Cognitive Policy Learner: Biasing Winning or Losing Strategies

Dominik Dahlem
SENSEable City Laboratory
Massachusetts Institute of Technology
Cambridge, USA
dahlem@mit.edu

Jim Dowling
Computer Systems Laboratory
Swedish Institute of Computer Science
Stockholm, Sweden
jim.dowling@sics.se

William Harrison
School of Computer Science and Statistics
Trinity College Dublin
Dublin, Ireland
bill.harrison@cs.tcd.ie

ABSTRACT

In continuous learning settings stochastic stable policies are often necessary to ensure that agents continuously adapt to dynamic environments. The choice of the decentralised learning system and the employed policy plays an important role in the optimisation task. For example, a policy that exhibits fluctuations may also introduce non-linear effects which other agents in the environment may not be able to cope with and even amplify these effects. In dynamic and unpredictable multiagent environments these oscillations may introduce instabilities. In this paper, we take inspiration from the limbic system to introduce an extension to the weighted policy learner, where agents evaluate rewards as either positive or negative feedback, depending on how they deviate from average expected rewards. Agents have positive and negative biases, where a bias either magnifies or depresses a positive or negative feedback signal. To contain the non-linear effects of biased rewards, we incorporate a decaying memory of past positive and negative feedback signals to provide a smoother gradient update on the probability simplex, spreading out the effect of the feedback signal over time. By splitting the feedback signal, more leverage on the win or learn fast (WoLF) principle is possible. The cognitive policy learner is evaluated using a small queueing network and compared with the fair action and weighted policy learner. Emphasis is placed on analysing the dynamics of the learning algorithms with respect to the stability of the queueing network and the overall queueing performance.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Experimentation

Keywords

Multiagent Reinforcement Learning, Stochastic Policies

1. INTRODUCTION

Multiagent Reinforcement Learning (MARL) techniques have been successfully applied to a number of domains, ranging from general sum games [12, 14] to application areas such as packet routing [18], robot control [15], and resource allocation [20, 21]. A theoretical framework for sequential decision and multiagent problem settings is provided by the formalisms of the decentralised Markov Decision Process (DEC-MDP) [5, 10]. Planning-based solution methods have been devised to solve these models offline. However, their complexity increases dramatically if no reward or transition model is available or the number of agents goes beyond small scenarios. In contrast, online approximate solutions have been shown to be useful in solving DEC-MDP problems [25]. Their central idea is that models of learning and memory are continuously updated and incorporated into a trial-and-error interaction within the agent's local context. Agents learn using only local information, but they should support near optimal global decision making. In unison, all agents contribute to the global goal of optimising some system objective. Simultaneous and independent interactions, however, pose a challenge to multiagent systems, because they are non-deterministic, may have non-linear effects, and may lead to slow convergence characteristics or even diverge. Some research directions tackle these difficulties by modelling the other agents in the environment [9] or by providing a mechanism to communicate feedback of parallel optimisation processes underway in the environment [10, 19].

Additionally, the modelling assumptions of DEC-MDP often need to be extended to capture the application specific constraints. For example, for packet routing or task allocation networks, the service stations or nodes have limited capacity to service requests and limited resources to store waiting tasks. Networked systems exhibit a level of complexity that is very challenging to deal with. In the absence of direct communication links between nodes sharing a common resource, coordination is difficult to achieve to optimally utilise this common resource. For example, consider the queueing network presented in Figure 1 which is used for all evaluation scenarios. Both agents (nodes 6 and 7) share a common resource (node 4) and may observe that the common resource offers enough capacity to service their individual requests. As such, both agents may decide to utilise this resource at the same time causing potential congestion. Under certain conditions, this may lead to fluctuating performance that may cascade through the network. Consequently, autonomous agents need to mitigate the occurrence of cascades (non-linear effects) and adapt quickly to changing conditions.

In this paper, we introduce two new features to the weighted policy learner: an inherent bias that magnifies or depresses rewards depending on how far they diverge from the average expected reward for different actions in that state, and, secondly, a transient memory of recent rewards for actions that smooth out the current

Cite as: Cognitive Policy Learner: Biasing Winning or Losing Strategies, Dominik Dahlem, Jim Dowling, William Harrison, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 601-608.

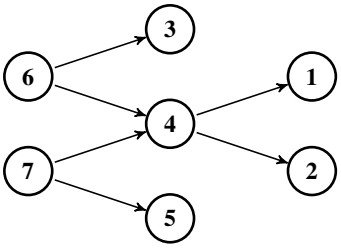

Figure 1: Queueing Network

reward. Both of these features dampen cascading effects of large changes in rewards at key nodes, preventing system hysteresis, but still enabling agents to converge to a stable stochastic policy. Moreover, the cognitive policy learner offers greater control over the extent of the win or learn fast strategy by interpreting the positive and negative feedback signals separately.

We evaluate the cognitive policy learner using a small queueing network given in Figure 1 and compare it with the fair action and weighted policy learner. We investigate whether modulating the strength of feedback signals can have a stabilising impact on the learning system. Emphasis is given to analysing the dynamics of the learning algorithms with respect to the stability of the queueing network and the overall queueing performance. Our results show improved queueing performance compared to the fair action learner, and similar queueing performance to the weighted policy learner. However, our results suggest that our cognitive policy learner yields a more stable multiagent learning system compared to the weighted policy learner, as it has a significantly lower total mean-squared training error for the $SARSA(0)$ steepest-descent gradient update.

2. BACKGROUND

This section provides the background to the collaborative multiagent reinforcement learning environment for queueing networks. We assume that the queueing network is given as a directed acyclic graph, which implies that all interactions between the agents are directed and do not form any cycles. Similar in concept to the collective intelligence framework of Wolpert et al. [23], a subworld, ψ_i, constitutes a number of queueing agents that together complete a task for agent i. Each agent can be viewed as though it is striving to maximise its own reward function with the consequence of improving the performance of the subworld as a whole. The engineering discipline is based on division of labour, where the system is sub-divided into smaller parts. The solution of the decentralised optimisation problem is brought about in a bottom-up fashion. More formally, a subworld can be defined as

DEFINITION 1 (SUBWORLD). *A **subworld**, ψ_i, is a subgraph of the queueing network comprising all agents j reachable from agent i.*

- *The queueing network induces nested subworlds. At the leaf nodes of the queueing network subworlds consist of empty sets.*
- *A **path**, p_i, in subworld ψ_i represents a realisation of a local queueing task assignment to agent i.*
- *$\mathcal{W}_i$ is a set of all possible paths in subworld ψ_i.*

With the help of the subworld definition, the multiagent sequential decision problem can be formalised in a DEC-MDP given in Definition 2.

DEFINITION 2. *An n-agent continuous state **DEC-MDP** of a queueing network is defined by a tuple $\mathcal{M} = \langle \mathcal{DAG}, \mathcal{A}, \mathcal{S}, \mathcal{P}, \mathcal{R} \rangle$, where*

- *$\mathcal{DAG}$ is the directed acyclic graph prescribed by the agent's interactions. Each agent is represented as a vertex on the graph and the arcs between the agents represent available actions to the respective agents.*
- *$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the finite set of actions and is given by the possible interactions.*
- *$\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ is finite set of queueing network states, which can be factored into local states $\mathcal{S}_i$ for each agent i, including queueing metrics such as delay, utilisation, or number of events in the queue.*
- *$\mathcal{P}_{w_i} = P\{s_{t+1} = s' \mid s_t = s, \vec{a}_t \exists p_i \in \mathcal{W}_i\}$ is the transition probability of state s' for agent i when the actions $\vec{a}$ comprising path p_i have been taken in state s.*
- *$\mathcal{R}_{w_i} = \mathbb{E}\{r_{t+1} \mid s_t = s, \vec{a}_t \exists p_i \in \mathcal{W}_i, s_{t+1} = s'\}$ is the expected value of the next reward for agent i when actions $\vec{a}$ are taken in state s and transitioning to the next state s'.*

It is important to note that a policy must exist for which the aggregated arrival rates at each node of the queueing network do not yield unstable queues. More specifically, this implies that solving the traffic equations

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}_0 (\mathbf{I} - \mathbf{Q})^{-1}, \tag{1}$$

where $\boldsymbol{\lambda}_0$ is the vector of external Poisson arrival rates for each node in the network and $\mathbf{Q}$ specifies the transition probabilities derived from the policy, requires that the stability criterion, $\frac{\lambda_i}{\mu_i} < 1$, holds for each node. Here, $\boldsymbol{\mu}$ is the vector of exponential service rates, which is considered fixed and represents the nodes' capability to service incoming requests.

Following [25], each agent observes local reward signals, which are given as the negative task processing time. Longer task completion times are less desirable, which makes this reward function a natural choice. This includes all local processing times of the task at each service station (agent) where no communication delay is assumed. Then the value function for a local policy π_i is defined with respect to the average expected reward as:

$$\rho_i(\pi_i) = \lim_{N \to \infty} \frac{1}{N} \mathbb{E}\left[\sum_{t=0}^{N-1} r_i^t \mid \pi_i \right], \tag{2}$$

where r_i^t is the observed reward at time t and it depends on the global states of the queueing network. However, unlike [25], our model cannot be reformulated into an average-reward factored DEC-MDP, because the rewards received by each agent are not independent, that is the global reward is not equal to the sum of the local rewards. Noting that the reward is the response time of the completion of a local task, one can informally see that a reward can only be given when the local task is completed. An external request for a task entering the system at agent i yields an execution path $p_i \in \mathcal{W}_i$ through this agent's subworld ψ_i. Each agent along this path maximises its own reward function forming their respective subworlds. Consequently, since the rewards are computed based on completion times of local tasks a reward relationship of $r_1 < r_2 < \cdots < r_i$ is established, so each r_i is the negative value of the response time of the local task t_i.

This reward structure is said to be global, because it incorporates the rewards of all agents involved in completing a task. More importantly, the reward function has optimal substructure. This means that if we take the negative task completion times as rewards, the credit given to a fulfilled task is apportioned fairly among the agents involved in the completed task. This yields a cooperative multiagent system, as distinct from local reward functions that encourages competitive behaviour among selfish agents. An advantage of this reward structure is that no communication is required to correlate rewards and apportion the reward fairly.

Therefore the agents' reward functions are not mutually independent and offline planning approaches are more difficult to achieve. Moreover, the lack of an explicit reward and transition model increases the complexity of solving such a system and consequently is only feasible for the most simple cases. Online approaches, however, offer a scalable and approximate alternative. Here, we use a standard backpropagation feedforward neural network with one hidden layer on each arc of the task network to estimate the Q-values for each action given a state vector [16]. The temporal difference scheme $SARSA(0)$ can then be expressed as the general gradient-descent update rule for neural network training as

$$\Delta\boldsymbol{\omega}_{t+1} = \alpha[\upsilon_{t+1} - Q(s_t, a_t)]\nabla_{\boldsymbol{\omega}_t} Q(s_t, a_t) + \eta\Delta\boldsymbol{\omega}_t, \quad (3)$$

$$\upsilon_t = r_t + \lambda Q(s_t, a_t), \quad (4)$$

where η is a constant representing the momentum, which determines the effect of past changes to the weight vector, $\boldsymbol{\omega}$, and $\nabla_{\boldsymbol{\omega}_t} Q(s_t, a_t)$ is the vector of partial derivatives of the value function $Q(s_t, a_t)$ with respect to the weight vector $\boldsymbol{\omega}_t$. The action-value estimation is updated every time a task in the queueing network is completed. That means, that all value functions of all arcs in the queueing network that were involved in forwarding a request to the next sub-task will be updated according to $\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \Delta\boldsymbol{\omega}_{t+1}$. The optimal action-value function Q^* is estimated with a parametric function approximator, $Q_{\boldsymbol{\omega}}$, where $\boldsymbol{\omega}$ is the vector of weights as given above. The neural network function approximator is instantiated with one hidden layer and 10 hidden neurons. The agents take only local information into account to train the Q-value function. In all evaluation scenarios of this paper, the delay $\hat{w}_i$ in the local queue forms the input to the neural network. The delay is calculated as the difference between the time of arrival of a task at a node in the queueing network and the time of scheduling the task. The state vector can easily be extended with other queueing metrics, such as current utilisation or the number of task assignments waiting to be scheduled. The queueing discipline is first-in-first-out. This means that as the node is processing a task all tasks arriving at the same time are put into a waiting queue. As the node finishes processing tasks, tasks in the waiting queue are dequeued on a first come, first served basis.

3. RELATED WORK

Multiagent reinforcement learning has seen significant contributions in packet routing [8, 18, 22, 23]. Q-routing was one of the first multiagent approaches applied to routing [8]. Q-routing by itself has its roots in the Bellman-Ford shortest path routing algorithm [4]. The original Q-learning algorithm has routing performance comparable to the Bellman-Ford algorithm under low load. However, since Q-routing uses estimates of the delivery time of a packet, it tends to congest paths if a better performing link has been over-estimated. This problem persists, because Q-routing is a deterministic protocol that always chooses the best performing link to deliver a packet. An attempt to mitigate choosing sub-optimal paths communication between adjacent nodes is established to update estimates before decisions are made. It turns out, however, that high-load scenarios result in fluctuations with routing performance worse off than in the basic Q-learning algorithm. This result can be reconciled with stochastic policies that adjust mixed strategies with continuously adapting agents.

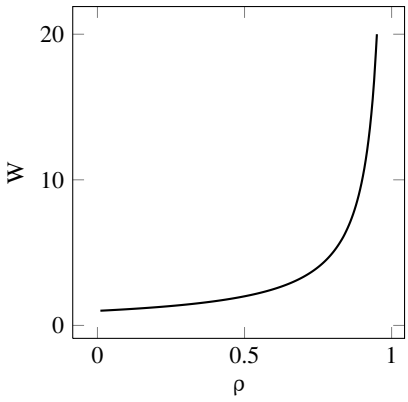

Figure 2: Mean Response Time versus Utilisation

This can be readily understood by studying the sensitivities of queueing metrics with respect to the utilisation of a single queue. Little's theorem states that the long-term average number of events in a stable queue is equal to the long-term average arrival rate multiplied by the long-term average time an event spends in the system [6]. Based on this theorem, the response time versus the utilisation can be examined, $W = \frac{1/\mu}{1-\rho}$, where $\rho = \frac{\mu}{\lambda}$ is the utilisation. Assuming $\mu = 1$ and varying the utilisation rate, the response time has two regimes as shown in Figure 2. For utilisation rates below ~ 70% the response time grows linearly. But for higher utilisation rates the response time has an exponential growth. So, if greedy link selection is employed, sub-optimal decisions have a dramatic impact if the queue is in the sensitive regime. This is why the fluctuations are observed with the extended Q-routing algorithm described above.

Tao et al. introduced a multiagent, partially observable Markov Decision Process for packet routing. Each node in the network is parameterised by a real-valued vector for each destination/outgoing link pair [18]. This vector is adjusted in order to ascend the gradient of the expected long-term average reward for all nodes. The rewards are computed at the destinations of the packets and broadcast into the network. The probability of selecting a link is calculated according to the Gibbs distribution using this vector. This approach is very similar to the application of collective intelligence (COIN) for packet routing [23]. Tumer and Wolpert mathematically formalised collective intelligence solutions and proved that "Tragedy of the Commons" does not exist under certain conditions [19]. These are that the environment can be factored into sub-worlds, which encompass all nodes that share the same destination. The reward is also computed at the destinations and is broadcast in its own sub-world. Each agent maximising its long-term average reward also leads to maximising the global reward. In an extended study, Wolpert and Tumer show that COIN-based models for network routing almost always avoid the Braess' paradox [22]. Braess' paradox states that selfish routing behaviour on a network can result in a lower throughput when additional capacity through a new edge in the network is introduced. In particular, the ideal shortest path algorithm introduced side-effects that lead to the observation of the Braess' paradox. With a COIN-based ap-

proach, Braess' paradox can almost always be avoided while at the same time exhibiting significantly improved global throughput performance.

In highly dynamic and large environments modelling other agents or extra communication effort becomes prohibitively expensive. Direct policy learning algorithms are a promising alternative, because they align with the expected reward for the actions. A direct policy called Fair Action Learning (FAL) is presented in [24]. FAL approximates the policy gradient of each state-action pair using the difference between the expected Q-value for the state and its actual Q-value. As such it learns a stochastic policy that increases the probability of actions receiving a higher reward then the current average. Consequently, FAL will converge to a fair policy reflecting the expected reward for all actions and states. However, if one action is always more favourable than the other ones, FAL will converge to a deterministic policy, which is not always desirable.

The weighted policy learner (WPL) addresses this issue of ensuring that all actions have a minimum probability of being selected [1, 2]. The WPL algorithm was also designed with the need of quickly converging to a stable stochastic policy. This is achieved by performing a gradient ascent towards a stable policy and slowing down learning gradually for as long as the gradient does not change direction and learn fastest when the gradient changes direction. This policy ensures that no action probabilities converge to a deterministic policy using a euclidean projection onto the probabilistic simplex, where each probability is greater than a given value ϵ. Mathematically, this projection is equivalent to solving a constrained optimisation problem for which closed-form and efficient solutions exist whose complexity are linear in time [11]. This is advantageous in settings where agents have a large number of actions. The euclidean projection is given as $\Pi_X(x) = \arg\min_{x' : \text{valid}(x')}(x - x')$, which returns a policy that is closest to x and satisfies the constraints that it sums to 1 and action probabilities are greater than a given parameter ϵ. The weighted policy learning (WPL) algorithm has been applied to distributed task allocation, a similar setting as described in this article, where stochastic stable policies are desirable [2].

Both FAL and WPL use the expected Q-value for the state and its actual Q-value to calculate the gradient. This is in contrast to "Win or Learn Fast" (WoLF) algorithms, such as Generalised Infinitesimal Gradient Ascent WoLF (GIGA-WoLF) [7], which use approximations to determine when an agent is moving towards or away from a Nash Equilibrium.

4. COGNITIVE POLICY LEARNER

This section introduces a policy learning algorithm that is inspired by the limbic system of the brain. There are two basic concepts underlying the cognitive policy learner: firstly, an inherent bias that magnifies or depresses rewards depending on how far they diverge from the average expected reward for different actions in that state, and, secondly, a transient memory of recent rewards for that action that smooth out the current reward. Rewards are categorized as either *positive* or *negative*, depending on whether they are higher or lower than the average expected reward, respectively. Both positive and negative rewards are scaled by the amount they differ from the average expected reward and a fixed bias called the amplitude, $A^{+/-}$. A^+ scales positive rewards, while A^- scales negative rewards. In addition to biases, a transient memory model stores an accrued sum of recent positive rewards, $c^+(a)$, and recent negative rewards, $c^-(a)$. Both $c^{+/-}(a)$ are decayed over time at a configurable rate of decay, $r^{+/-}$. To give an example, this enables us to define a reward model that amplifies positive rewards and spreads out the assignment of the reward over time. So, a positive reward may persist for longer than the current time step. Additionally, the factor for the amplitude can be used to interpret the intensity of the positive or negative feedback signal. For example, by assigning an amplitude twice as high to the positive signal compared to the negative signal, positive signals have a larger impact on the update step on the probabilistic simplex than negative ones. This setting embodies some notion of risk aversion, because punishment does not induce a rapid update of one's strategy to avoid similar negative experiences. While, there might be situations favouring such a setting, it is more intuitive and in fact more natural to have risk-averse agents. Hence the win or learn fast strategies.

CPL is presented in Algorithm 1. The basic principle is similar to the weighted policy learner. Before conducting the update on the policy simplex, the memory for each signal is decayed, and the new signal is multiplied by the amplitude and added to the decayed signal. The memory signals are bounded, i.e., $0 < c^+(a) \leqslant s(a)_{\max}$ and $s(a)_{\min} \leqslant c^+(a) < 0$, where $s(a)_{\min\,/\,\max}$ are the minimum negative and maximum positive observed feedback signal. This means that the accrued feedback signals cannot attain values higher than the single strongest component of the feedback signal. The resultant positive and negative signals are added together, giving $\Delta(a)$, and the vector of all such signals for all actions, Δ, is used to proceed with the policy projection routine $\pi \leftarrow \Pi_X(\pi + \zeta\Delta)$. ζ denotes the update rate also used in FAL and WPL.

Algorithm 1: CPL: Cognitive Policy Learner

Input: $Q(s, a)$, the expected reward for executing action a in state s
Input: $c^+(a)$ & $c^-(a)$, the accrued reward/punishment signal for action a
Input: $A^{+/-}$ & $r^{+/-}$, the amplitude and decay rate for the respective feedback signals

$\bar{Q} = \sum_{a \in A} \pi(a) Q(s, a)$
foreach *action* $a \in A$ **do**
  $c^+(a) \leftarrow c^+(a) * e^{r^+ t}$
  $c^-(a) \leftarrow c^-(a) * e^{r^- t}$
  Decay
  $s(a) \leftarrow Q(s, a) - \bar{Q}$
  if $s(a) > 0$ **then** $c^+(a) \leftarrow c^+(a) + A^+ * s(a)$
  else $c^-(a) \leftarrow c^-(a) + A^- * s(a)$
  $\Delta(a) \leftarrow \max(c^-(a), s(a)_{\min}) + \min(c^+(a), s(a)_{\max})$
end
$\pi \leftarrow \Pi_X(\pi + \zeta\Delta)$
Output: A new policy π
Output: Updated reward/punishment signals $c^+(a)$ & $c^-(a)$

The amplitude parameter can be tuned in four different ways to modulate the effects of the positive and negative feedback signals:

1. $A^+ > A^-, r^+ > r^-$: Positive feedback signals are amplified more than negative ones. Also, accrued positive rewards decay at a slower rate.

2. $A^+ > A^-, r^+ < r^-$: Positive feedback signals are amplified more than negative ones. In contrast to the previous case, accrued negative rewards decay at a slower rate.

3. $A^+ < A^-, r^+ > r^-$: Negative feedback signals are amplified more than positive ones. Also, accrued negative rewards decay at a slower rate.

4. $A^+ < A^-, r^+ < r^-$: Negative feedback signals are amplified more than negative ones. In contrast to the previous case, accrued positive rewards decay at a slower rate.

If both decay rates are set to $-\infty$ and $A^+ = A^-$ then the fair action learner is recovered. If $A^+ < A^-$, then the cognitive policy learner resembles the effects of the weighted policy learner with a win or learn fast strategy without taking the accrued reward/punishment signals into account.

5. EVALUATION

We evaluate the cognitive policy learner using a small queueing network given in Figure 1. The respective external Poisson arrival, λ_0, and Exponential service rates, μ, are given in Table 1. The network represents three decision makers, nodes 4, 6, and 7. Each node's objective is to optimise the routing decisions of the tasks based on the negative completion times. Node 4 experiences neighbours 1 and 2 with different external arrival and service rates. Node 1 has a lower intrinsic utilisation than node 2 and consequently, node 4 needs to learn a policy that balances this difference such that the reward is maximised, or the task completion times are minimised in the long run. Both nodes 6 and 7 rely on the assignment of tasks given to node 4. Due to the reward functions having optimal substructure, the queueing performances of nodes 6 and 7 improve if node 4 learns an optimal policy. Node 7 has a higher external arrival rate of tasks and therefore receives a higher number of feedback signals from its task assignments than node 6. This implies that node 7 may be faster in recognising deteriorating performances than any of its neighbours. Both nodes share a common resource (node 4) and have each a private resource (node 3 and 5 respectively). Because node 3 has a much lower intrinsic utilisation than node 5, node 6 may be the first to utilise this resource in case node 4 deteriorates.

Table 1: Arrival and Service Rates

Node	1	2	3	4	5	6	7
λ	0.33	0.51	0.04	0.11	0.21	0.32	0.51
μ	0.68	0.9	0.48	0.76	0.58	0.55	0.55

The initial policy assumes uniformly random probabilities and three policy learning algorithms are evaluated including CPL, the fair action learner (FAL) [24] and the weighted policy learner (WPL) [1]. The underlying euclidean projection is the same for all three algorithms to ensure that action probabilities do not attain values less than a specified parameter $\epsilon = 0.05$. Each algorithm was individually optimised within the range of parameters $\alpha \in [0.0001; 0.1]$, $\lambda \in [0.01; 0.9]$, $\eta \in [0.01; 0.5]$, $\zeta \in [0.1; 0.0001]$ for both FAL and WPL and additionally $A^+ \in [0.01; 2.0]$, $A^- \in [0.01; 2.0]$, $r^+ \in [-20.0; -0.01]$, $r^- \in [-20.0; -0.01]$ for CPL using Gaussian Process Regression [3, 10, 17]. The results of this global optimisation are summarised in the following Table 2.

In all three algorithms the learning rate, discount factor, and the momentum for the $SARSA(0)$ gradient-update descent (Equation 3) are 0.1, 0.9 and 0.5 respectively. The update factor on the policy simplex, ζ, is low for FAL and high for WPL and CPL. The optimal parameters for the cognitive policy learner resemble the win or learn fast strategy, because both the amplitude for the negative signal is higher and the decay rate slower. This means that the memory for negative signals persists for a longer period of time. This result is interesting in that the global simulation optimisation technique found optimal values for CPL that reflect risk-aversion.

Table 2: Optimal Learning Parameters

	FAL	WPL	CPL
α	0.1	0.1	0.1
λ	0.9	0.9	0.9
η	0.5	0.5	0.5
ζ	0.0001	0.1	0.1
A^+			1.76
A^-			2.0
r^+			-1.7
r^-			-4.55

The analysis of the performance and the dynamics of the different algorithms are based on at least 10 replications of simulation runs using the optimal parameters. These simulation runs are also controlled to be within 90% confidence intervals with a relative error of 10% [13].

Figures 3 and 4 present the queueing results with respect to mean utilisation of the queueing network and total average delay in the queues. The delay measures the time a task waits in the queue until it can be serviced, since each node in the queueing network can only process one task at a time.

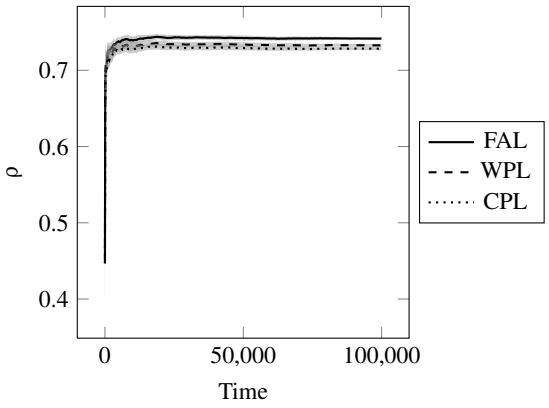

Figure 3: Utilisation

Both utilisation and delay are inferior for the fair action learner, while WPL and CPL show similar queueing performance. The utilisation and the 95% confidence interval half-widths for WPL and CPL respectively are 73.3%($\pm$0.0009) and 72.8%($\pm$0.0009).

Figure 5 shows the percentage of unstable nodes across the replications in order to illustrate the dynamics of WPL and CPL in the steady state using Equation 1. An unstable queue is defined as having a utilisation rate larger than 100% in the steady state. Intuitively, unstable queues show a behaviour of processing the tasks slower than they arrive, which leads to a growing waiting queue.

This calculation is equivalent of assuming the current policy to be fixed. FAL does not yield unstable queues at any given time and is, therefore, not shown in this plot. Nodes 1 and 4 show similar values for the percentage of unstable queues (23% and 8% respectively). But the percentage of unstable queues is increased for CPL for nodes 2 and 5 (2.5 and 9 percentage points higher). This may be explained by the fact that both their respective alternative paths have a lower intrinsic utilisation and since CPL has an accrued memory of the feedback signals, it is slower to adapt to rapidly

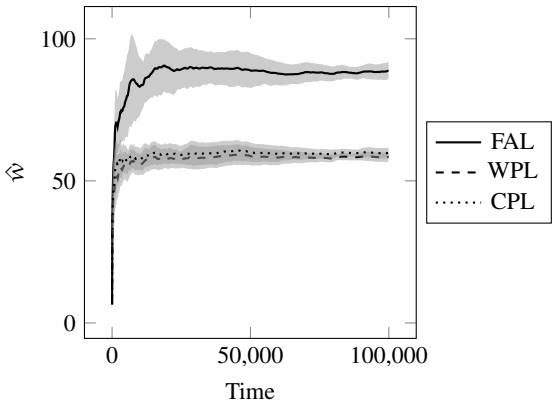

Figure 4: Delay

changing conditions.

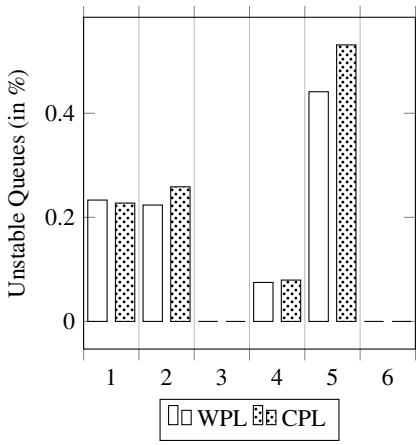

Figure 5: Stability of the Queues

The dynamics of queueing performance measures can be captured in several ways. The first metric we use is based on the matrix of routing probabilities $\mathbf{Q}$ derived from the employed policy. The following quantity captures learning as distinct from random behaviour, where uniformly random decisions manifest themselves in all outgoing links, $\deg^+(\cdot)$, in the network having equal probabilities of being selected. The distance measure from random behaviour is denoted as

$$d_n = \|\mathbf{Q}_{(n)} - \mathbf{Q}^r_{(n)}\|_1, \ \forall\, n \in \mathcal{V} \,\&\, \deg^+(n) > 0, \qquad (5)$$

where $\|\cdot\|_1$ is the ℓ_1-norm of a vector, i.e., $\|\mathbf{a}\|_1 = \sum_{i=1}^{n} |\mathbf{a}_i|$ and $\mathbf{Q}^r_{nj} = \frac{1}{\deg^+(n)}$ is the probability of taking a uniformly random action for all actions j available to n. The probability of taking actions $\mathbf{Q}_{(n)}$ is derived from the employed policy. This measure is bounded by $d_n = 0$, if the action selection probabilities are uniformly random, and $\sup\{d_n\} = 2$ for deterministic action selection as $\deg^+(n) \to \infty$. Also, $d_n = 0,\ n \in \mathcal{V} \,\&\, \deg^+(n) = 1$.

This metric does not make any qualitative statement about learning behaviour, because it cannot be ruled out that uniformly random behaviour is actually the best policy. Instead it gives an indication of how distinctive the learnt policies are.

Figure 6 shows the result for this metric. FAL learns the least distinctive policies in the queueing network, which means that the policy updates are very close to the initial policy configuration of uniformly random decisions. Additionally, the policy gradient update factor, ζ, for FAL is low, suggesting that FAL prefers small incremental changes to the policy. CPL in turn learns the most distinctive policies. These results show that temporarily unstable queues in the steady state lead to a higher throughput in the algorithms considered in this evaluation.

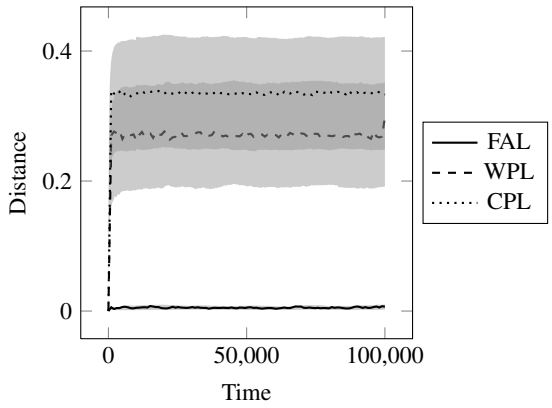

Figure 6: Distance from Uniformly Random Decision Policies

In order to quantify the level of distinctiveness of the policies we evaluate the ownership of node 4. The ownership metric is calculated as the normalised fourth column of the routing matrix $\mathbf{Q}$, where nodes 6 and 7 are the only predecessors. Figure 7 depicts this evolution of ownership. In all cases node 6 directs most of its tasks towards node 3, while node 7 directs most of its tasks towards node 4. Since node 7 also has a higher arrival rate with the same service rate compared to node 6, node 7 dominates node 4. This plot also mirrors our previous result that CPL learns more distinctive policies, i.e., the spread between nodes 6 and 7 is higher for CPL. Importantly, FAL exhibits barely any fluctuations in its learning dynamics. This shows that FAL learns a stochastic stable policy, while WPL and CPL learn stochastic unstable policies. An interesting result, however, is that this instability (which exists only in the steady state) yields a better performing system as a whole.

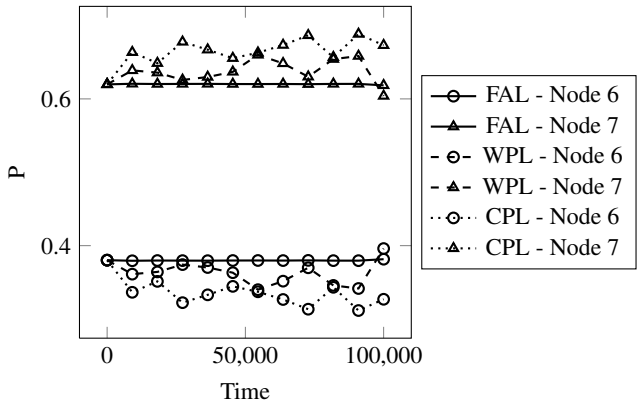

Figure 7: Ownership of Node 4

In order to understand the extent of the fluctuations, the coefficient of variation of the Q-value estimation is calculated based on a buffer of the last 100 Q-value estimations:

$$c_v(Q) = \frac{\sigma_Q}{\mu_Q}. \quad (6)$$

Figure 8 shows the densities of the coefficient of variation for each decision making node individually. Intuitively, one would assume that the coefficient of variation scales with the height of the queueing network, i.e., the least dependent node (here node 4) has lower values than the nodes that depend on it. Interestingly, this is only observed for FAL, which can be interpreted as FAL's learning dynamics results in cascading effects. Because of this behaviour, FAL learns the least distinctive policies and also performs poorly compared to WPL and CPL. Because the values for the coefficient of variation are significantly higher than the ones with WPL and CPL, the densities are left out of Figure 8.

WPL and CPL on the other hand do not exhibit cascading effects. In fact, the fluctuations observed for those algorithms appear to have a stabilising impact on the learning dynamics. For all nodes, the absolute value of the coefficient of variation is slightly higher for CPL compared to WPL.

Figure 9 presents the total mean-squared error of the loss function of the neural network excluding FAL, because it is significantly higher than WPL and CPL again. This plot suggests that the CPL algorithm yields a more stable reinforcement learning system than WPL as its total mean-squared error is significantly lower.

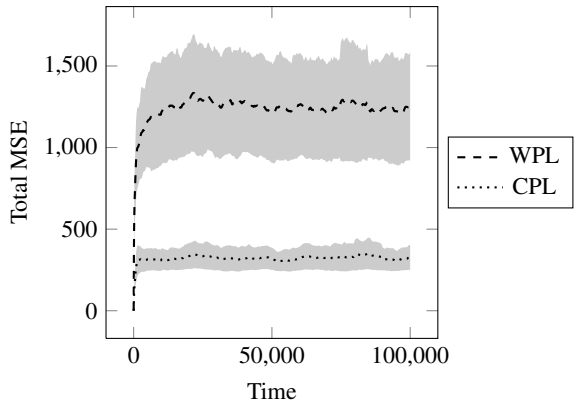

Figure 9: Neural Network Error

To summarise the results, FAL has inferior queueing performance than WPL and CPL. FAL also learns the least distinctive policies, which may be attributed to the large fluctuations in the coefficient of variation of the Q-value estimation. These fluctuations lead to cascading effects in the fair action learner and therefore this policy is not a good candidate for autonomous agents in queueing networks. Instead, both WPL and CPL learn stochastic unstable policies with respect to the instantaneous steady state distribution. However, these instabilities only persist temporarily with agents continuously adapting to the changing conditions in the queueing environment. In fact, these instabilities do not exhibit cascading effects. The coefficients of variation are approximately normally distributed with a slight skewness towards the left. This is in contrast to the distribution for FAL, which has a long tail representing rare events. In multiagent systems behaviours that can be characterised by long-tailed distributions introduce challenges for the other agents to adapt accordingly.

Finally, the mean-squared error of the neural network is the lowest for the CPL algorithm, which implies more stable reinforcement learning updates. However, the structure of the neural network is considered fixed in this paper. Optimising with respect to the neural network structure itself may be one way of reducing the mean-squared error for both FAL and WPL.

6. CONCLUSION AND FUTURE WORK

This paper investigated an extension to the weighted policy learner which modulates the strength of positive and negative feedback. The cognitive policy learner is inspired by the limbic system of the brain. The feedback signal is split into two parts, positive and negative, with respect to the current estimate of the Q-value function. Each signal is given free parameters to model an amplitude and a decay factor. This way the win or learn fast strategy obtains a higher level of control in terms of the updates on the probabilistic simplex. We showed that the cognitive policy learner performs as well as the weighted policy learner.

The empirical investigation of a small queueing network also revealed that the fair action learner exhibits cascading effects in the queueing network. This means that deteriorating performance closer to the leaf nodes of the network has a detrimental impact on the queueing performance of the other nodes dependent on them. This behaviour was not observed with the weighted and cognitive policy learners where the variations of the Q-value estimation is much better behaved. Biasing a losing strategy and maintaining a transient memory of received rewards and punishments results in a more stable multiagent learning system, which was shown to reduce the total mean-squared error of the $\mathrm{SARSA}(0)$ steepest descent gradient update.

Future work will investigate more dynamic and larger queueing settings. Also, the global optimisation of the simulation parameters need to be analysed with respect to how sensitive the optimal values are to small perturbations.

7. ACKNOWLEDGMENTS

The work described in this paper was supported by the Science Foundation Ireland. We thank the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] S. Abdallah and V. Lesser. Learning the task allocation game. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 850–857, New York, NY, USA, 2006. ACM.

[2] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[3] B. Ankenman, B. L. Nelson, and J. Staum. Stochastic kriging for simulation metamodeling. In *2008 Winter Simulation Conference (WSC)*, pages 362–370. IEEE, December 2008.

[4] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[5] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[6] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. WileyBlackwell, 2nd edition edition, May 2006.

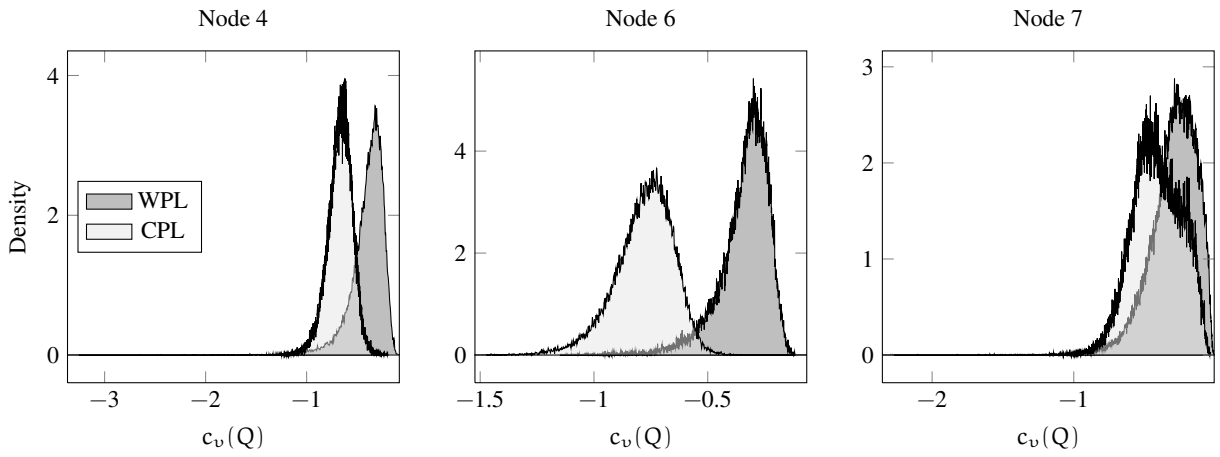

Figure 8: Distribution of CVs

[7] M. Bowling. Convergence and no-regret in multiagent learning. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 209–216. MIT Press, Cambridge, MA, 2005.

[8] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6*, volume 6, pages 671–678, 1994.

[9] G. Chalkiadakis and C. Boutilier. Bayesian reinforcement learning for coalition formation under uncertainty. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1090–1097, Washington, DC, USA, 2004. IEEE Computer Society.

[10] D. Dahlem and W. Harrison. Collaborative function approximation in social multiagent systems. In *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 48–55, Los Alamitos, CA, USA, September 2010. IEEE Computer Society.

[11] J. Duchi, S. S. Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 272–279, New York, NY, USA, 2008. ACM.

[12] J. Hu and M. P. Wellman. Nash q-learning for general-sum stochastic games. *J. Mach. Learn. Res.*, 4:1039–1069, 2003.

[13] A. M. Law and D. W. Kelton. *Simulation Modelling and Analysis*. McGraw-Hill Education - Europe, April 2000.

[14] M. L. Littman. Friend-or-foe q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328. Morgan Kaufmann, 2001.

[15] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 246–253, New York, NY, USA, 2001. ACM.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, chapter 8, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[17] J. Staum. Better simulation metamodeling: The why, what, and how of stochastic kriging. In *Simulation Conference, 2009. WSC 2009. Winter*, December 2009.

[18] N. Tao, J. Baxter, and L. Weaver. A multi-agent policy-gradient approach to network routing. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 553–560, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[19] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, 1 edition, May 2004.

[20] K. Verbeeck, A. Nowé, J. Parent, and K. Tuyls. Exploring selfish reinforcement learning in repeated games with stochastic rewards. *Autonomous Agents and Multi-Agent Systems*, 14(3):239–269, November 2006.

[21] K. Verbeeck, J. Parent, and A. Nowé. Homo egualis reinforcement learning agents for load balancing. In *Innovative Concepts for Agent-Based Systems*, pages 81–91. Springer Berlin / Heidelberg, 2003.

[22] D. H. Wolpert and K. Tumer. Collective intelligence, data routing and braess' paradox. *J. Artif. Int. Res.*, 16(1):359–387, 2002.

[23] D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 952–958, Cambridge, MA, USA, 1999. MIT Press.

[24] C. Zhang, V. Lesser, and P. Shenoy. A multi-agent learning approach to online distributed resource allocation. In *IJCAI 2009, Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*, pages 361–366, July 2009.

[25] C. Zhang, V. R. Lesser, and S. Abdallah. Self-organization for coordinating decentralized reinforcement learning. In W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *AAMAS*, pages 739–746. IFAAMAS, 2010.