory inputs to its motor actuator values. This has been applied in various setup, from foraging agents [5] to pole balancing [4].

B. The model

In this work, we consider the artificial Gene Regulatory Network (GRN) introduced by [8]. In this model, the network is coded into the genome as a sequence of 32-bit strings (termed *sites*). Each gene in the genome is marked by a particular sequence named the "promoter". When a promoter is detected, the next five sites represent a gene sequence that codes for a protein to be produced. Each site codes for a different molecule of the protein. The concentration of this protein will determine the expression level of the matching gene. To determine the protein's concentration and thus the gene expression level, two sites, coded upstream of the promoter, enhance and inhibit the protein production. The dynamics of enhancer signal e_i and inhibitor signal h_i of a protein i are given by the following equations:

$$e_i = \frac{1}{N}\sum_{j=1}^{N} c_j e^{\beta(u_j^+ - u_{max}^+)} \quad ; \quad h_i = \frac{1}{N}\sum_{j=1}^{N} c_j e^{\beta(u_j^- - u_{max}^-)}$$

where N is the total number of proteins, c_j is the concentration of the protein j, β is a scaling factor, u_j^+ (resp. u_j^-) is the matching degree of the enhancer (resp. inhibitor) site with the protein j and u_{max}^+ (resp. u_{max}^-) is maximum enhancer's (resp. inhibiter's) matching degree observed in the whole genome. The matching degree u_j^+ (resp. u_j^-) consists in counting the number of "1" resulting from the application of a XOR operation to the protein j and the enhancer (resp. inhibitor) pattern. The exponential function increases the impact of high value of gene expression and filter low values.

Finally, the concentration of produced protein p_i follows the differential equation $dc_i/dt = \delta(e_i - h_i)c_i - \Phi(1.0)$, where δ is a scaling factor and $\Phi(1.0)$ constrains the sum of all concentration equals to 1.0.

C. Extension to a computational model

Originally, Banzhaf's artificial GRN is limited to study internal network dynamics. In order to use this model as a control function, Nicolau et al. proposed an extension by adding inputs and outputs to the regulatory network [4] . This extension is detailed in the following.

1) Inputs: Input values are coded with integers that will correspond to existing proteins. These input proteins can be involved in the regulatory process in two different ways: with their signatures to be considered during the matching process (in equations of e_i and h_i) or with their input value to modify the differential equation dc_i/dt of protein concentrations. Here, the second solution has been chosen as it allows a better resolution with regard to a continuous domain of the problem addressed in this paper.

2) Outputs: In order to produce outputs in the regulatory networks, genes are separated into classes: transcription factors *TF-genes* and product proteins *P-genes*. Whereas TF-genes play the roles of regulatory proteins as in the original Banzhaf's model, P-genes are only regulated but do not regulate other proteins: their expression levels provide the desired output signals. These two kinds of genes are identified by introducing two new promoters, whose signatures are chosen so that their probability of occurrence is equivalent and their matching as low as possible.

D. Evolutionary algorithm

1) Evolution strategy: A classical (250+250) evolution strategy (ES) evolves a population of regulatory networks coded by the binary string previously presented. The (250+250) evolution strategy consists in producing 250 offspring from 250 parents and choosing the 250 best genomes to form the next population. The fitness function that evaluates each genome consists of calculating the distance traveled by the Radbot until its destruction or it reach the end of the environement. The evolution strategy evolves the genome in order to maximize this distance.

Genome modifications are only regulated by a common bit-flip mutation operator. The mutation rate is set to 2% at the beginning of the run and adapted by the 1/5 rule of evolution strategies [18]: (1) the mutation rate is doubled when the rate

of successful mutation is higher than 20%; (2) the mutation rate is divided by two when the rate of successful mutation is lower than 20%; (3) the mutation rate is doubled when the number of gene mutations in the population is less than 250 by generation. The regulatory network's genome is randomly initialized. It is then duplicated 9 times with a mutation rate of 2% in order to increase the appearance probability of regulation sites.

2) Problem encoding: In this paper, the 6 inputs and 4 outputs are used to control the Radbot. The 4 outputs directly regulate the 4 possible behaviors of the Radbot (Activate S1, Activate S2, Run and cool). Each input is mapped to a sensor: $input_1$ to $energy_A$, $input_2$ to $energy_B$, $input_3$ to $temp_T$, $input_4$ to rad_{R1}, $input_5$ to rad_{R2} and $input_6$ to $distance$. The inputs are normalized then scaled between 0 and 0.05 in order not to overflow the regulatory network with input proteins. Outputs are also normalized. They express the activation level of each behavior. The sum of the four output values is lower or equal to 1 (which is an inherent property of the regulatory network). Because more than 4 output proteins are generally available in the regulatory network, we decide to split the genome in 4 parts, one for each output. The actual value of an output is given by the maximum concentration of proteins in its matching part.

At last, to help manage the complexity of the problem, shields $S1$ and $S2$ have an activation threshold θ. The shield outputs follow the equation:

$$Out_{shield} = \begin{cases} \frac{Out_{GRN}}{\theta} & \text{if } Out_{GRN} <= \theta \\ 1 & \text{if } Out_{GRN} > \theta \end{cases}$$

where Out_{shield} is the final value sent to the Radbot to activate shields $S1$ or $S2$ and Out_{GRN} is the value read from the regulatory network. 0.4, 0.5 and 0.6 are the 3 values used as threshold θ in the next experiments. They allow to gradually increase the problem complexity and to stress the Radbot more. Next section presents two experiments we have made using the GRN-based approach to control the Radbot.

IV. Experiments

A. Parameters of the Radbot

In the following experiments, the Radbot is set up with identical parameters. It allows us to study the behavior of the Radbot according to the environment and not according to its setup. This setup has been empirically determined, through a set of tests, to generate a rather complex problem that can be solved with reasonable computation time. The Radbot's maximum speed is 10 units per time step. It means that if the Radbot only activates the running action, it will go forward by 10 units at each simulation step. When the Radbot reaches the maximum speed, it consumes 5 units of energy from A and fills B with 12 units of energy.

The activation at full intensity of shield $S1$ (or shield $S2$), within an area that receives x units of radiation R1, consumes $\frac{40}{x}$ units of energy from B and fills A with $\frac{50}{x}$ units of energy. Cooling down at full rate consumes 5 units of energy from A to decrease temperature T by 10 units. Finally, the Radbot energy levels A and B and the temperature T can vary from 0 units to 100 units. When T is equal or greater than 100 units, the Radbot overheat and is (virtually) destroyed. This marks the end of the simulation. At the beginning of each simulation, energy levels of A and of B are set up to 100 units and the Radbot is cold (i.e. T set up to 0 unit).

B. Discrete mode

In the discrete mode, the environment is built as a sequence of areas with different levels of radiations. In this experiment, the environment has 1000 areas. Each area is numbered from 0 to 999 and its size is 100 units. Even areas receive a 10 units of radiation $R1$ and 0 units of radiation $R2$ at each time step and odd areas 10 units of radiation $R2$ and 0 units of radiation $R1$. The 10 units correspond to the temperature T caused at each time step to the Radbot if the shield is activated at 0%. The activation of the shield reduces the heat increase according to the equation presented in table I. The regulatory network has been evolved for each value of shield activation threshold previously given (0.4, 0.5 and 0.6). Figure 4 shows the convergence curves obtained with the three threshold values. The convergence is longer as the threshold increase. As expected, the problem is more complex as the threshold increase.

Figure 5 represents the data extracted from the simulation with the threshold set up at 0.4. Whereas curves (a) and (b) represent the levels of energies, of temperature and of actions activation through the whole simulation, curves (c) focus on a particular region of the simulation. On curves (a) and (b), we can notice a short stabilization period of the GRN before having constant behaviors. After that, all levels remains globally stable, oscillating area after area. Curve (c) focuses on few time steps after the oscillatory stage to detail the behavior of the Radbot. First of all, we can notice that the shields (blue curves) are never activated to their maximums. The regulatory network prefers balance this by pushing up the cooling behavior (dark-green curve). Whereas the shield $S1$ is sufficiently high, the shield $S2$ stays relatively low. In order to protect the Radbot from radiation $R2$, the regulatory network chooses to increase the cooling down and the speed behaviors

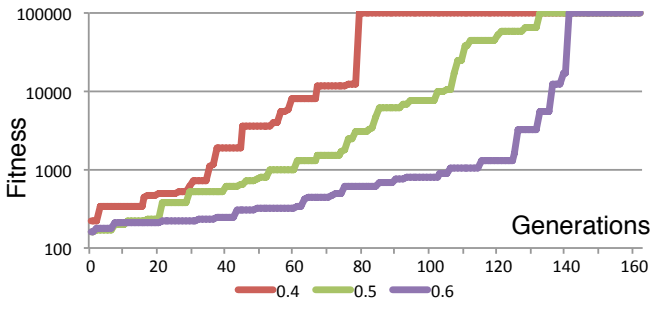
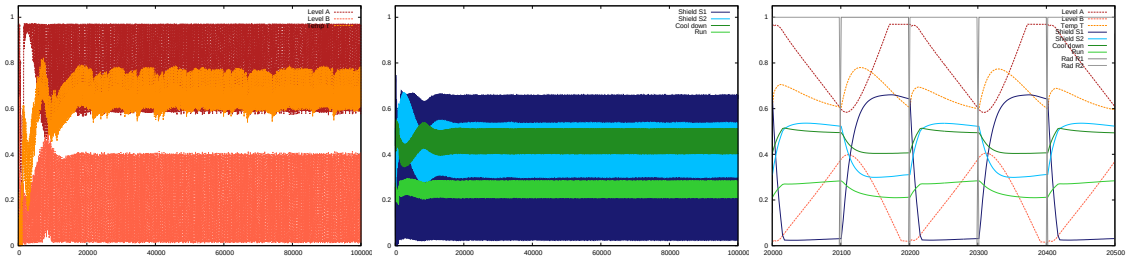


Fig. 4. Influence of the shield's full activation threshold on the convergence of the evolution strategy



(a) (b) (c)

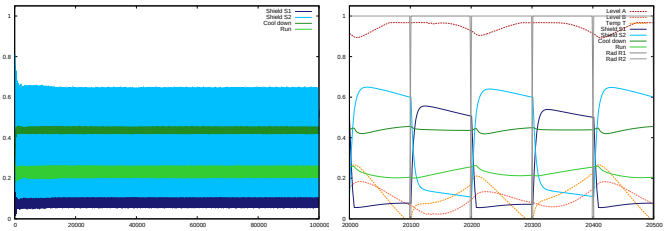
Fig. 5. Discrete Mode, Threshold=0.4 – (a) Internal state of the radbot (b) Behavior levels during all the simulation; (b) Zoom on all the levels between time steps 20000 and 20500.

of the Radbot. However, the dark-orange dotted curve shows the heating of the Radbot that this choice implies.

Finally, we can also notice the variety of possible behavior shifts generated by the regulatory network. It can be smooth such as for cooling down and running behavior, very steep such as during the inhibition of the shield $S1$ or between such as for the activation of both shields.

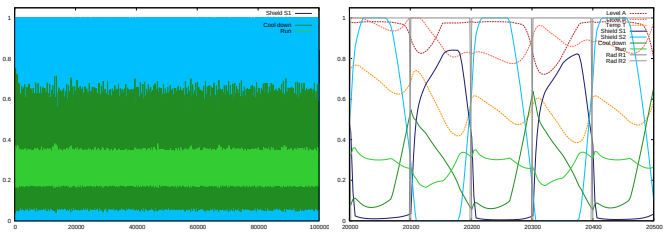
Curves on figures 6 and 7 present the data extracted from the simulation using a threshold value of 0.5 (figure 6) and 0.6 (figure 7). Curves 6(a) and 7(a) represent the behavior levels during the whole simulation. We can notice that the simulation is more complex through the increase of the threshold and this causes a strong optimization of the regulatory network stabilization stage. Indeed, whereas the stabilization stage takes about 12000 steps with the threshold at 0.4 (see curves (a) of figure 5), its duration strongly decreases with the increase of the complexity: about 2500 steps with a 0.5 threshold and 1000 steps with 0.6. This optimization is necessary in order to keep the Radbot alive in the first simulation steps.

Another observation we can make is about the management of different behaviors. Whereas the use of the shield was not very optimal with a threshold value equal to 0.4 (figure 5(c)), a more complex simulation helps a better use of them. First, with threshold equal to 0.5 (figure 6(b)), the transitional stages between shield shifts have been strongly optimized: the regulatory network shift from a shield to the other in very few time steps. When the threshold increase again to 0.6 (figure 7(b)), the shields are really used as expected, the shield $S1$ being used at 80% and the shield $S2$ at 100%. Moreover, even if the transition phases during a shield shift are not so optimized, the regulatory network balances with a strong use of the cool down behavior for a short period.



(a) (b)

Fig. 6. Discrete mode, Threshold=0.5 – (a) Actions intensities, complete simulation; (b) Behavior, energy and temperature levels, time steps 20000 to 20500



(a) (b)

Fig. 7. Discrete mode, Threshold=0.6 – (a) Actions intensities, complete simulation; (b) Behavior, energy and temperature levels, time steps 20000 to 20500

C. Continuous mode

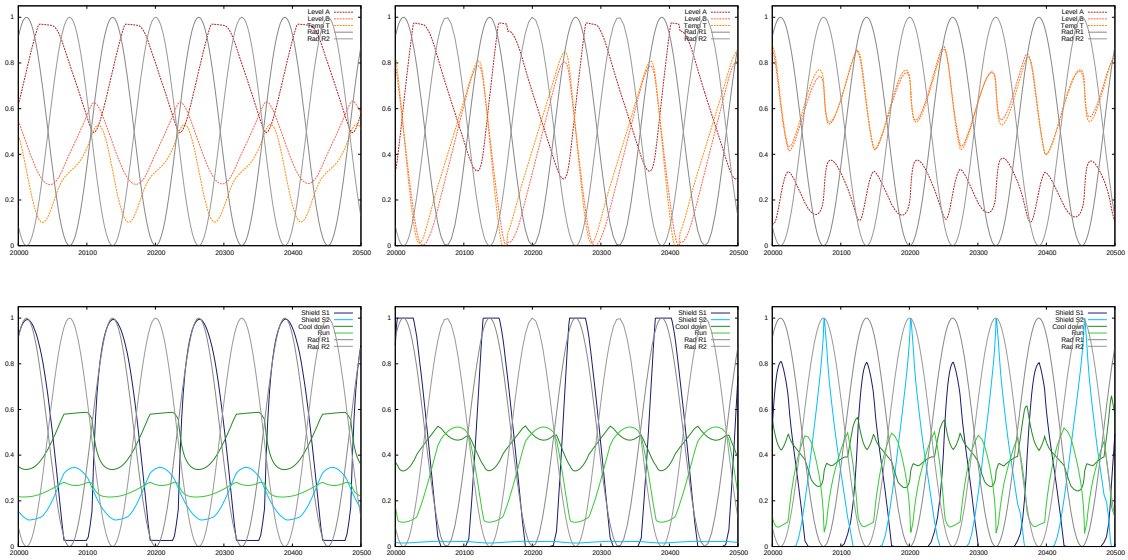
In the continuous mode, two functions replace the areas of the discrete mode. These functions give the levels of radiations $R1$ and $R2$ according to the position of the Radbot in the environment. In this particular experiment, the levels of radiations are given by following equations:

$$rad_{R1} = 5 * sin(x/20) + 5 \quad ; \quad rad_{R2} = -5 * sin(x/20) + 5$$

where x is the position of the Radbot in the environment.

These equations have been made in order to produce sinusoidal radiations in opposite-phases. The size of a phase is equal to $40\pi \cong 125$. In other words, the Radbot will receive exactly the same radiations every time it travels for 125 steps.

Figure 8 represents the curves of both levels of energies and temperature (top) and of the different behavior (bottom) for each possible activation threshold of the shields (from left to right: 0.4, 0.5 and 0.6). We can observe that on every curves, the Radbot compensates the lack of shield $S2$ by a strong amount of cooling. Shield $S1$ is usually sufficiently well-used to protect the Radbot from the corresponding radiation. An interesting behavior has emerged with the threshold 0.5: the Radbot leaves the shield $S2$ and prefers strongly speeding up and cooling down in the same time. This behavior is regulated in order to keep all the level in good ranges. Whereas level



(a) threshold=0.4 (b) threshold=0.5 (c) threshold=0.6

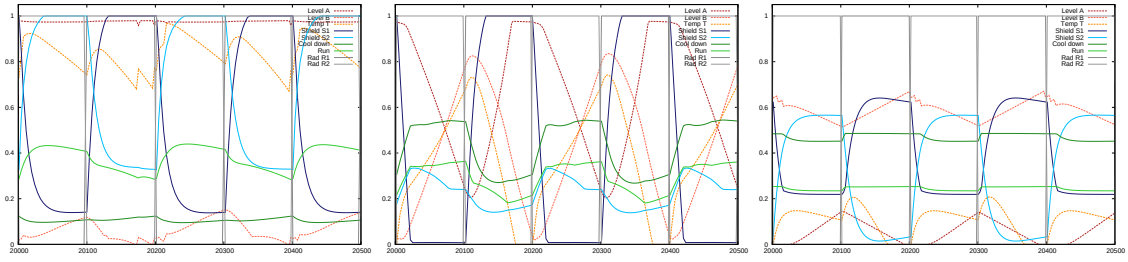
Fig. 8. Continuous mode – Radbot state (first row) and actions intensities (second row) between time steps 20000 to 20500 for the different values of the shield activation threshold (columns).

A strongly decreases and level B and temperature T strongly increase during the fast run, the Radbot strongly slows down and keeps a high level of shield $S1$ in order to regenerate as much as possible all the resources. This behavior was not expected and proposes an interesting alternative to the perfect use of the shields. With a threshold value equal to 0.6, the shields are both globally well managed but transition phases do not match with the radiation curves. Once again, the Radbot compensates this lack of precision by speeding up during the transition phases. The experiment proves the capacity of the regulatory network to answer as well in a discrete environment as in a continuous one. Next section presents the diversity of produced behavior on the discrete problem.

D. Diversity of obtained behaviors

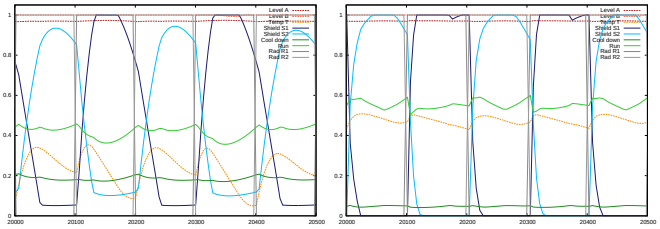
The last experiments present a resolution of the Radbot problem in various cases of environments and complexities. This section focuses on only one of these cases to study the diversity of generated behaviors. To do so, we decide to choose a discrete world with the lower threshold value for the shield activation (i.e. 0.4) so that the convergence of the regulatory network will be as easy as possible.

Figure 9 presents 3 different behaviors obtained. Figure 9(a) has the most expected behavior: the Radbot activates the shields at the maximum. It always keeps a small amount of cooling in order to compensate the temperature accumulated during the transitory phases. The strategy allows the Radbot to keep a relatively high speed, that could be optimized if the shields were completely disabled when there is no corresponding radiation. In figure 9(b), the shield $S1$ is perfectly managed, with a total activation and inhibition consistent with radiation $S1$. Unfortunately, shield $S2$ is almost not used at all. In the areas with radiation $R2$, the Radbot compensates by increasing its speed and more cooling down. It implies huge variations of the energy and temperature levels. In figure



(a) (b) (c)

Fig. 9. 3 different behaviors generated by 3 evolved regulatory networks in a discrete world with a shield activation threshold at 0.4. (a) Both shield are well managed. It allows the Radbot to run faster than usually. (b) The Radbot mainly uses the shield $S1$ and cools down and runs faster in an area with radiation $R2$. (c) Shields are not used as expected in both cases but energies and temperature stay at good levels.



(a) Generation 206 (b) Generation 4429

Fig. 10. Behavior optimization to increase the speed of the Radbot. (a) At the half of its optimization, it takes 24294 time steps to reach the end of the environment. (b) At the end of the optimization, it only takes 18160 time steps.

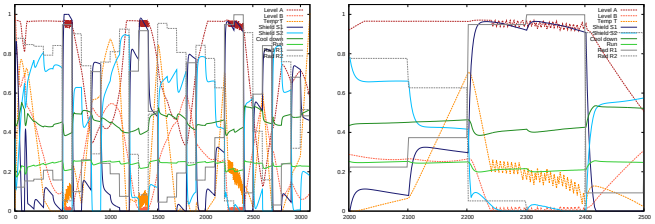


Fig. 11. Results of the RadBot's learning in a random environment. The left figure presentes the global behavior on the 3000 first steps and, the right figure, the same curves are here zooment between 2000 and 2500 steps.

9(c), shields are not so well managed. As always, the Radbot uses run and cool not to be destroyed by radiations. Unlike in figure (b), here the Radbot is able to keep the energy and temperature levels very stable, with a very low heating. This strategy was not expected but seems to be also efficient.

According to the variety of behaviors and especially according to the behavior obtained on figure 9, we can expect the regulatory network to be able to optimize the use of the shields just by giving it one more objective: optimizing the speed of the Radbot. The next experiment shows how adding this parameter to the fitness function optimize each part of the regulatory network.

E. Speed optimization

In this experiment, we decide to optimize the distance covered by the Radbot and its speed. We want the Radbot to run through the environment as fast as possible. The previous experiments were limited to 50000 time steps so that inefficient GRNs would not lock the optimization algorithm. Best GRNs usually reach the end of the environment in about 39000 simulation steps. But this amount of needed steps to perform a successful run do not influence fitness calculation. The fitness of the previous experiments only takes into account the distance traveled by the Radbot. To make the Radbot swifter, we decide to penalize the Radbot with the duration of the simulation. The fitness function is now given by $fitness = distance\_traveled - simulation\_duration$.

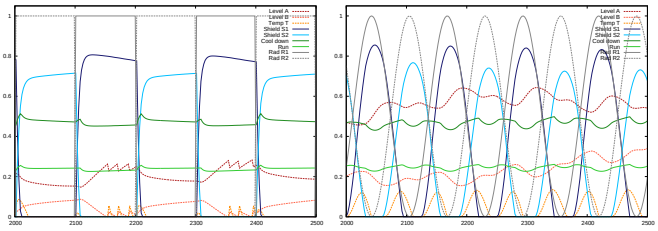
Figure 10 presents the curves of Radbot behavior and levels during the speed optimization phase, after 206 and 4429 generations. The graphics are zoomed after the stabilization stage. The more the optimization goes, the more the shield activation are optimized. First, their activation level of the shields are optimized (a), that allows a decrease of the use of the cooling action and a first increase of the speed. Then, the transition phases between two areas are optimized (b). This optimization is slower than the previous one but allows a new increase of the speed during the transition phases because the cooling action is lesser and lesser necessary.

F. Generalization

The first two experiments showed that the GRN can successfully handle the four actions in both discrete environment and continuous environment. Our main concern now is to evaluate if the GRN can produce generalized behaviors or, in other words, a strategy that can fulfill more than one specific environment. In this preliminary work, our idea is to make the Radbot learn in a random environment, and to apply the best solutions to both discrete and continuous environment. The random environment works as follows: each 100 distance units, the levels of Rad_{R1} and Rad_{R2} are randomly computed with the only limitation that their sum must be equal to 10.

Using the easiest 0.4 threshold, the Radbot never manages to run through all the random environment. In the best cases, the Radbot overheats before the end of the run. However, the results of the Radbot indicate that the GRN can partially handle the new difficulty. Figure 11 shows a typical result in random environment: the Radbot has correctly learned to coordinate the activation shield $S1$ and $S2$ with the presence of radiations $R1$ and $R2$ (activation of $S2$ is superior than $S1$ activation when level of $R2$ is superior than level of $R1$). But the intensity of the shield $S2$ activation is insufficient (i.e. not correctly learned) and the Radbot gradually overheats until the premature end of its run. As in previous experiments, the Radbot stabilizes quickly Run and Cool behaviors at nearly constant intensities. This allows to produce enough energy B to power the shields and to limit overheating enough to sufficiently cool the Radbot while $S1$ is active.

All our experiments in this environment show such slightly imperfect behaviors (in the shown case a lack of $S2$ intensity). A statistical study of diversity of population during evolution shows that, in the random case, the method quickly falls in a local optimum and stays in. This is probably due to the



(a) Discrete environment (b) Continuous environment

Fig. 12. The RadBot running in a (a) discrete environment or in a (b) continuous environment with the GRN evolved in the random environment previously presented.

evolutionary algorithm that only used mutation to explore the solution space and, so, fails to discover other strategies.

Nevertheless, figures 12(a) and 12(b) show that the best imperfect behaviors are generic enough to perform correctly in discrete and continuous environments with 0.4 threshold. In both cases, the Radbot (using the strategy from figure 11) manages to almost run through all the environment. The figures show that the shields are correctly activated and that their activation curves nearly match the curves of $R1$ and $R2$ levels. The main difference with the results of previous experiments is the presence of the slight imperfections: activation of $S2$ is a bit low and there is a slight delay in activation of both shields after a radiations switch.

V. Conclusions and Perspectives

In this paper, we shows how a Genetic Regulatory Network can perform at controlling behaviors, more specifically, at managing simultaneous continuous actions in discrete and continuous environment. To that end, we introduce a new experiment, the Radbot, that allows to simply evaluate how a controller can manage four cooperative and conflicting behaviors and two resources in many various environment. This new problem allows multiple experiments to be run with a given resolution method. The problem is easy to parametrize to generate a large range of complexity easily. It has also been thought to keep the programming cost to connect the Radbot problem implementation to a given method as low as possible.

The GRN, an extended version of Banzhaf's Gene Regulatory Network, already used to solve the pole balancing problem [4] and to generate French flag patterns [7], manages to coordinate almost perfectly the four behaviors to solve the Radbot problem in discrete and continuous environments. These results denote the capacity of the regulatory network to generate an adapted behavior to a dynamic environment. We consider them very encouraging and they must be compared with other existing methods such as NEAT [10], Artificial Neural Networks or Learning Classifier Systems using the same Radbot experiments. Besides its adaptive and learning abilities, the last generalization experiment shows that the GRN seems to have some generalization abilities and we consider that as crucial to use it as a behavioral controller in real world applications.

However, the experiments, and particularly the generalization one, show two major limitations in the used GRN. The first concerns the complexity of the regulatory network : even if the obtained results are promising, the computational cost is still expensive using this method. The evolution strategy has been deployed on 128 CPUs to obtained these results. In our opinion, this complexity comes from two factors. The encoding of the GRN, very close to the natural encoding, generates a lot of non coding DNA which is good in nature to defend from mutations but counterproductive in our computational case. Moreover, the dynamics of the GRN does not allow the production of non existing proteins. In other words, if the concentration of a protein is null, this protein will never be produced anymore. This is a limitation in the case of the behavior generation because it means that an unused action can not be used anymore.

The second one concerns the exploration of solution space. The actual GRN only supports mutation as exploratory method, crossover being too aggressive with the network. Thus, the population converges rapidly towards a local optimum and shows difficulties to explore around it. It will be interesting to consider another GRN more specific to behavioral handling problem. A GRN that allows crossover without intefering with the whole newtork structure should perform a better exploration of the solution space.

References

[1] J. Knabe, M. Schilstra, and C. Nehaniv, "Evolution and morphogenesis of differentiated multicellular organisms: autonomously generated diffusion gradients for positional information." *Artificial Life XI*, vol. 11, p. 321, 2008.

[2] M. Joachimczak and B. Wróbel, "Evo-devo in silico: a model of a gene network regulating multicellular development in 3d space with artificial physics," in *Proceedings of the 11th International Conference on Artificial Life*. MIT Press, 2008, pp. 297–304.

[3] T. Quick, C. Nehaniv, K. Dautenhahn, and G. Roberts, "Evolving embodied genetic regulatory network-driven control systems," *Advances in Artificial Life*, pp. 266–277, 2003.

[4] M. Nicolau, M. Schoenauer, and W. Banzhaf, "Evolving genes to balance a pole," *Genetic Programming*, pp. 196–207, 2010.

[5] M. Joachimczak and B. Wróbel, "Evolving Gene Regulatory Networks for Real Time Control of Foraging Behaviours," in *Proceedings of the 12th International Conference on Artificial Life*, 2010.

[6] L. Schramm, Y. Jin, and B. Sendhoff, "Emerged coupling of motor control and morphological development in evolution of multi-cellular animats," *Advances in Artificial Life*, 2011.

[7] S. Cussat-Blanc, N. Bredeche, H. Luga, Y. Duthen, and M. Schoenauer, "Artificial gene regulatory networks and spatial computation: A case study," in *Proceedings of the European Conference on Artificial Life (ECAL'11)*. MIT Press, Cambridge, MA, 2011.

[8] W. Banzhaf, "Artificial regulatory networks and genetic programming," *Genetic Programming Theory and Practice*, pp. 43–62, 2003.

[9] J. Muñoz, G. Gutierrez, and A. Sanchis, "A human-like TORCS controller for the Simulated Car Racing Championship," in *Proceedings 2010 IEEE Conference on Computational Intelligence and Games*, August 2010, pp. 473–480.

[10] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE Transactions on Evolutionary Computation*, pp. 653–668, 2005.

[11] S. Priesterjahn, O. Kramer, E. Weimer, and A. Goebels, "Evolution of human-competitive agents in modern computer games," in *In Proceedings of the IEEE Congress on Evolutionary Computation*, 2007.

[12] V. Scesa, C. Raevsky, S. Sanchez, H. Luga, and Y. Duthen, "Rule fusion for the imitation of a human tutor." in *CIG*, G. N. Yannakakis and J. Togelius, Eds. IEEE, 2010, pp. 154–161.

[13] E. H. Davidson, "The regulatory genome: gene regulatory networks in development and evolution," *Academic Press*, 2006.

[14] P. Eggenberger, "Evolving morphologies of simulated 3d organisms based on differential gene expression," in *Proceedings of the Fourth European Conference on Artificial Life*. MIT Press, 1997.

[15] T. Reil, "Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny," *Advances in Artificial Life*, 1999.

[16] A. Chavoya and Y. Duthen, "A cell pattern generation model based on an extended artificial regulatory network," *Biosystems*, vol. 94, no. 1-2, pp. 95–101, 2008.

[17] R. Doursat, "Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering," *Organic Computing*, pp. 167–200, 2008.

[18] I. Rechenberg, "Evolution strategy," *Computational Intelligence: Imitating Life*, pp. 147–159, 1994.