Genetically-regulated Neuromodulation Facilitates Multi-Task Reinforcement Learning

Anonymous author 1
Anonymous address 1
Anonymous address 1
Anonymous address 1
Anonymous email 1

Anonymous author 2
Anonymous address 2
Anonymous address 2
Anonymous address 2
Anonymous email 2

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 serves as neuromodulator of 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 genetically-regulated neuromodulation (GRNM) performs comparably or better than SARSA with fixed parameters. We then extend the GRNM SARSA algorithm to multi-task problem generalization, and show that GRNs optimized 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 [15]. We apply GRNM to additional problem classes, and show that GRNs evolved on agents learning a subset of problems can facilitate learning on previously unencountered problems.

This paper is organized as follows. Firstly, we briefly summarize reinforcement learning and in more specifically the SARSA algorithm used in this study. Then, an introduction to gene regulatory network is given with the details on the computational model we use. Section 4 shows how we use the regulatory network for neuromodulation. Section 5 presents the four problems we use neuromodulation on and the results of GRNs trained specifically on each problem and the capacity of the GRNs to generalize their behavior to unknown problems.

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

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 [24].

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 encoded in a table which associates action preferences with states. The basic TD(γ) algorithm updates one state-action association at a timestep, which restricts sequence learning. Eligibility traces are used to associate reward with sequences of actions by reinforcing a weighted history of recent actions. In this study an online version of TD-RL, SARSA (short for, state-action-reward-state-action), is used. A review 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. 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

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

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, reward propagation can be partially alleviated by 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

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

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 [6]. These protein sensors represent receptor molecules localized to the cellular membrane which transduce external activity into excitatory and/or inhibitory regulatory signals. Cells can use external signals collected from protein sensors localized on the membrane to activate or inhibit the transcription of the genes. Computational models have been designed and used in many developmental models of the literature [16, 9, 4] and to control virtual and real robots [34, 22, 17, 5].

Our model is based on Banzhaf's biologically-inspired model [1], a gene regulatory network is defined as a set of abstract 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 greater the enhancement/inhibition of 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.
- 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 are calculated as follows. First, the affinity of a protein a with another protein b is given by the enhancing factor u^+_{ab} and the inhibiting u^-_{ab}:

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

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:

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

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 [8, 19, 20]. We have already noted that the temporal difference learning algorithm

for error prediction has been observed in neural substrates [25]. An extensive review of computational models of neuromodulation can be found in [12], and some recent models are reviewed in [19]. In this study we extend work on the evolution of neuromodulation [27, 15], focusing on the relationship between evolved neuromodulatory GRNs and reinforcement learning [15].

4.1 Regulating parameters

With the intention of focusing on the optimization of genetically-regulated neuromodulation, the physical mechanisms underlying neuromodulation are used as inspiration and the neuromodulation of reinforcement learning was optimized. Neuromodulation has been considered in the context of RL [10, 26, 11], 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: the learning rate α, the discount factor γ and the memory depth λ.

To do so, the GRN uses three inputs that describe the current performance of the agent in the environment. They have been chosen to be problem-independent as one of our goals is to define a problem-independent neuromodulation architecture to minimize problem-specific adjustments. The first input describes the duration from the beginning of current episode: the concentration of this first input protein increases with the time spent in each learning episode. The concentration $C_{I_1}(t)$ 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 the eligibility trace and Q is the Q-function, both described in Section 2. The aim of this input is to capture the quality of the current state history relative to previous experiences. 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 eligibility trace. This input encodes the frequency that the agent has reentered the same state.

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

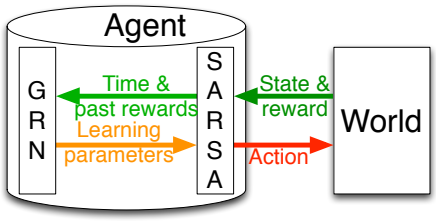


Figure 1: At every time step, SARSA updates the GRN inputs. The GRN returns updated learning parameters that will be used by the SARSA algorithm.

- the output protein O_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 a gene regulatory network can be used for successful neuromodulation, the GRN must be optimized. In this paper, we use an evolutionary algorithm inspired by the NEAT algorithm [28], but adapted for GRNs called GRNEAT [3]. GRNEAT incorporates three key features: (1) *initialization* of the algorithm - as opposed to initializing with network sizes randomly sampled from the complete distribution, only small networks are used in the initial population so as to allow for a more progressive complexification, (2) *speciation* protects newly appeared solutions by giving them some time to optimize their structures before competing with the whole population, and (3) *alignment crossover* uses of a distance metric between proteins to maintain subnetwork architecture during the crossover operation. This modified algorithm has been shown to converge faster and to better solutions than standard genetic algorithms. More details on GRNEAT can be found in [3].

During optimization, each GRN is evaluated independently on a given problem with 25 reruns in order to reduce the effect of stochasticity on fitness. As fitness is problem dependent, more detail is provided in the corresponding experiment sections. To avoid any memory bias, SARSA is completely reset before each evaluation. The parameters used for GRNEAT in this work are given by Table 1, and are consistent for all problems.

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

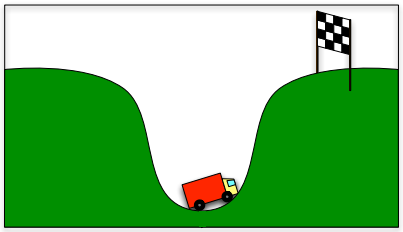


Figure 2: Mountain Car

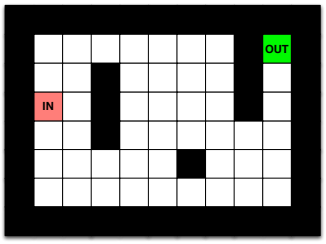


Figure 3: Maze

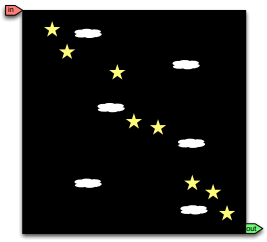


Figure 4: Puddle World

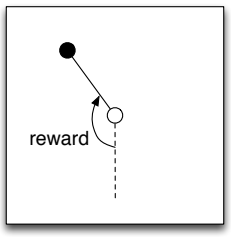


Figure 5: Acrobat

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: GRNEAT parameters used for evolution.

5. PROBLEMS

We have studied GRNM on four classic problems: Mountain Car, Maze, Puddle World and Acrobat. These problems are implemented within the RLPark software package [7]. In all cases agents do not have prior information about the specific task. Before detailing results obtained with GRNM, this section presents each problem.

5.1 Mountain Car

The Mountain Car problem was introduced by Moore to promote the broad applicability of dynamic programming [21]. In the mountain car problem, depicted in 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. For evolution of GRNs, the fitness of an evaluation is given by the number of step necessary to complete a series of 25 episodes.

5.2 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 researchers, of particular interest is Hanada's study which applies evolutionary programming to a multi-task version of Sutton's maze [14]. The fitness of a training run is given by the number of steps necessary to complete a series of 30 episodes.

5.3 Puddle World

The puddle world problem was introduced by Boyan and Moore [2], but was later presented in greater detail by Sutton [31]. As shown 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 agents 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 average of rewards received during 40 episodes.

5.4 Acrobat

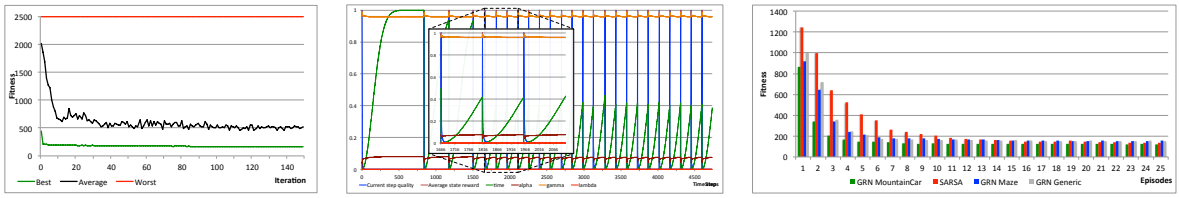
The acrobat problem has a long-standing history in robotics and control, some references to which may be found in [32]. The agent controls 2 segments with 2 joints, where the first joint represents attachment to the bar, and the second joint represents the waist of the acrobat. The agent can only control 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 problem is schemetized by figure 5. The fitness is given by the average of rewards received during 30 episodes.

5.5 Training GRN on one specific problem

In this first experiment, we have trained our GRNM model independently on each problem. To evaluate the gain provided by neuromodulation, we first determine the best fixed parameters to use with SARSA on each problem. We use combinatorial 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

	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.



(a) Convergence curve (b) Best GRN inputs and outputs (c) GRNM vs SARSA

Figure 6: Mountain car: (a) Convergence curve of GRNEAT (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 fixed-parameter SARSA algorithm (red). Results are averaged on 100 independent runs. Lower is better.

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.

5.6 Mountain Car

Figure 6(a) shows the evolutionary convergence curve on the Mountain Car problem. 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 almost constant parameter values over each episode time, except at the beginning where α and γ are increased. This reinforces learned values for the initial part of the agent's trajectory. In this experiment, it should also to be noticed that λ is kept at zero all over time, suggesting that on this simple problem sequence learning may 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. Figure 6(c) shows the results obtained for each episode averaged on the 100 runs. We can observe that GRNM 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 leads to agents that solve the problem faster than fixed-parameter SARSA. 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.

5.7 Maze

The same procedure has been used to evolve a gene regulatory network to regulate SARSA algorithm's learning parameters on the Maze problem. Figure 7(a) presents the behavior of the best GRN obtained after evolution. The regulatory dynamics differ significantly from those obtained on the Mountain Car problem. This GRN starts with very high values for all learning parameters leading to a strong memorization of initial experiences. Then, upon discovering successful solutions, the learning rate α and the memory depth λ decrease to exploit the learned behaviors. γ is kept very high all along the simulation.

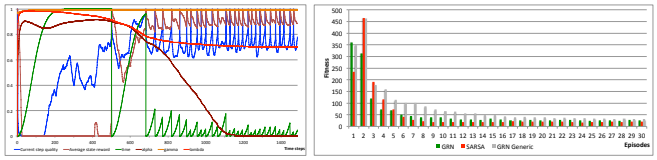
When compared to the fixed-parameter SARSA algorithm (figure 7(b)), we can observe that GRNM is learning faster than the SARSA algorithm (episodes 2 to 5). However, the advantage turns to the fixed-parameter SARSA algorithm in the remaining episodes: the latter is doing slightly better than GRNM from episode 6 to the end. When referring to the detailed results given by table 3, GRNM and fixed-parameter SARSA give comparable results.

5.8 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 chooses to use higher learning parameters α and λ at the beginning of the simulation to learn from initial experiences faster. After this initial phase, the GRN increases the discounting factor and reduces the amount of memorization in order to better exploit learned experiences. This gives good results in comparison to fixed-parameter SARSA: as depicted in figure 8(b), GRNM learns the value of states and actions faster and converges to an equivalent solution to fixed-parameter SARSA. 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 of modulating the learning parameters when harder scenarios are encountered.

5.9 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 fixed-parameter SARSA algo-



(a) Best GRN's behavior (b) GRNM vs SARSA

Figure 7: Maze: (a) GRN dynamics of genetically-regulated neuromodulation with the best GRN obtained; (b) Comparison of the fitnesses per episode (abscissa) obtained by GRNM with a GRN trained on Maze (green) and with fixed-parameter SARSA (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 from the four problems with the best GRN trained on the specific problem, the SARSA algorithm with fixed parameters, the GRN trained on Maze (noted GRN_m in the table), and a generic GRN trained both on the Maze and 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, per row. Italics indicate second best.

rithm. 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 fixed-parameter SARSA algorithm on this problem (figure 9(b)), GRN-based neuromodulation both learns faster and finishes with a better behavior than the fixed-parameter 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 fixed-parameter SARSA algorithm.

6. USING THE GRN TRAINED ON MAZE

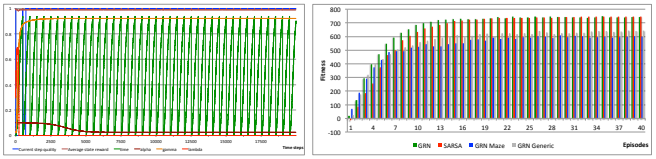
A powerful property of artificial gene regulatory networks is their capacity to generalize to unknown situations [23]. In the case of genetically-regulated neuromodulation, we considered the best GRN trained on the maze problem and tested it on other problems. We noticed that it performs well in comparison to the fixed-parameter SARSA algorithm: as presented in Table 3, GRNM trained on maze (noted GRN_m) obtains better results than the fixed-parameter SARSA algorithm on all other problems we have tested. In our opinion, the regulatory dynamics of this GRN are very generic: when the rewards start to increase, the learning rate is lowered to preserve previously learned experiences. This regulatory dynamic appears to be appropriate for most problems. However, this generic regulatory network does not beat GRNs trained on specific problems. This can be explained by the fact that optimizing the GRN on an individual problem allows the system to exploit problem-specific features, improving the quality of the parameter regulation.

Figure 10 shows the behavior of the GRN trained on the Maze problem and used on other problems. The behaviors are slightly 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 differ 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. These values have to be compared to the best values obtained for fixed parameters in table 2: it shows the capacity of the GRN to approximate the best learning parameters to a problem without further learning.

When comparing to the fixed-parameter SARSA algorithm and GRNM optimized specifically on one problem (Table 3), the GRN evolved on Maze outperforms fixed-parameter SARSA on Mountain Car and Acrobat but has poorer results on Puddle World. However, the worst-case is better than fixed-parameter SARSA on all problems, even Puddle World where the average reward is lower. Once again, this shows the capacity of GRNM to compensate learning parameters in harder scenarios.

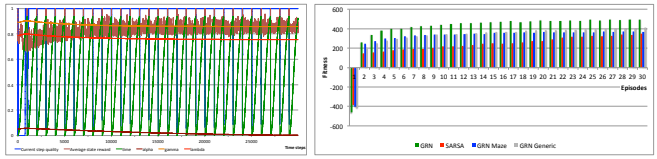
7. EVOLVING A GENERIC GRN

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 has been optimized for multiple problems. 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 regulatory behavior can be observed on the different problems: at the beginning



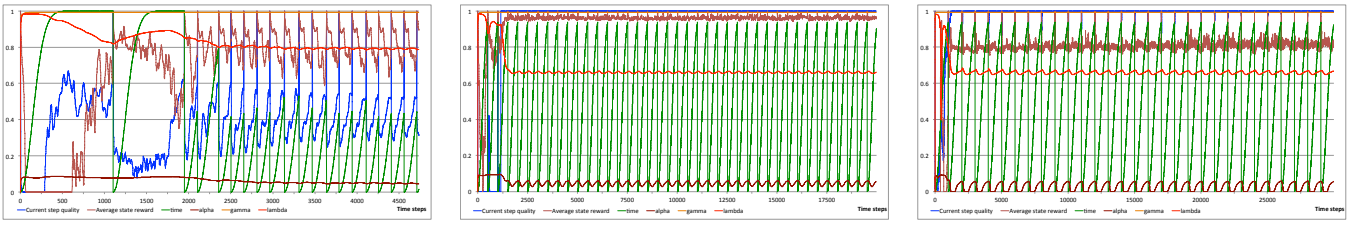
(a) Best GRN's behavior (b) GRNM vs SARSA

Figure 8: Puddle World: (b) Comparison of the fitnesses per episode (abscissa) obtained by genetically-regulated 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 fixed-parameter 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 best GRN obtained; (b) Comparison of the fitnesses per episode (abscissa) obtained by genetically-regulated neuromodulation with a GRN trained on this problem (green) and a GRN trained on Maze (blue) and with the fixed-parameter 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 problem when used to regulate learning parameters on Mountain Car, Puddle World and Acrobat.

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 for each problem. The GRN might use the current step quality and the average state reward inputs to regulate these levels. These inputs provide the GRN with an estimation of the agent's progression in an episode. As the agent progresses further into an episode, the number of sequences of actions that could have led to any given state are almost always increasing, thus when the agent is learning values of states and actions later in an episode it appears to be useful to discount older actions and favor shorter and more recent sequences of actions.

To compare the generic GRN to specific GRNs and to GRNs trained on Maze, we have compared this GRN using the same procedure as with prior experiments: the generic GRN is evaluated on 100 independent runs with different random seeds. Corresponding results are shown in Table 3 as the GRN_g column. Interestingly, the generic GRN beats GRN_m on the Puddle World and Acrobat problems but does not beat the GRNs specifically trained on these problems. The generic GRN and the GRN trained on Maze perform comparably on the Mountain Car problem, but the GRN trained on Maze is doing better on the Maze itself. This was expected since the GRN trained on Maze is specifically optimized for this problem whereas the generic GRN has been trained on two problems and thus its optimization was also affected by training on Mountain Car. Just as the GRN trained on Maze, the generic GRN beats the fixed-parameter SARSA algorithm on the Mountain Car and Acrobat problems. Both the Maze and Puddle World problems favor fixed-parameter SARSA. However, all GRNs do better in the worst-case scenarios due to the ability of GRNs to react to situations dynamically and avoid incorrect credit assignment in sequences of actions (for example, by varying λ).

8. CONCLUSION

In this paper, we employ gene regulatory networks to serve as neuromodulators of learning parameters for a reinforcement learning algorithm on four benchmark problems. We have evolved GRNs specifically on each task, which generally produced better or equivalent results to the standard fixed-parameter SARSA algorithm. In all cases, the worst-case scenario is always handled better by GRNM due to its ability to perform on-the-fly parameter adaptations which reduces complications of learning with fixed parameters, such as incorrect credit assignment in sequences of actions.

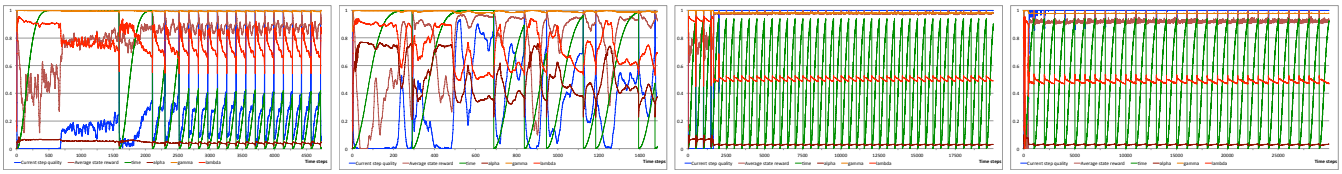
Additionally, the GRN evolved on the Maze problem as well as a generic GRN have been tested on the problems and offers encouraging results for multi-task learning in multiple problem domains. Once again, the worst-case scenario is consistently handled better by genetically-regulated neuromodulation with generic GRNs than with fixed-parameter algorithms. However, generic GRNs are not as good as GRN specifically optimized for each task. Other inputs might be useful to the GRN for estimating the quality of the current neuromodulation and the quality of the agent behavior. The pursuit a generic GRN capable of generalizing the regulation of learning parameters across multiple tasks and problem domains would not only eliminate the need to search for additional GRNs, but also the parameter sampling phase commonly required for reinforcement learning applications.

As was noted in [26], neuromodulation is neither problem nor RL-algorithm specific, thus future work may investigate the application of genetically-regulated neuromodulation to 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 [13].

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.



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

Figure 11: Behavior of the generic GRN when used to regulate learning parameters on all problems.

[2] 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.

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

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

[5] 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.

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

[7] T. Degris. RLPark, 2014.

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

[9] 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.

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

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

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

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

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

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

[16] 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.

[17] 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.

[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] 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.

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

[23] 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.

[24] 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.

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

[27] A. Soltoggio, J. Bullinaria, and C. Mattiussi. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Artificial Life*, volume 11, pages 569–576, 2008.

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