Genetically-regulated Neuromodulation Facilitates Multi-Task Learning

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

Kyle I. S. 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 SARSA'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-task 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**]: Parameter learning

Keywords

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

1. INTRODUCTION

Transfer and multi-task learning has recently been receiving attention within the reinforcement and machine learning communities [18, 33]. The ability to generalize between learnable tasks is a challenging ambition, that is far from fully solved, but is achievable, as evidenced by biological organisms capable of generalizing across multiple tasks. In this work we use an evolutionary algorithm to optimize gene-regulatory networks (GRNs) that dynamically tune the parameters of a reinforcement learning (RL) algorithm. This genetically-regulated neuromodulation (GRNM) extends previous results where we showed that GRNM can enhance the learning of agents beyond the performance of traditional fixed parameter RL [14]. We apply GRNM to additional problem classes, and show that GRNs evolved on agents learning a subset of problems can facilitate the learning of agents on previously unencountered problems.

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 [32]. 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 [30]. 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 [25]. This line of research has continued and is now been supported by fMRI data of reward processing for tastes, money, and love [12].

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 are learned and 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 [32].

We include a few of the key equations from the SARSA algorithm. 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 the 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

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

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

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

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

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

By combining the error predictive capabilities of SARSA 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

Artificial gene regulatory networks (GRNs) are a class of biologically-inspired algorithms. In living systems, gene regulatory networks are used within the cell to control DNA transcription and, correspondingly, the phenotypic gene expression. Although the inner workings of the cell are governed by a large collection of complex machines, simplified models of cells as entities with protein sensors and actuators both exhibit complex behavior and offer insights into natural systems [5]. These protein sensors represent receptor molecules localized to the cellular membrane which transduce external activity into excitatory and/or inhibitory regulatory signals. Cells use external signals collected from protein sensors localized on the membrane to activate or inhibit the transcription of the genes.

Our computational 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 [15, 7, 3] and to control virtual and real robots [35, 23, 16, 4].

In our model, a gene regulatory network is defined as a set of proteins. Each protein has the following properties:

- The *concentration* is the quantity of each protein avalaible in the network. This concentration influences the regulation of other proteins: the higher the concentration, the higher the enhancing or the inhibiting impact on other proteins.

- 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}:

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

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:

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

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:

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

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 [6, 19, 20]. 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 [17]. It has been found that neuromodulators tune and synchronize neuromuscular signals [34].

We have already noted that the temporal difference learning algorithm for error prediction has been observed in neural substrates [26]. 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 [21]. 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 [10], and some recent models are reviewed in [19]. In this study we extend our previous observations on the relationship between evolved neuromodulator-producing GRNs and learned behaviors [14].

4.1 Regulating parameters

With the intention of focusing on the optimization of neuromodulator GRNs, the physical mechanisms underlying neuromodulation are only loosely used to inspiration and the neuromodulation of learning behaviors was optimized. Neuromodulation has been considered in the context of RL [8, 27, 9], but this previous work has not focused on the implications of neuromodulation on problem solving capacity. In this work we utilize the artificial gene regulatory network presented in the previous section to regulate 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:

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

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

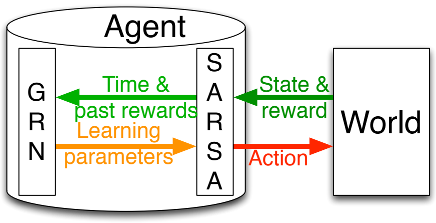


Figure 1: At every time step, SARSA updates the GRN inputs. The GRN returns updated learning parameters that will be used by the SARSA algorithm.

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 $\alpha = C_\alpha/(C_\alpha + C_n)$,
- 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 [28]. 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,

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

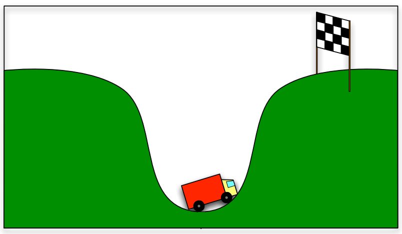


Figure 2: Mountain car

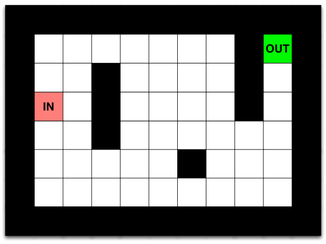


Figure 3: Maze

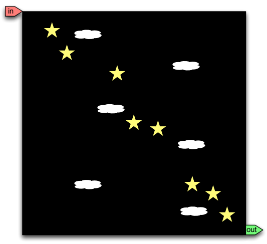


Figure 4: Puddle world

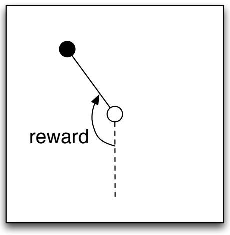


Figure 5: Acrobat

Parameter	value
Population size	500
Initial duplicates	19
Speciation threshold	0.15
Species min size	15
Selection	3-players tournament
Crossover rate	0.2
Mutation rate	0.8

Table 1: Parameters used for evolutions

- 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 on this evolutionary algorithm 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. The parameters used to evolved the GRNs in this work are given by table 1. They are the same for all evolutions.

5. EXPERIMENTS

5.1 Problems

We have studied GRNM on four classical problems: mountain car, maze, puddle world and acrobat. In all cases agents do not have prior information about the specific task. Before detailling the results obtained with GRNM, this section presents each problems.

5.1.1 Mountain Car

The mountain car problem was introduced by Moore to help broad applicability of dynamic programming [22]. In the mountain car problem, depicted by figure 2, the agent is placed at the bottom of a valley with no initial velocity and must drive to the top of the right hill. The agent can take one of 3 actions: full thrust forward, full thrust in reverse, or no thrust. The difficulty of the task lies in the fact that the right hill is too steep to be climbed at full thrust, and thus the agent must first reverse up the left hill to gain sufficient momentum to climb the right hill. The agent experiences a negative reward for all time steps that it has not reached the top of the right hill, at which point the episode is completed. In the GRNM architecture and especially during the optimization stage, the fitness of a training run is given by the number of step necessary to complete a series of 25 episodes.

5.1.2 Maze

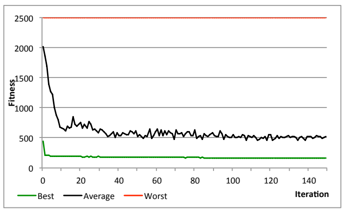
Depicted in figure 3, the maze task was originally introduced by Sutton as an example problem for demonstrating the capabilities of initial reinforcement learning algorithms [29]. The maze is a 6 by 9 grid with 7 obstacles, a start, and a goal position. From any position, the agent may take any of 4 actions (right, left, up, and down); however, when the action would move the agent out-of-bounds or onto an obstacle, the agent remains in place. The only time a reward is received is when the agent reaches the goal, at which point the episode is completed. Although this problem and its variants have been studied by a number of groups, of particular interest is the Hanada's study which applies evolutionary programming to a multi-task version of Sutton's maze [13]. The fitness of a training run is given by the number of step necessary to complete a series of 30 episodes.

5.1.3 Puddle World

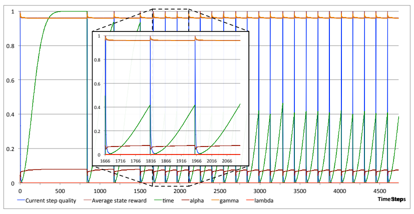
The puddle world problem was also introduced by Boyan and Moore [1], but was later presented in greater detail by Sutton [31]. As presented in figure 4, agents solving the puddle world task exist in a 2D continuous space. Agents may take 4 actions as in the maze problem (left, right, up, and down), but as opposed to the maze problem, actions in puddle world are stochastic. For these experiments the world is 100 by 100, and an agent action moves a unit distance $\pm$ noise taken uniformly from [-0.1,0.1]. The agent experiences a reward of -1 for each timestep until the episode is complete, and, if within either puddle, $-400 * d$, where d is the closest distance to a puddle's edge. The fitness is given by the aggregation of the rewards received during 40 episodes.

5.1.4 Acrobat

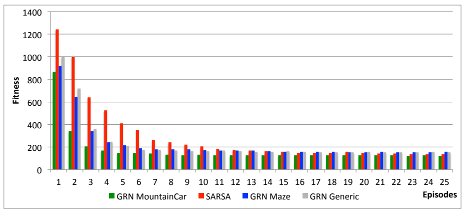
The acrobat problem has a long-standing history in robotics and control, some references to which may be found in [32]. The agent controls 2 segments with 2 joints, where the first joint represents attachment to the bar, and the second joint represents the waist of the acrobat. The agent can only control the second joint, and the corresponding actions are: apply a fixed amount of positive torque, apply a fixed amount of negative torque, or apply no torque. The agent is initialized in a downward vertical position and must reach an upward vertical position. The agent receives a reward of -1 for each timestep until the episode is complete. This prob-



(a) Convergence curve



(b) Best GRN inputs and outputs



(c) GRNM vs SARSA

Figure 6: Mountain car: (a) Convergence curve of the genetic algorithm (lower is better); (b) Regulation of the learning parameters of the best GRN obtained; (c) 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 algorithm (red). Results are averaged on 100 independent runs. Lower is better.

lem is schemetized by figure 5. The fitness is given by the aggregation of the rewards received during 30 episodes.

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 the 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 average 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 2.

	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
Acrobate	0.05	0.928571	0.785714

Table 2: Fixed learning parameters for the SARSA algorithm obtained by parameter sampling.

5.2.1 Mountain car

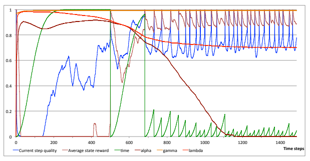
Firstly, the GRN is trained on Mountain car. Figure 6(a) shows the convergence curve of the genetic algorithm on this particular problem. We can observe that the best GRN is found very quickly (lower green curve) before being slowly optimized to regulate the learning parameters more efficiently. Figure 6(b) plots the behavior of the best GRN obtained after 150 iterations. This figure plots the input proteins' 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, memorization 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 random seeds. The point is to evaluate the capacity of both approaches to handle the noise generated by stochasticity. Figure 6(c) 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 the SARSA algorithm with fixed parameters (in red) on every episodes. More specifically, the first episodes (from 1 to 3) show that GRNM learns faster than the SARSA algorithm and later episodes (from 3 to 25) show that neuromodulation solves the problem faster than the SARSA algorithm. When comparing detailed results on table 3, the GRN-based neuromodulation gives better results on the mountain car problem when averaged on 100 independent runs.

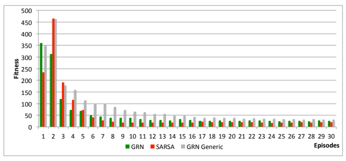
5.2.2 Maze

The same procedure has been used to evolve a gene regulatory network to regulate SARSA algorithm's learning parameters on the maze problems. Figure 7(a) 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 learnt during the first scenarios. γ is kept very high all along the simulation.

When compared to the parameter-fixed SARSA algorithm (figure 7(b)), we can observe that GRNM is learning faster than the SARSA algorithm (episodes 2 to 5) due to its high learning rate at the begining of the simulation. However, the avantage turns to the fixed-parameter SARSA algorithm in the remaining episodes: this latter is doing slightly better



(a) Best GRN's behavior



(b) GRNM vs SARSA

Figure 7: Maze: (a) Regulation of the learning parameters of the besst GRN obtained; (b) Comparison of the fitnesses per episode (abscissa) obtained by our neuromodulation with a GRN trained on maze (green) and with parameter-fixed SARSA algorithm (red). Results are averaged on 100 independent runs. Lower is better.

	Mountain car				Maze			Puddle world				Acrobat			
	GRN	SARSA	GRN$_m$	GRN$_g$	GRN$_m$	SARSA	GRN$_g$	GRN	SARSA	GRN$_m$	GRN$_g$	GRN	SARSA	GRN$_m$	GRN$_g$
avg	**170.5**	294.2	*225.3*	228.3	*56.8*	**53.2**	83.5	**661.9**	*639.0*	533.7	569.5	**420.1**	231.86	320.0	*330.0*
stdev	*16.7*	**4.1**	17.2	41.6	*6.1*	**4.3**	8.4	**105.2**	234.7	177.6	*175.2*	**14.7**	127.9	*37.2*	57.9
best	**152.3**	286.0	191.0	*183.9*	*44.0*	**43.7**	68.1	728.7	**907.1**	728.7	*757.1*	*453.2*	**555.0**	376.7	404.2
worst	**227.4**	308.1	*269.6*	573.2	*73.2*	**73.1**	127.2	**281.1**	-49.5	164.6	*183.8*	**382.2**	93.5	*156.8*	149.3

Table 3: Detailed results of the 100 runs done on the four problems with the best GRN specifically trained on the problem, with the SARSA algorihtm without neuromodulation, with the GRN trained on the maze (noted GRN$_m$ in the table), and a generic GRN trained both on the maze and the mountain car problems (noted GRN$_g$). Notice that, for the first two problems (mountain car and maze), lower is better and for the two others (puddle world and acrobat), higher is better. Bold values are best per problem and per row and italic ones are second best.

than neuromodulation from episode 6 to the end. When refering to the detailed results given by table 3, neuromodulation and SARSA algorithm give comparable results.

5.2.3 Puddle world

Figure 8(a) presents the behavior of the best GRN obtained when neuromodulation is optimized on the puddle world problem. Once again, the GRN choses to use higher learning parameters α and λ at the begining of the simulation in order to explore the environment and memorize the rules faster. When done, the GRN increases the discounting factor and reduce the exploration and memorization values in order to exploit more. This gives good results in comparison to the SARSA algorithm with no neuromodulation: as depicted on figure 8(b), neuromodulation learns faster the rules to solve the problem and converge to an equivalent solution than the SARSA algorithm. When comparing the results in detail (table 3), neuromodulation obtains equivalent results when averaged on 100 independent runs but the standard deviation and worst result are better: the GRNM is capable to modulate the learning rates when harder scenarios are met.

5.2.4 Acrobat

Figure 9(a) shows the regulation obtained with the best GRN evolved on the acrobat problem. The GRN finds learning parameters very close to the one obtained with parameter sampling for the parameter-fixed SARSA algorithm. However, in addition to finding these parameters, the GRN reduces these values all along the episodes, in particular α which decreases down to zero. As in the puddle world problem, the aim is to exploit more the results when an appropriate behavior is found by the SARSA algorithm. When compared to the parameter-fixed SARSA algorithm on this problem (figure 9(b)), GRN-based neuromodulation both learns faster and finishes with a better behavior than the parameter-fixed SARSA algorithm. This is confirmed by table 3 in which the reward averaged on 100 independent runs is largely over with neuromodulation than with the parameter-fixed SARSA algorithm.

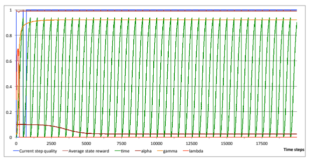
5.3 Generalization of the regulation

5.3.1 Using the GRN trained on the maze

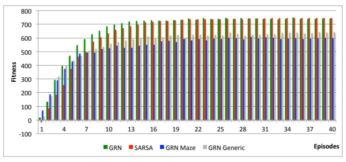
One of the main property of the gene regulatory network is its capacity to generalize its behavior to unknown situations [24]. In this neuromodulation problem, we noticed that the best GRN trained on the maze problem and used on other problems good provides results in comparison to the SARSA algorithm with no neuromodulation. As presented in table 3, GRNM trained on maze (noted GRN$_m$) obtains better results than the parameter-fixed SARSA algorithm on all problems we have tested. In our opinion, the regulation dynamics are very generic: when the rewards start to increase, lowering the learning rate to enter an exploitation phase seams appropriate to most problems.

However, this generic regulatory network does not beat GRNs specificially trained on the problems. This can be explained by the fact that optimizing the GRN specifically on a problem allows the system to find particular artifacts and thus improves the quality of the parameter regulation as well as the best initial values.

Figure 10 shows the behavior of the GRN trained on the maze and used on other problems. The behavior are slightly

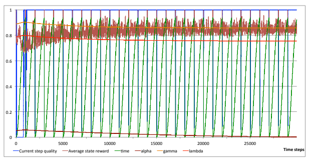


(a) Best GRN's behavior

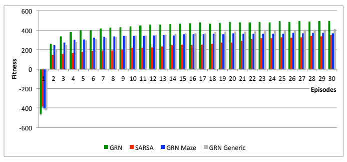


(b) GRNM vs SARSA

Figure 8: Puddle world: (b) Comparison of the fitnesses per episode (abscissa) obtained by our neuromodulation with a GRN trained on this problem (green), a GRN trained on maze (blue), a GRN trained on maze and mountain car (gray) and with the parameter-fixed SARSA algorithm (red). Results are averaged on 100 independent runs. Higher is better.

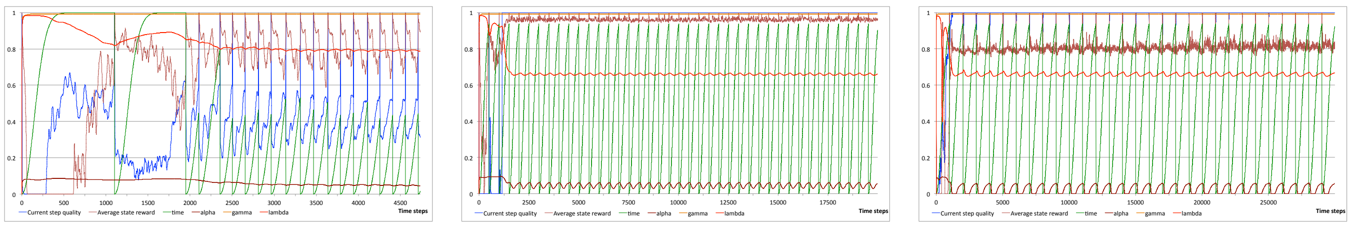


(a) Best GRN's behavior



(b) GRNM vs SARSA

Figure 9: Acrobat: (a) Regulation of the learning parameters of the besst GRN obtained; (b)Comparison of the fitnesses per episode (abscissa) obtained by our neuromodulation with a GRN trained on this problem (green) and a GRN trained on maze (blue) and with the parameter-fixed SARSA algorithm (red). Results are averaged on 100 independent runs. Higher is better.



(a) Mountain Car (b) Puddle world (c) Acrobat

Figure 10: Behavior of the GRN trained on the maze when used to regulate learning parameters on mountain car, puddle world and acrobat.

different on each problem: the GRN is able to adapt to problem specificities. In particular, when the episodes are taking to much time, this GRN increases λ first and then α. It is also interesting to notice that the stabilized values of the learning parameters different from a problem to another. For example, λ is kept around 0.8 on the mountain car problem, around 0.65 for the puddle world and the acrobat problems whereas it is equal to 0.7 on the maze problem. It shows the capacity of the GRN to approximate the best learning parameters to a problem without further learning.

When compared to the SARSA algorithm and neuromodulation with GRN trained specifically on one problem (table 3), the GRN evolved on the maze is doing better than the SARSA algorithm on mountain car and acrobat problems but have poorer results on puddle world. The worst reward obtained is better than the parameter-fixed SARSA algorithm in all problem, even on puddle world where the average reward is lower. Once again, this shows the capacity of neuromodulation to balance the learning parameters in harder scenarios.

5.3.2 *Evolving a generic GRN*

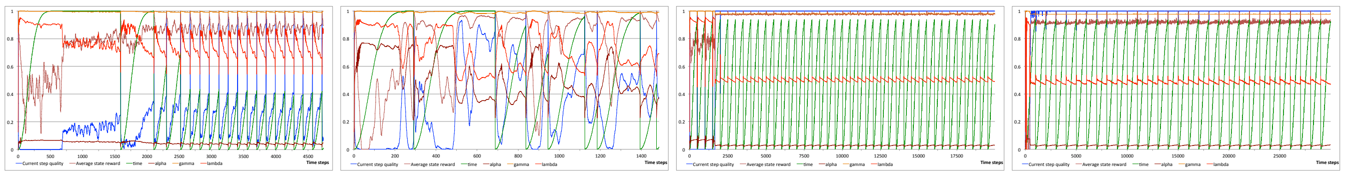
Based on this observation, we have trained a new GRN on two problems instead of only one. The aim is to obtain a generic GRN that have seen two problems instead of one. Therefore, we have chosen to train the GRN on the maze and the mountain car problems. Figure 11 shows the behavior of the best GRN obtained. Comparable regulation behaviors can be observed on the different problems: at the begining of each episode, λ is set up to a high value and then reduces until the end of the episode. However, the parameter values are regulated to different levels according to the problem. The GRN might use the current step quality and the average state reward inputs to regulate these levels. These inputs provide the GRN an estimation of the agent progression in the episode, in other words how well or bad the agent is currently doing, and, consequently, drive the GRN to set the SARSA alogirthm up to explore more to find better behavior rules.

To compare the generic GRN to specific GRNs and to the GRN trained on the maze problem, we have compared this GRN following the same procedure as in past problems: the generic GRN is evaluated on 100 independent runs with different random seeds. Its result are reported in table 3 within the GRN$_g$ column. Interestingly, the generic GRN beats GRN$_m$ on the puddle world and the acrobat problems but doesn't beat the GRNs specifically trained on these problems. The generic GRN and the GRN trained on maze are doing comparably on the mountain car problem, but the GRN trained on the maze is doing better on the maze. This was expected since the GRN trained on the maze is specifically trained on this problem whereas the generic GRN has been trained on two problems and thus its training is also impacted by the mountain car results. Just as the GRN trained on the maze, the generic GRN beats the parameter-fixed SARSA algorithm on the mountain car and the acrobat problems. Both the maze and the puddle world problems are in favour to the standard SARSA algorithm. However, the generic GRN does better in the worst case scenarios, as well as all other GRNs: again, this generic GRN seems to be able to extricate the agent to dead ends.

6. CONCLUSION

In this paper, we proposed to evolve a gene regulatory network to control reinforcement learning paramaters on four state-of-the-art problems. We have evolve GRN specificially on each task, which produced better or equivalent results than the parameter-fixed SARSA algorithm on average. In all cases, the worst case scenario is always better handle by GRN-based neuromodulation because of its capacity to modify the learning parameters on-the-fly to escape dead ends.

In addition, the GRN evolved on the maze as well as a generic GRN have been tested on the problems and provides



(a) Mountain Car (b) Maze (c) Puddle world (d) Acrobat

Figure 11: Behavior of the GRN trained on the maze and mountain car when used to regulate learning parameters on other problems.

encouraging results with no further learning. Once again, the worst case scenario is better handled by neuromodulation with generic GRNs than without neuromodulation. However, generic GRNs are not as good as GRN specifically evolved on each task. Other inputs might be usefull to the GRN to estimate the quality of the current neuromodulation and the quality of the agent behavior. Finding a generic GRN able to perfectly modulate the learning parameter would make possible not to evolve the GRN anymore and use it as is on all possible problems. That would make the parameter sampling phase of reinforcement learning out of date.

As was noted in [27], neuromodulation is neither problem nor RL-algorithm specific, thus future work may investigate the application of GRN-neuromodulation on alternative RL algorithms. We have shown that a simple feedback controller can optimize community features of agent-based swarm simulations, the use of GRN-based neuromodulation is likely to further optimize these environmental controllers [11]. Having a problem-independent and algorithm-independent neuromodulation architecture would save a large amount of set up for both the problem and the algorithm.

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

7. REFERENCES

[1] J. Boyan and A. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

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

[6] A. Destexhe and E. Marder. Plasticity in single neuron and circuit computations. *Nature*, 431(7010):789–795, 2004.

[7] 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.

[8] K. Doya. Metalearning and neuromodulation. *Neural Networks*, 15(4):495–506, 2002.

[9] K. Doya. Modulators of decision making. *Nature neuroscience*, 11(4):410–416, 2008.

[10] J. Fellous and C. Linster. Computational models of neuromodulation. *Neural computation*, 10(4):771–805, 1998.

[11] J. Gold, A. Wang, and K. Harrington. Feedback Control of Evolving Swarms. In *Proceedings of Artificial Life XIV*, pages 884–891, 2014.

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

[13] H. Handa. Evolutionary Computation on Multitask Reinforcement Learning Problems. In *Networking, Sensing and Control, 2007 IEEE International Conference on*, pages 685–688, 2007.

[14] K. I. Harrington, E. Awa, S. Cussat-Blanc, and J. Pollack. Robot Coverage Control by Evolved Neuromodulation. In *IJCNN 2013*, pages 543–550, 2013.

[15] 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.

[16] 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.

[17] P. Katz. Intrinsic and extrinsic neuromodulation of motor circuits. *Current opinion in neurobiology*, 5(6):799–808, 1995.

[18] H. Li, X. Liao, and L. Carin. Multi-task reinforcement learning in partially observable stochastic environments. *The Journal of Machine Learning Research*, 10:1131–1186, 2009.

[19] E. Marder. Neuromodulation of neuronal circuits: back to the future. *Neuron*, 76(1):1–11, 2012.

[20] E. Marder and V. Thirumalai. Cellular, synaptic and network effects of neuromodulation. *Neural Networks*, 15(4):479–493, 2002.

[21] P. Montague, P. Dayan, and T. Sejnowski. A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *The Journal of Neuroscience*, 16(5):1936–1947, 1996.

[22] A. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate realvalued state-spaces. In *Proceedings of the Eighth International Conference on Machine Learning*, pages 333–337, 1991.

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

[24] S. Sanchez and S. Cussat-Blanc. Gene regulated car driving: using a gene regulatory network to drive a virtual car. *Genetic Programming and Evolvable Machines*, 15(4):477–511, 2014.

[25] 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.

[26] 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.

[27] N. Schweighofer and K. Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

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

[29] R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on Machine learning*, pages 216–224, 1990.

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

[31] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

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

[33] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[34] Y. Zhurov and V. Brezina. Variability of motor neuron spike timing maintains and shapes contractions of the accessory radula closer muscle of Aplysia. *The Journal of neuroscience*, 26(26):7056–7070, 2006.

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