Genetically-regulated Neuromodulation Facilitates Multi-Task Learning

Sylvain Cussat-Blanc
University of Toulouse
21 Allée de Brienne
31042 Toulouse, France
cussat@irit.fr

Kyle Harrington
Beth Israel Deaconess Medical Center
Harvard Medical School
02215 Boston, MA
kharrin3@bidmc.harvard.edu

ABSTRACT

In this paper, we use a gene regulatory network (GRN) to regulate a reinforcement learning controller, the State-Action-Reward-State-Action (SARSA) algorithm. The GRN modulates SARS's learning parameters: learning rate, discount factor, and memory depth. We have optimized GRNs with an evolutionary algorithm to regulate these parameters on specific problems but with no knowledge of problem structure. We show that GRN-regulated SARSA performs equally or better than the SARSA with fixed parameters. We then extend the GRN-regulated SARSA algorithm to multi-domain problem generalization, and show that GRNs trained on multiple problem domains can generalize to previously unknown problems with no further optimization.

Categories and Subject Descriptors

I.2.6 [**Learning**]:

Keywords

Reinforcement learning, Gene regulatory network, Parameter control, Multi-task Learning, Neuromodulation

1. INTRODUCTION

2. REINFORCEMENT LEARNING

Reinforcement learning is a reward-based learning algorithm that allows agents to learn from experience. More formally, reinforcement learning (RL) is a mathematical framework for learning from a reward signal that is derived from Bellman's equation for optimal control [13]. One of the most important forms of RL is temporal-difference (TD) RL. TD-RL is a method for learning optimal behavior from changes in state and reinforcement by error prediction [12]. TD-RL agents learn an expected return that will be received after taking an action in any state. Strong correlations with this type of error predictive behavior have been found in studies of dopamine neurons [10]. This line of research has continued and is now been supported by fMRI data of reward processing for tastes, money, and love [6].

TD-RL is used to solve Markov decision processes, which are an extension of Markov chains to problems framed in terms of state, action, and reward. Reward signals (such as reinforcement of dirt cleanup) are geometrically encoded in a table which associates action preferences with states. The basic TD(γ) algorithm updates one state-action association at a time which prohibits sequence learning. Eligibility traces are used to associate reward with sequences of actions by reinforcing a weighted history of most recent actions. In this study the online version of TD-RL, SARSA (short for, state-action-reward-state-action), is used. A review of the nuances of reinforcement learning can be found in [13].

Transfer and multi-task learning has recently been receiving attention within the reinforcement and machine learning communities [**?**, **?**].

2.1 SARSA Algorithm

We include a few of the key equations from the RL algorithm which are employed. If we are in state, s_t at time t, then we will take some action a_t which will bring us a reward r_t. This action will also cause us to transition to a new state, s_{t+1}. The SARSA algorithm learns a Q-function, which maps a value to each state-action pair, (s_t, a_t). From each state multiple actions, A_t, may be taken which may be a function of s_t (for example, an obstacle may prevent an action in a given state). Given an optimal Q-function the best action to take is

$$argmin_{a_t \in A_t} Q(s_t, a_t). \tag{1}$$

The Q-function is approximated by SARSA with the following update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big] \tag{2}$$

where α is the learning rate, and γ is the discounting factor. Given only this update rule it can be difficult to compute the Q-value for state-action pairs which indirectly contribute to obtaining a reward. This update method propagates information only to the preceeding state-action pair, for those that are very distant from the reward, such as in the case of maze solving problems, this can require a large number of repeated trials. However, this problem of reward propagation can be partially aleviated by the use of eligibility traces. Eligibility traces store an accumulating trace of state-action pairs. The "memory" of these state-action pairs can be tuned

GECCO'15, July 11-15, 2015, Madrid, Spain.

with the trace decay parameter λ. Eligibility traces are updated with

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \end{cases} \quad (3)$$

By combining the error predictive capabilities of TD-RL with the state-action sequence memory of eligibility traces we can amplify the effects of our reward and speed up the learning process. When performing on-policy learning it is important to ensure that a sufficient amount of exploration occurs. To this end the ϵ-greedy method is used, where a random action is taken with $p(\epsilon)$, otherwise the agent's most preferred action is taken. However, the RL algorithm can still fail to capitalize on rarely experienced rewards.

3. GENE REGULATORY NETWORK

Our model uses an optimized network of abstract proteins. The inputs of the agent are translated to protein concentrations that feed the GRN. Output proteins regulate the reinforcement learning parameters previously described. This kind of controller has been used in many developmental models of the literature [7, 5, 3] and to control virtual and real robots [14, 9, 8, 4].

We have based our regulatory network on Banzhaf's model [1]. It is designed to be as close as possible to a real gene regulatory network but neither to be evolved nor to control any kind of agent. However, Nicolau used an evolution strategy to evolve the GRN to control a pole-balancing cart [9]. Though this experiment behaved consistently, the evolution of the GRN has been an issue. We have decided to modify the encoding of the regulatory network and its dynamics. In our model, a gene regulatory network is defined as a set of proteins. Each protein has the following properties:

- The protein *tag* coded as an integer between 0 and p. The upper value p of the domain can be changed in order to control the precision of the GRN. In Banzhaf's work, p is equivalent to the size of a site, 16 bits.
- The *enhancer tag* coded as an integer between 0 and p. The enhancer tag is used to calculate the enhancing matching factor between two proteins.
- The *inhibitor tag* coded as an integer between 0 and p. The inhibitor tag is used to calculate the inhibiting matching factor between two proteins.
- The *type* determines if the protein is an *input* protein, the concentration of which is given by the environment of the GRN and which regulates other proteins but is not regulated, an *output* protein, the concentration of which is used as output of the network and which is regulated but does not regulate other proteins, or a *regulatory* protein, an internal protein that regulates and is regulated by other proteins.

The dynamics of the GRN is calculated as follow. First, the affinity of a protein a with another protein b is given by the enhancing factor u^+_{ab} and the inhibiting u^-_{ab}:

$$u^+_{ab} = p - |enh_a - id_b| \quad ; \quad u^-_{ab} = p - |inh_a - id_b| \quad (4)$$

where id_x is the tag, enh_x is the enhancer tag and inh_x is the inhibiting tag of protein x.

The GRN's dynamics are calculated by comparing the proteins two by two using the enhancing and the inhibiting matching factors. For each protein in the network, the global enhancing value is given by the following equation:

$$g_i = \frac{1}{N}\sum_j^N c_j e^{\beta(u^+_{ij} - u^+_{max})} \quad ; \quad h_i = \frac{1}{N}\sum_j^N c_j e^{\beta(u^-_{ij} - u^-_{max})} \quad (5)$$

where g_i (or h_i) is the enhancing (or inhibiting) value for a protein i, N is the number of proteins in the network, c_j is the concentration of protein j and u^+_{max} (or u^-_{max}) is the maximum enhancing (or inhibiting) matching factor observed. β is a control parameter described hereafter.

The final modification of protein i concentration is given by the following differential equation:

$$\frac{dc_i}{dt} = \frac{\delta(g_i - h_i)}{\Phi} \quad (6)$$

where Φ is a function that keeps the sum of all protein concentrations equal to 1.

β and δ are two constants that set up the speed of reaction of the regulatory network. In other words, they modify the dynamics of the network. β affects the importance of the matching factor and δ affects the level of production of the protein in the differential equation. The lower both values, the smoother the regulation. Similarly, the higher the values, the more sudden the regulation.

4. NEUROMODULATION

In living organisms, neuromodulators are neuropeptides or small molecules, such as dopamine and serotonin. The production of these substances within the cell is controlled by gene regulatory networks. Neuromodulators change the behavior of neural networks within individual neurons, amongst neighboring neurons, or throughout the entire network. Neuromodulation has been found to be pervasive throughout the brain, and can have drastic consequences on the behavior of neurons and neuronal circuits [**?**, **?**, **?**]. A particularly applicable example in the realm of robotics is the neuromodulation of motor signals produced by central pattern generators in the brain and spinal cord [**?**]. It has been found that neuromodulators tune and synchronize neuromuscular signals [**?**].

We have already noted that the temporal difference learning algorithm for error prediction has been observed in neural substrates [**?**]. Dopamine neurons of the ventral tegmental area (VTA) and substantia nigra exhibit this error predictive behavior. The dopamine system is itself a neuromodulatory system. While the temporal difference learning algorithm extends ideas of reward processing to engineering, there are models of the dopamine system with closer ties to biology [**?**]. These models also confirm the error predictive behavior found in the brain for a variety of physiological data including reaction-time and spatial-choice tasks. Dopamine is an important neuromodulator, especially in learning, but it is but one of many neuromodulatory substances found in the brain. An extensive review of computational models of neuromodulation can be found in [**?**], and some recent models are reviewed in [**?**]. In this study we extend our previous observations on the relationship between evolved neuromodulator-producing GRNs and learned behaviors [**?**].

4.1 Regulating parameters

With the intention of simplyfing the computational model, all the molecular pathway of neuromodulation has been deliberately ignored and only the consequences of neuromodulation on learning behaviors are targetted. Therefore, the artificial gene regulatory network presented in the previous section directly regulates the learning parameters of the SARSA algorithm. Three learning parameters are considered in this work: the learning rate α, the discount factor γ and the memory depth λ.

To do so, the GRN uses three inputs that describes the current performance of the agent in the environment. They have been chosen to be problem-independent: one of our goal is to reduce the configuration of our neuromodulation architecture to have minimum changes to applied when the problem changes. The first input describes the duration since the beginning of current episode: the concentration of this first input protein increases when the agent takes too much time to solve the problem or is getting closer to the end of the episode. The concentraction $C_{I_1}(t)$ of this input protein at time step t is calculated as follows:

$$C_{I_1}(t) = e^{-\frac{t^2}{t^2_{max}}} \tag{7}$$

where t_max is the expected duration of an episode. The second output protein's concentration $C_{I_2}(t)$ describes the quality of the current sequence of actions in term of rewards, smoothed on 25 steps:

$$C_{I_2}(t) = \frac{\sum_{s=1}^{25} \Big((25-s) \times q_s(t-1)\Big)}{\sum_{s=1}^{25} s} \tag{8}$$

$$\text{with } q_s(t) = \begin{cases} 1 + 1000\frac{q.e}{q.q*e.e} & \text{if} \frac{q.e}{q.q*e.e} > 0.4995 \\ 0 & \text{otherwise} \end{cases}$$

where e is a vector that contains the current possible action rewards and q is a matrix that contains all the state/action rewards encountered by the agent at time step t. The aim of this input is to capture the quality of the current state according to the past states the agent has visited. Finally, the third input protein's concentration $C_{I_3}(t)$ informs the GRN about the 25-step smoothed average reward the agent can obtain in its current state:

$$C_{I_3}(t) = \frac{\sum_{s=1}^{24} \Big((25-s) \times C_{I_3}(t-s)\Big) + 25 \times \sum_{r_e \in e} r_e}{\sum_{s=1} 25s} \tag{9}$$

where e is the vector of current possible rewards. This input informs the GRN whether or not the agent is stranded within a long-term decreasing reward.

In addition to these inputs, the GRN uses four output proteins to regulate the learning parameters:

- the output protein O_n, which concentration C_n normalizes the concentration of other outputs[1],
- the output protein O_α of concentration C_α, which provides the value for α to the SARSA algorithm with $\alpha = C_\alpha/(C_\alpha + C_n)$,

[1]this is generally used in regulatory networks to obtain output values in $[0, 1]$

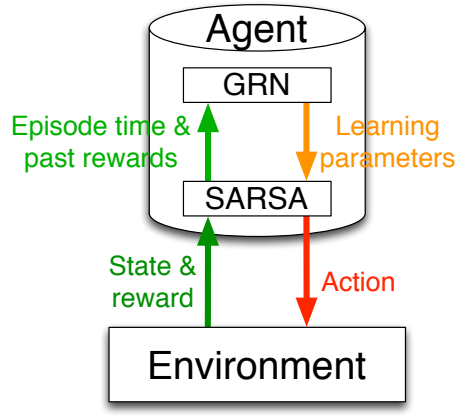

Figure 1: At every time step, SARSA updates the GRN inputs. The GRN returns updated learning parameters that will be used by the SARSA algorithm.

- the output protein O_γ of concentration C_γ, which provides the value for γ to the SARSA algorithm with $\gamma = C_\gamma/(C_\gamma + C_n)$,
- the output protein O_λ of concentration c_λ, which provides the value for λ to the SARSA algorithm with $\lambda = C_\lambda/(C_\lambda + C_n)$.

As depicted by figure 1, the GRN updates SARSA's learning parameters at every time step, before SARSA updates its internal variables and prediction. The GRN returns the learning parameters SARSA uses for its own decision step.

4.2 GRN Optimization

Before using a gene regulatory network for neuromodulation, the network of proteins needs to be optimized. In this paper, we use an adapted genetic algorithm inspired from the NEAT algorithm [11]. Three features are modified in comparison to a standard genetic algorithm:

- the *initialization* of the algorithm - as opposed to initializing with individuals randomly sampled from the complete distribution, only small networks are used in the initial population so as to allow for a more progressive complexification,
- the *speciation* protects newly appeared solutions by giving them some time to optimize their structures before competing with the whole population, and
- the *alignment crossover* with the use of a distance metric between proteins to keep subnetwork architecture during the crossover operation.

This modified algorithm has been proven to converge faster to better solutions. More details can be find in [2].

During its optimization, each GRN is evaluated independently on a given problem with 25 reruns in order to reduce stochasticity impact of the problems on the fitness. The fitness being problem dependent, more explanations will be provided in the experience section. To avoid any memory biais, SARSA is completely resetted before each evaluation.

!!! Provides GREAT's parameters here !!!

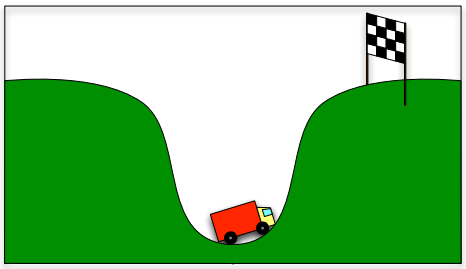

Figure 2: In the mountain car problem, a car must escape from a valey. It cannot climp the hill without moving backward first to gain speed. The agent receives a reward only when it escapes.

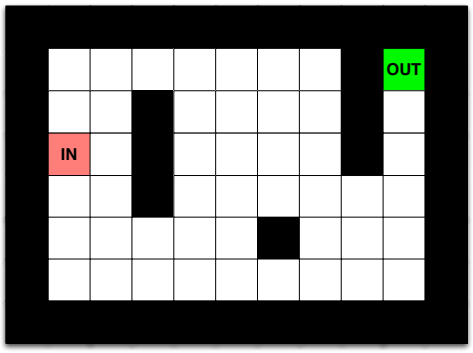

Figure 3: In the maze problem, the agent have to find its way in a maze from the start (in red) to the end (in green). The agent receives a reward only when it escapes.

5. EXPERIMENTS

Parameter sampling for SARSA. Range sampled over α, γ, and λ. What ϵ was used.

5.1 Problems

5.1.1 Mountain Car

5.1.2 Maze

5.1.3 Puddle World

5.1.4 Maze

Originally introduced by Sutton as an example for deterministic problem solving with RL, maze [**?**].

Evolutionary programming has previously been applied to a multi-task version of Sutton's maze problem [**?**].

5.1.5 Mountain Car

Introduced to test RL in sparse coding [**?**]

5.1.6 Puddle World

Stochastic

Introduced to test RL in sparse coding [**?**]

5.1.7 Actor critic pendulum

citation [13].

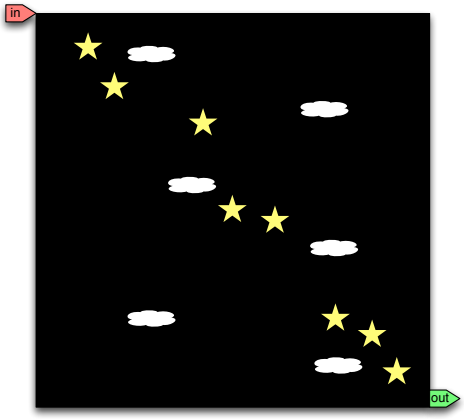

Figure 4: In the puddle world problem, the agent have to find its way in a dark environment from the start (in red) to the end (in green). Local rewards are given all along its exploration and when it escapes.

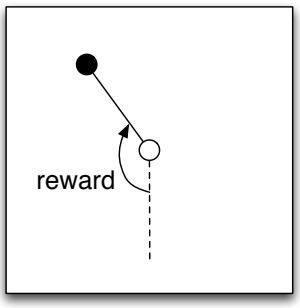

Figure 5: In the actor critic pendulum, the agent has to balance a pendulum to the highest possible position. The reward is given by the cosinus of the angle between the pendulum and the resting position.

	α	γ	λ
Mountain car	0.071429	1.0	0.928571
Maze	1.0	0.928571	0.928571
Puddle world	0.057142	0.928571	0.5
Actor critic pendulum	0.05	0.928571	0.785714

Table 1: Fixed learning parameters for SARSA obtained by parameter sampling

5.2 Training GRN on one specific problem

In this first experience, we have trained our GRN-based neuromodulation model independently on each problem above mentionned. To evaluate the gain provided by neuromodulation, we first determine the best fixed learning parameter to use with SARSA on each problem. We use parameter sampling on α, γ and λ in $[0, 1]$ with 0.0714 steps (15 evaluations). Each evaluation is averaged on 10 replicates in order to reduce the randomness of the problems. At the end of the parameter sampling stage, we chose the best fixed learning parameters for a given problem by selecting the tuple with the highest fitness. These parameters are given in table 1.

5.2.1 *Mountain car*

First, the GRN is trained on Mountain car. Figure 7 shows the convergence curve of the genetic algorithm on this particular problem. We can observe that the best GRN is found very quickly (green curve) before being slowly optimized to regulate the learning parameters more efficiently. Figure 6 plots the behavior of the best GRN obtained after 150 iterations. This figure plots the input protein concentrations and the three learning parameters calculated by the gene regulatory network. The GRN maintains the parameter values almost constant over the time except at the beginning of each episode where α and γ are increase to explore more, certainly due to the novelty of the environment. In this experience, it has also to be noticed that λ is kept to zero all over time. On this simple problem, memory might not be necessary.

This GRN is then compared to the SARSA algorithm with fixed learning parameters on 100 independent reruns of mountain car with different different random seed. The point is to evaluate the capacity of both approaches to handle the noise generated by randomness. Figure 10 shows the results obtained for each episode averaged on the 100 runs. We can observe that neuromodulation with GRN trained on the mountain car (in green) beats SARSA with fixed parameters (in red) on every episodes. More specifically, the first episodes (from 1 to 3) show that GRN-based neuromodulation learns faster than SARSA and later episodes (from 3 to 25) show that neuromodulation solves the problem faster than SARSA.

5.2.2 *Maze*

The same procedure has been used to evolve a gene regulatory network to regulate SARSA's learning parameters on the maze problems. Figure 9 presents the behavior of the best GRN obtained after evolution. The regulation broadly differs from the one obtained on the mountain car problem. This GRN starts with very high values for all learning parameters with the aim to explore widely the environment. Then, with sucessfull episodes incoming, the learning rate α and the memory depth λ decrease to exploit the behavior

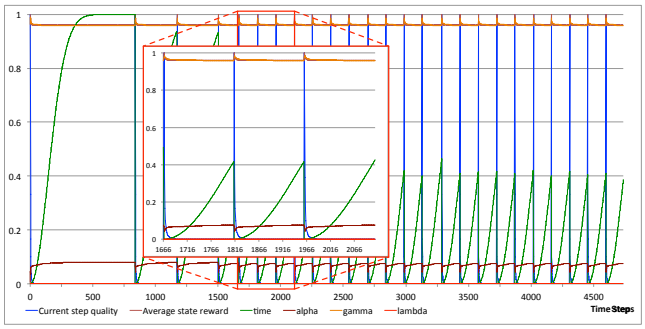

Figure 6: Moutain car: Regulation of the learning parameters of the best GRN obtained.

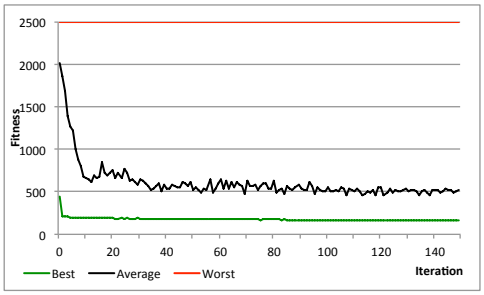

Figure 7: Mountain car: Convergence curve of the genetic algorithm (lower is better - time to escape the valley).

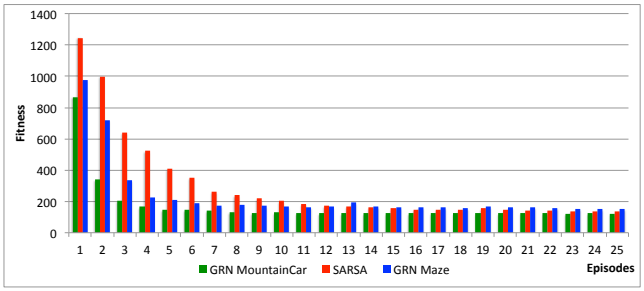

Figure 8: Mountain car: Comparison of the fitnesses per episode (abscissa) obtained by our neuromodulation with a GRN trained on mountain car (green), with a GRN trained on maze (blue) and with parameter-fixed SARSA (red). Results are averaged on 100 independent runs.

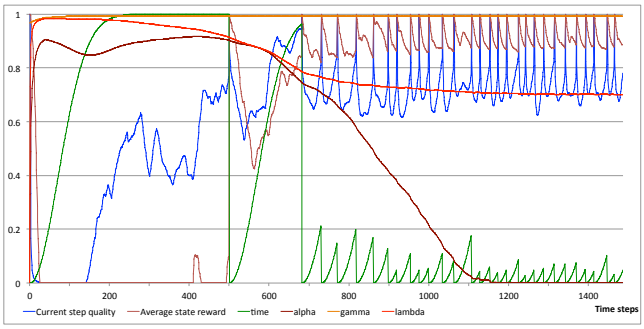

Figure 9: Maze: Regulation of the learning parameters of the besst GRN obtained.

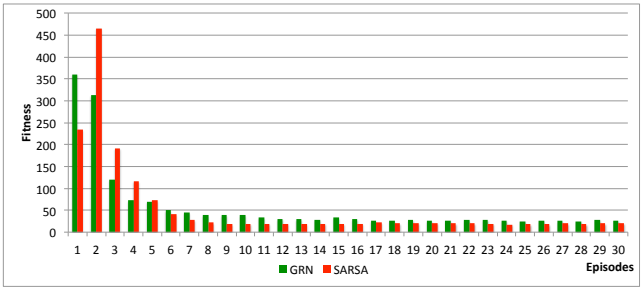

Figure 10: Maze: Comparison of the fitnesses per episode (abscissa) obtained by our neuromodulation with a GRN trained on maze (green) and with parameter-fixed SARSA (red). Results are averaged on 100 independent runs.

learnt during the first scenarios. γ is kept very high all along the simulation.

When compared to parameter-fixed SARSA (figure **??**), we can observe that the GRN is learning faster than SARSA (episodes 2 to 5) due to its high learning rate at the begining of the simulation. However, the avantage turns to fixed-parameter SARSA in the remaining episodes: SARSA is doing slightly better than neuromodulation from episode 6 to the end.

5.2.3 Puddle world

5.2.4 Actor critic pendulum

Number of GRN runs, number of generations

5.3 Generalization of the regulation

Generalizing from which problems to which other problems. Does training on stochastic and testing on deterministic lead to better GRNs than training on deterministic then testing on stochastic?

6. CONCLUSION

7. FUTURE WORK

Introduce a GRN for regulation of swarm controllers [**?**].

8. ACKNOWLEDGEMENTS

Kyle Harrington is supported by NIH grant 2T32HL007893-16A1.

9. REFERENCES

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

[2] S. Cussat-Blanc, K. Harrington, and J. Pollack. Gene regulatory network evolution through augmenting topologies. *IEEE Transaction on Evolutionary Computation*, 2015, to be published.

[3] S. Cussat-Blanc, J. Pascalie, S. Mazac, H. Luga, and Y. Duthen. A synthesis of the Cell2Organ developmental model. *Morphogenetic Engineering*, 2012.

[4] S. Cussat-Blanc, S. Sanchez, and Y. Duthen. Controlling cooperative and conflicting continuous actions with a gene regulatory network. In *Conference on Computational Intelligence in Games (CIG'12)*. IEEE, 2012.

[5] R. Doursat. Facilitating evolutionary innovation by developmental modularity and variability. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 683–690. ACM, 2009.

[6] S. N. Haber and B. Knutson. The reward circuit: linking primate anatomy and human imaging. *Neuropsychopharmacology*, 35(1):4–26, 2009.

[7] 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*, pages 297–304. MIT Press, 2008.

[8] 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.

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

[10] W. Schultz, P. Apicella, and T. Ljungberg. Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *The Journal of Neuroscience*, 13(3):900–913, 1993.

[11] K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[12] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[13] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[14] J. Ziegler and W. Banzhaf. Evolving control metabolisms for a robot. *Artificial Life*, 7(2):171–190, 2001.