Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Coordinated Reinforcement Learning for Decentralized Optimal Control

Daniel Yagan and Chen-Khong Tham
Department of Electrical and Computer Engineering
National University of Singapore
Email: {daniel.yagan, eletck}@nus.edu.sg

***Abstract*— We consider a multi-agent system where the overall performance is affected by the joint actions or policies of agents. However, each agent only observes a partial view of the global state condition. This model is known as a Decentralized Partially-Observable Markov Decision Process (DEC-POMDP), which can be considered more applicable in real-world applications such as communication networks.**

It is known that the exact solution to a DEC-POMDP is *NEXP-complete* and memory requirements grow exponentially even for finite-horizon problems. In this paper, we propose to address these issues by using an online model-free technique and by exploiting the *locality of interaction* among agents in order to approximate the joint optimal policy. Simulation results show the effectiveness and convergence of the proposed algorithm in the context of resource allocation for multi-agent wireless multi-hop networks.

I. Introduction

Markov Decision Process (MDP) have been widely used as a mathematical framework for sequential decision-making in stochastic domains. In particular, a single agent controls the system to optimize a global objective, while it completely observes the state of the controlled Markov chain and acts based on its policy [1].

There has been a number of research works that extended the basic MDP model to multi-agent systems. Examples of such extensions include Multi-agent MDP [2], Partially-Observable Identical Payoff Stochastic Game (POIPSG) [3], and Communicative Multi-agent Team Decision Problem (COM-MTDP) [4]. As researchers apply these models in real-world scenarios, one may realize their limitations and disadvantages, especially on how each agent physically interacts with other agents in the system. An example is a typical communication network, where an agent does not observe the condition of the entire network and only interacts with a limited set of neighboring agents.

In order to capture these issues, a promising multi-agent model known as Decentralized Partially-Observable Markov Decision Process (DEC-POMDP) was proposed in [5]. DEC-POMDP is a *decentralized stochastic control* system, where the joint policy of the agents determines system performance. However, each agent only has local observations and does not have complete observability of the global state of the environment. It is known that exact solutions to a DEC-POMDP are complete for the complexity class non-deterministic exponential time (NEXP-complete) [5]. In other words, a general DEC-POMDP does not admit polynomial-time algorithms since P $\neq$ NEXP.

In this paper, we propose a model-free Reinforcement Learning (RL) solution to approximate the joint optimal policy for a DEC-POMDP. Our approach is based on the RL technique known as *policy gradient* that parameterizes and updates the policies of agents during execution. In addition, we also exploit the *locality of interaction* of neighboring agents for a distributed coordination mechanism to optimize the global long term performance.

DEC-POMDP has been widely studied recently. The concept of locality of interaction for DEC-POMDP was only introduced in the *model-based* algorithm in [6]. However, to the best of the authors' knowledge, our proposed algorithm is the first attempt to use a *coordinated model-free* mechanism to solve a multi-agent DEC-POMDP, while considering the locality of interaction of agents.

This paper is organized as follows. Section II summarizes the DEC-POMDP framework and discusses non-trivial complexity issues of exact optimal algorithms. In addition, we also discuss the idea of locality of interaction among agents in solving a DEC-POMDP. In Section III, we propose a model-free control algorithm known as *Locally Interacting Distributed Reinforcement Learning Policy Search* (LID-RLPS), where each agent uses a coordinated policy gradient mechanism among its neighbors to approximate the joint optimal policy efficiently in a decentralized manner. In Section IV, we study the performance of LID-RLPS in the context of resource allocation and scheduling for a multi-hop wireless network using the NS2 network simulator [7]. Finally, we conclude our work and describe future research directions in Section V.

II. Decentralized Stochastic Control

A. DEC-POMDP

A N-agent Decentralized Partially Observable Markov Decision Process (DEC-POMDP) can be expressed as the tuple [5]:

$$\left\langle S, \overrightarrow{A}, P, C, \overrightarrow{\Omega}, O, p_o \right\rangle \tag{1}$$

where:
S is a finite set of *global* states.
$\overrightarrow{A} = \{A_i\}$ is a finite set of joint actions, where A_i is the set of actions available to agent i.

$P(S'|\overrightarrow{a}, S)$ denotes the probability that the next state is S' given that the agents execute the joint action $\overrightarrow{a} = \{a_1, ..., a_N\}$ when the current state is S.
$\overrightarrow{\Omega} = \{\Omega_i\}$ is a finite set of joint observations, where Ω_i is the set of observations by agent i.
$O(\overrightarrow{o}|S, \overrightarrow{a}, S')$ is the probability of jointly observing $\overrightarrow{o} = \{o_1, ..., o_N\} \in \overrightarrow{\Omega}$ when the agents take actions $\overrightarrow{a}$ in state S, resulting in state S'.
$C(S, \overrightarrow{a}, S')$ denotes the immediate cost function.
p_0 is the initial state distribution of the system.

In a DEC-POMDP, the joint action of the agents and the current state determine the next state. However, each agent only observes its own local observation o_i. In addition, none of the agents know the complete state of the system. Note that this is similar to the centralized single-agent Partially Observable Markov Decision Process (POMDP) framework where an agent uses its history of local observations and actions to find its optimal policy [8], [9].

The local history of observations for agent i up to time t is defined as:

$$\overrightarrow{o_i^h}(t) = \{o_i(1), ..., o_i(t)\} \tag{2}$$

A local policy w_i for agent i is defined to be a mapping from its observation history $\overrightarrow{o_i^h}(t)$ to its local actions $a_i(t)$. A joint policy $\overrightarrow{w} = \{w_1, ..., w_N\}$ is defined to be a tuple of local policies. Solving a DEC-POMDP can be seen as finding a set of N policies, one for each agent [5].

In this paper, we only consider the average cost criterion. The average cost of a particular joint policy $\overrightarrow{w}$ for a given initial state $S(0) = s_0$ is defined as:

$$J(\overrightarrow{w}, s_0) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E_{s_0}^{\overrightarrow{w}} \{C(S(t), \overrightarrow{a}(t), S(t+1))\} \tag{3}$$

A policy $\overrightarrow{w^*}$ is optimal if $J(\overrightarrow{w^*}, s_0) \leq J(\overrightarrow{w}, s_0)$ for all policies $\overrightarrow{w}$ and any initial state s_0.

B. Exact Optimal Algorithms for DEC-POMDP

As mentioned in the previous subsection, DEC-POMDP appears to be a POMDP, where each agent only has local observation and the joint policies of the agents determine the next state transition. In this sense, one can intuitively convert a N-agent DEC-POMDP to N independent POMDPs and use established POMDP techniques [10], [11].

However, the authors in [5] have shown that, on the contrary, a DEC-POMDP requires a fundamentally different algorithmic structure. By reducing the control problem to a *tiling* problem, they have shown that if the underlying transition probability function is known, the DEC-POMDP in a *finite-horizon* with a constant number of agents (i.e. $N \geq 2$) is *complete* for the complexity class non-deterministic exponential time (NEXP). This implies that problems modeled as DEC-POMDP *provably* do not admit polynomial-time algorithms. This trait is not shared by finite-horizon MDP or POMDP problems and thus, has direct implications when solving problems involving distributed agents.

It should be noted that, even if one can convert a DEC-POMDP into N independent POMDPs, exact POMDP methods are PSPACE-hard [9]. Hence, approximate tractable solutions are preferable.

Recently, an exact Dynamic Programming algorithm was proposed for a general DEC-POMDP [12]. Though the algorithm was used in a finite-horizon context, the authors mentioned ways to extend it to the infinite-horizon case. Their model-based algorithm uses *policy trees* that enumerate the possible policies at each state and every possible next state transitions up to a given depth in the tree, and performs pruning of tree branches. The algorithm obviously suffers from large memory requirements with each iteration as the tree grows and in practice, has only been used to solve very small problems. It is likely that any exact optimal algorithm would suffer this curse of dimensionality and exponential-time complexity, due to the *NEXP-complete* complexity result in [5].

C. Locality of Interaction among Neighboring Agents

While a general DEC-POMDP captures real-world uncertainty in multi-agent domains, it fails to exploit the fact that each agent has limited interactions with a small number of neighboring agents.

Following [6] for finite-horizon problems, we apply the concept of *locality of interaction* under the infinite-horizon criterion. To illustrate this idea, we consider a slotted communication network with N nodes or agents. We shall use this scenario for the simulation in Section IV.

We define the global state of the N-agent DEC-POMDP as follows:

$$S(t) := \overrightarrow{x}(t) = [x_1(t), .., x_N(t)] \tag{4}$$

where:
$\overrightarrow{x}(t)$ is a *factored* representation or concatenation of the local observation $x_i(t)$ for all i, where $x_i(t)$ represents the *queue length* or congestion level at node i.

The immediate cost function $C(S(t), \overrightarrow{a}(t), S(t+1))$ is defined as the congestion level $C(\overrightarrow{x}(t), \overrightarrow{a}(t))$ among nodes:

$$C(\overrightarrow{x}(t), \overrightarrow{a}(t)) = \sum_{i=1}^{N} x_i(t) \tag{5}$$

Due to the interaction among neighbors, we observe that the immediate cost function is a summation of cost functions of a sub-group of agents. Consider the network scenario with six nodes shown in Figure 1.

In a single time slot t, the queue length of a node i is only affected by neighboring nodes $\{N_i\}$. The immediate cost or congestion function for N nodes can be simply expressed as:

$$C(\overrightarrow{x}, \overrightarrow{a}) = \sum_{i=1}^{N} C(x_i, x_{N_i}, a_i, a_{N_i}) \tag{6}$$

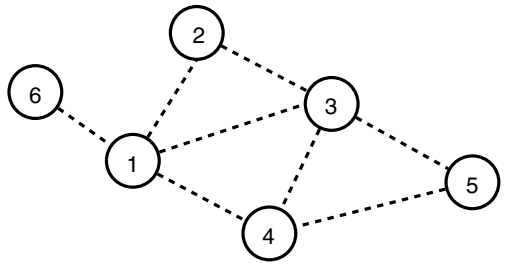

Fig. 1. Locality of interaction among neighboring nodes

where:
x_{N_i} is a vector representing the local observations of the neighboring agents N_i of agent i (i.e. $x_{N_1} = [x_2, x_3, x_4, x_6]^T$ in Figure 1).
a_i is the chosen action of agent i at node i.
a_{N_i} is a vector for the actions of the neighboring agents N_i (i.e. $a_{N_1} = [a_2, a_3, a_4, a_6]^T$ in Figure 1).
$C(x_i, x_{N_i}, a_i, a_{N_i})$ is the immediate cost incurred by agent i which depends on its current local observation x_i and action a_i, and the relevant information x_{N_i} and a_{N_i} from its neighbors.

Let $C_{loc,i}(t) := \sum_k C(x_k, x_{N_k}, a_k, a_{N_k}),\ \forall k \in \{i \cup N_i\}$ be the immediate *localized* cost. This effectively represents the sum of the individual cost functions in (6) where x_i and a_i components are included at the current time slot t. We use this quantity to separate the immediate cost function $C(\overrightarrow{x}, \overrightarrow{a})$ into two independent components: one is affected by agent i, while the other is independent of agent i.

For instance, in Figure 1, the immediate cost function in (6) can be expressed in the following ways, depending on the index i in $C_{loc,i}$:

$$\begin{aligned} &C(\overrightarrow{x}, \overrightarrow{a}) = C_{loc,1} + C(x_3, x_4, x_5, a_3, a_4, a_5) \\ &= C_{loc,2} + C(x_1, x_3, x_4, x_5, a_1, a_3, a_4, a_5) + \\ &C(x_3, x_4, x_5, a_3, a_4, a_5) + C(x_1, x_6, a_1, a_6) \\ &= C_{loc,3} + C(x_1, x_6, a_1, a_6) \\ &= C_{loc,4} + C(x_1, x_2, x_3, a_1, a_2, a_3) + C(x_1, x_6, a_1, a_6) \end{aligned}$$

We define the *local neighborhood utility* of agent i as $B_{\overrightarrow{w}}(N_i, s_0)$ to represent the expected average long term cost for executing joint policy $\overrightarrow{w} = \{w_1, ..., w_N\}$, starting with some global state $S(0) = s_0$:

$$B_{\overrightarrow{w}}(N_i, s_0) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} E_{s_0}^{\overrightarrow{w}} \{C_{loc,i}(t)\} \quad (7)$$

Lemma 1: To find the best policy for agent i *given* its neighbors' policies in optimizing its local neighborhood utility, agent i does not need to consider the non-neighbors' policies.

Proof: We observe from (7) that the local neighborhood utilities of agent i for two joint policies $\overrightarrow{w_a} = [w_{a,1}, ..., w_{a,N}]$ and $\overrightarrow{w_b} = [w_{b,1}, ..., w_{b,N}]$ are equal, if the corresponding policy vector components are equal: $B_{\overrightarrow{w_a}}(N_i, s_0) = B_{\overrightarrow{w_b}}(N_i, s_0)$ if $w_{a,k} = w_{b,k}$ for all $k \in \{i \cup N_i\}$. Thus, any policy vector $\overrightarrow{w_b}$ that has different policies for only *non-neighborhood* agents as compared to policy $\overrightarrow{w_a}$ has equal value as $B_{\overrightarrow{w_a}}(N_i, s_0)$. Furthermore, given the neighbors' policies, optimizing the local neighborhood utility of agent i does not affect the local neighborhood utility of agent k if $k \notin \{N_i\}$. ■

The proof of Lemma 1 is similar to [6]. In the example above, we have considered a general communication network with the state descriptor in (4) and cost function in (5). For a general DEC-POMDP, we can refer to [6] for the concepts of locality of interaction and local neighborhood utility.

In summary, the main idea is to capture a local neighborhood utility that localizes the effect of agent i to the global cost function, given the policies of its neighbors. In the next section, we shall use the idea of Lemma 1 and [6] in proposing an algorithm for solving a DEC-POMDP, while considering locality of interaction among agents. We emphasize that the proposed model-free algorithm differs from model-based solution in [6].

III. Proposed Model-Free Algorithm for DEC-POMDP

A. Multi-Agent Finite-State Controller (MFSC)

Due to the complexity issues of exact model-based algorithms for DEC-POMDP as discussed in Section II-B, we use *model-free* techniques to *approximate* the optimal joint policy of agents.

When an agent does not completely observe the state of the system and that the underlying transition probability distribution is unknown, the agent needs *memory* of the past observations and actions to act optimally, as in the case of single-agent POMDP [9].

Following [9] for POMDP, we use the concept of a *finite-state controller* (FSC) to capture relevant past information to act optimally. Each agent i contains a finite-state controller which is represented as the tuple:

$$\langle I_i, \phi_i, f_i, \theta_i, \mu_i \rangle \quad (8)$$

where:
I_i is the set of *internal states* (I-states) of the FSC.
$f_i(\cdot)$ and $\mu_i(\cdot)$ are I-state transition and action selection distribution functions, respectively.
$\phi_i \in \mathbb{R}^{n_{\phi_i}}$ and $\theta_i \in \mathbb{R}^{n_{\theta_i}}$ are vector parameters.

FSC uses ϕ_i as a n_{ϕ_i}-dimensional vector to parameterise the I-state transition probabilities, while θ_i is a n_{θ_i}-dimensional vector used to parameterise the action selection probabilities in (8). Each agent learns to use the I-states I_i to remember only what is needed to act optimally. Specifically, the I-state transitions and action selections are *learned* by intelligently searching the space of parameters ϕ_i and θ_i. The actual policy search mechanism is discussed in the next subsection.

From [6], [13], for a DEC-POMDP, each agent faces a complex but normal single-agent POMDP if the policies of all other agents are *fixed* at a given decision instant. However, in addition to the unobserved state $S(t)$, each agent must also reason about the action selection and observation histories of other agents.

Furthermore, as discussed in Section II-C and shown in Lemma 1, given the neighbors' policies, agent i does not need to consider non-neighboring agents to find its best policy. The local neighborhood utility $B_{\overrightarrow{w}}(N_i, s_0)$ in (7) has effectively localized the effect of agent i's policy to the global cost function in (6). In other words, agent i only has to optimize $B_{\overrightarrow{w}}(N_i, s_0)$, *given* the policies of the neighboring agents.

From these concepts, we propose a *multi-agent finite state controller* (MFSC) for agent i which extends the FSC in (8) and captures the locality of interaction of agents.

Following [6], in order to exploit the locality of interaction, we define the tuple $\overrightarrow{e_i}(t)$ as:

$$\overrightarrow{e_i}(t) = \left\langle S_{i,N_i}(t), \overrightarrow{o^h_{N_i}}(t) \right\rangle \tag{9}$$

where:
$S_{i,N_i}(t)$ is a vector containing the local observations of neighboring agents and agent i at the current time slot t.
$\overrightarrow{o^h_{N_i}}(t) = \left\langle \overrightarrow{o^h_k}(t) \right\rangle$ represents the joint observation histories of neighbors for all $k \in \{N_i\}$ up to time t, where $\overrightarrow{o^h_k}(t)$ is defined in (2).

Given the policy of neighboring agents, treating $\overrightarrow{e_i}(t)$ as state of agent i results in a single-agent POMDP [6]. The *multi-agent belief state* for an agent i given the distribution over the initial state $p_0(s_0) = P(S(0) = s_0)$ is defined as [13]:

$$B_i(t) = Pr(\overrightarrow{e_i}(t) | \overrightarrow{o^h_i}(t), \overrightarrow{a^h_i}(t-1), p_0(s_0)) \tag{10}$$

where:
$\overrightarrow{a^h_i}(t-1)$ is the history vector of actions up to time $(t-1)$.
$\overrightarrow{o^h_i}(t)$ is the observation history for agent i up to time t.

Thus, when reasoning about the agent's policy in the *context* of other neighboring agents (i.e. neighboring agents' policies are fixed at the *current context*), we maintain a distribution over $\overrightarrow{e_i}(t)$ in (9). We emphasize that considering $\overrightarrow{e_i}(t)$ results in a single-agent POMDP only for a given set of policies of neighbors at the *current context* where the neighbor policies are fixed.

Furthermore, to capture the multi-agent belief state $B_i(t)$ under the MFSC with unknown transition probabilities (i.e. model-free), the MFSC must also depend on the policy parameters of neighboring agents, and not just ϕ_i and θ_i for agent i in the original FSC in (8).

Hence, we define the internal state transition distribution of the MFSC as:

$$f_i(\cdot \mid \phi_i, g, y, a_i(t-1), \delta_{N_i}) \tag{11}$$

where:
$g \in I_i$.
y is current local observation.
$a_i(t-1)$ is the previous local action.
$\delta_{N_i} \in \mathbb{R}^{n_{\delta_i}}$ is a n_{δ_i}-dimensional vector that captures the effect of the neighbors' policy parameters $\langle \theta_k, \phi_k \rangle$ for all $k \in \{N_i\}$.

Similarly, the action selection policy is based on the action selection probabilities defined as $\mu_i(\cdot \mid \theta_i, h, y, \delta_{N_i}) > 0$ for each I-state h, local observation y, and parameters θ_i and δ_{N_i}. The I-state transitions and policy distribution are *learned* by searching the space of parameters ϕ_i and θ_i.

It is known from [14] that if $\overrightarrow{e_i}(t)$ is the state defined in (9) and $g(t) \in I_i$ is an internal state of the MSFC for agent i, then the tuple $\langle \overrightarrow{e_i}(t), g(t) \rangle$ forms a Markov chain. Interaction begins at an initial state $\overrightarrow{e_i}(0)$ and agent i completely observes its own initial I-state $g(0) \in I_i$. Given the *fixed* neighbors' policies $\overrightarrow{w_{N_i}}$, the goal of agent i is to find ϕ_i and θ_i that minimizes the local neighborhood utility, independent of the initial state:

$$B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}}) = \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_{\phi_i, \theta_i} \{ C_{loc,i}(t) | \overrightarrow{w_{N_i}} \} \tag{12}$$

where the expectation E_{ϕ_i, θ_i} denotes the expectation over all trajectories $\langle \overrightarrow{e_i}(0), g(0) \rangle, \langle \overrightarrow{e_i}(1), g(1) \rangle, ...$

This search process can be viewed as an automatic *quantization* of the multi-agent belief state $B_i(t)$ in (10) to provide the optimal policy representable by $\|I_i\|$ internal states. As $\|I_i\| \to \infty$, we can represent the optimal policy accurately, *without* knowing the exact model of the system [15].

In summary, a MFSC consists of the tuple $\langle I_i, \phi_i, f_i, \theta_i, \mu_i \rangle$ with internal state transition distribution in (11) and action selection distribution $\mu_i(\cdot \mid \theta_i, h, y, \delta_{N_i})$. Agent i searches the space of parameter vectors ϕ_i and θ_i to act optimally, with consideration of neighbor policy parameters δ_{N_i}, *without* knowing the state transition and observation probabilities of the DEC-POMDP in (1).

The next subsection describes the actual parameter search mechanism and how each agent coordinates its own policy search among its neighbors to attain the global optimal average cost.

B. Model-Free Policy Generation algorithm

The idea of exploiting locality of interaction in distributed agents to optimize a global objective function has already been addressed in the formalism known as *Distributed Constraint Optimization* (DCOP) [16], [17]. A DCOP problem includes a set of variables, each variable is assigned to an agent who can control its value, and agents must coordinate their choice of values. DCOPs have successfully exploited limited agent interactions in multi-agent systems, with over a decade of algorithm development. However, DCOPs do not capture planning under uncertainty as compared to DEC-POMDP.

In this sub-section, we extend the concepts of *Networked Distributed-POMDP* (ND-POMDP) and the *model-based* LID-JESP algorithm [6]. ND-POMDP combines DEC-POMDP and an N-ary DCOP, where N is the number of

Algorithm 1 Locally Interacting Distributed Reinforcement Learning Policy Search (LID-RLPS)

Let $t = 0$ and $T =$ required number of iterations.
Each agent i starts with a random policy w_i represented as $\langle \theta_i, \phi_i \rangle$.
Let w_{N_i} be the policies of agent i's neighbors $\{N_i\}$, and are represented as the set of policy parameters $\{\theta_k, \phi_k\}\ \forall k \in \{N_i\}$.
Let g_t be the I-state of the agent at time t.
While $t < T$
  Obtain y_t as local observation
  Agent i exchanges policies w_i with neighbors $\{N_i\}$
  Form δ_{N_i} as a *feature* vector from $\langle \theta_k, \phi_k \rangle\ \forall k \in \{N_i\}$
  Choose g_{t+1} from $f_i(\cdot \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$
  Choose and execute action $a_i(t)$ from $\mu_i(\cdot \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$
  Get the immediate localized cost $C_{loc,i}(t)$
  $cValue =$ *GetEstimate*(w_i, δ_{N_i})
  $w_i^* = \langle \theta_i^*, \phi_i^* \rangle =$ *LocalBestPolicy*$(w_i, \delta_{N_i}, y_t, C_{loc,i}(t))$
  $mValue =$ *GetEstimate*(w_i^*, δ_{N_i})
  $gain_i = \|cValue - mValue\|$
  Broadcast $gain_i$ to $\{N_i\}$
  $maxGain = \max_{k \in \{i \cup N_i\}} gain_k$
  $winner = \arg \max_{k \in \{i \cup N_i\}} gain_k$
  If $maxGain > 0$
    If $i = winner$ then
      Update policy: $w_i = w_i^*$ or $\langle \theta_i, \phi_i \rangle = \langle \theta_i^*, \phi_i^* \rangle$
      Broadcast w_i^* to $\{N_i\}$
    Else
      Receive policy w_{winner} from winner
      Update w_{N_i}
    End If
  Else
    Break While;
  End If
  $t = t + 1$
End While

agents and the DCOP variable at each node is the individual agent's policy.

We call our proposed *model-free* algorithm as: *Locally Interacting Distributed Reinforcement Learning Policy Search* (LID-RLPS) that uses the MFSC to generate the policies (i.e. find best policy vectors) while considering locality of interaction, and without the state and observation transition model of DEC-POMDP. The pseudo-code of LID-RLPS is shown in Algorithm 1.

The proposed LID-RLPS algorithm can be summarized as follows: Each agent i starts with a random local policy and exchanges its policies $\langle \theta_i, \phi_i \rangle$ with its neighbors $\{N_i\}$. Note that the policy w_i of agent i is represented by the policy vector $\langle \theta_i, \phi_i \rangle$. During the exchange of policies, the actual policy vector is communicated among the neighbors. Agent i then chooses an action from its action or policy distribution $\mu_i(\cdot \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$. The immediate localized cost $C_{loc,i}(t)$ is then obtained from its neighbors. Afterwards, agent i computes its local neighborhood utility in (12) with respect to its *current* policy and its neighbors' policies (i.e. function $GetEstimate(w_i, \delta_{N_i})$ in Algorithm 1). It then uses a *policy-gradient* model-free technique (i.e. function $LocalBestPolicy(w_i, \delta_{N_i}, y_t, C_{loc,i}(t))$ in Algorithm 2) to get the local neighborhood utility of agent i's *best* policy given the policies of its neighbors. The difference between the two local neighborhood utilities is represented as the *gain* message.

Each agent broadcasts its gain message among its neighbors

Algorithm 2 Finding the Best Local Policy Response

Given $\beta \in [0, 1)$. $\alpha_t = \frac{1}{t} =$ learning rate
Set $z_0^{\theta_i} = z_{new}^{\theta_i} = [0]$, $z_0^{\phi_i} = z_{new}^{\phi_i} = [0]$;
$\triangle_0^{\theta_i} = \triangle_{new}^{\theta_i} = [0]$; $\triangle_0^{\phi_i} = \triangle_{new}^{\phi_i} = [0]$;
$\phi_{i,new} = [0]$, $\theta_{i,new} = [0]$;
where $z_0^{\theta_i}, z_{new}^{\theta_i}, \triangle_0^{\theta_i}, \triangle_{new}^{\theta_i}, \theta_{i,new} \in \mathbb{R}^{n_{\theta_i}}$,
$z_0^{\phi_i}, z_{new}^{\phi_i}, \triangle_0^{\phi_i}, \triangle_{new}^{\phi_i}, \phi_{i,new} \in \mathbb{R}^{n_{\phi_i}}$.

Function *LocalBestPolicy*$(w_i, \delta_{N_i}, y_t, C_{loc,i}(t))$
  $z_{new}^{\phi_i} = \beta z_t^{\phi_i} + \frac{\nabla f_i(g_{t+1} | \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})}{f_i(g_{t+1} | \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})}$
  $z_{new}^{\theta_i} = \beta z_t^{\theta_i} + \frac{\nabla \mu_i(a_i(t) | \theta_i, g_{t+1}, y_t, \delta_{N_i})}{\mu_i(a_i(t) | \theta_i, g_{t+1}, y_t, \delta_{N_i})}$
  $\triangle_{new}^{\phi_i} = \triangle_t^{\phi_i} + \frac{1}{t+1} \left[C_{loc,i}(t) z_{new}^{\phi_i} - \triangle_t^{\phi_i} \right]$
  $\triangle_{new}^{\theta_i} = \triangle_t^{\theta_i} + \frac{1}{t+1} \left[C_{loc,i}(t) z_{new}^{\theta_i} - \triangle_t^{\theta_i} \right]$
  $\phi_{i,new} = \phi_i - \alpha_{t+1} \triangle_{new}^{\phi_i}$
  $\theta_{i,new} = \theta_i - \alpha_{t+1} \triangle_{new}^{\theta_i}$
  Return $w_i^* = \langle \phi_{i,new}, \theta_{i,new} \rangle$
End Function

for the current context. Agent i is allowed to improve its policy if its gain message is larger than all the gain messages it receives from all its neighbors. Essentially, agent i changes its policy to the computed best local policy if it is the winner at the current context or *cycle* of the algorithm. This process is then repeated.

The idea of exchanging policies and gain messages in the LID-RLPS algorithm to improve agent i's policy with respect to its neighbors' policies in a distributed manner is based on the *Distributed Breakout Algorithm* (DBA) for DCOPs [6], [16]. However, LID-RLPS includes planning under uncertainty, where the value of the local neighborhood utility depends on the *expected* long term value, whereas DBA does not handle uncertainty in the variables of a DCOP problem.

C. Algorithm Implementation Issues

The function *LocalBestPolicy*$(w_i, \delta_{N_i}, y_t, C_{loc,i}(t))$ in Algorithm 1 returns the best response policy: $w_i^* = \arg\min_{w_i} B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$. This function is implemented as a *policy gradient* algorithm similar to the model-free *IState-GPOMDP* algorithm proposed in [9]. The pseudo-code for *LocalBestPolicy*(w_i, δ_{N_i}) is shown in Algorithm 2. The proof of convergence is omitted for brevity.

The function *GetEstimate*(w_i, δ_{N_i}) in Algorithm 1 computes the estimated local neighborhood utility $B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$ given the neighbors' policies. If we simply follow the *IState-GPOMDP* algorithm in [9], this estimate is obtained from the immediate localized cost $C_{loc,i}(t)$ as follows: Let $\eta_{B_i}(t)$ be the current estimate of $B(N_i, \phi_i, \theta_i | \overrightarrow{w_{N_i}})$ given the *current* neighbor policies $\overrightarrow{w_{N_i}}$. It is then updated as follows:

$$\eta_{B_i}(t+1) = \eta_{B_i}(t) + \frac{1}{t+1} \left[C_{loc,i}(t+1) - \eta_{B_i}(t) \right] \qquad (13)$$

However, this update structure may not be suitable in storing the estimates of *every* possible policies of neighbors $\overrightarrow{w_{N_i}}$. This is due to the fact that δ_{N_i} is obtained from the set of neighbor parameters $\langle \theta_k, \phi_k \rangle\ \forall k \in N_i$, which are from *continuous* vector spaces. In this paper, we propose that the estimated

utility $\eta_{B_i}(t)$ is approximated using a form of *neural network*, known as *Cerebellar Model Articulation Controller* (CMAC).

A CMAC is a tile-coding structure that performs linear function approximation, where the output (i.e. $\eta_{B_i}(t)$) is a weighted linear sum of the features of the input vector parameters. The input vector for the CMAC is represented as $[g_{t+1}, y_t]$. The CMAC internal neural network weights are represented by the vector parameters $\langle \delta_{N_i}, \theta_i, \phi_i \rangle$. The CMAC displays *local generalization* for approximating the estimated utility. With this structure, different policies of neighbors are represented compactly and the estimate $\eta_{B_i}(t)$ is retrieved easily. The CMAC neural network weights are first initialized to zero, and are then updated with the immediate localized cost $C_{loc,i}(t)$ as the target value. More details on CMAC networks can be found in [18].

In Algorithm 1, the vector parameter δ_{N_i} is obtained from the policy vectors of the neighbors $\{N_i\}$. In our experiments, δ_{N_i} is computed where its vector elements are the *average* of the corresponding elements in the policy vectors: $\langle \theta_k, \phi_k \rangle$ $\forall k \in \{N_i\}$. Other possible *feature* representation for δ_{N_i} can also be investigated.

In representing the MFSC distribution functions $f_i(g_{t+1} \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$ and $\mu_i(a_i(t) \mid \theta_i, g_{t+1}, y_t, \delta_{N_i})$, we face the same issue for storing the function values for every possible neighbor policies, since the vector parameter δ_{N_i} is a continuous real-valued vector. Hence, we use neural networks to represent the these distribution functions. Following [9], we use the *soft-max* function to generate these distributions.

Specifically, for $f_i(\cdot \mid \phi_i, g_t, y_t, a_i(t-1), \delta_{N_i})$, a neural network is used where the input vector is represented as $[g_t, y_t, a_i(t-1)]$, which is a concatenation of the current I-state g_t, current observation y_t, and previous action $a_i(t-1)$. The weights of the neural network are represented by the parameter vectors $\langle \delta_{N_i}, \phi_i \rangle$. For the sake of brevity, we omit the update equations (see [9]) for computing the gradient as required in Algorithm 2. Basically, the update expression is implemented similar to error back propagation, which is a standard procedure for training neural networks [18]. However, instead of propagating the gradient of an error measure, we back propagate the soft-max gradient for the agent's choice of the next I-state. We derive $\frac{\nabla \mu_i(a_i(t)|\theta_i, g_{t+1}, y_t, \delta_{N_i})}{\mu_i(a_i(t)|\theta_i, g_{t+1}, y_t, \delta_{N_i})}$ in Algorithm 2 in the same way by having another neural network with input as $[g_{t+1}, y_t]$ and evaluating the soft-max distribution for each possible action $a_t(t)$ by using the real-valued outputs of the neural network.

IV. Simulation and Discussion

In this section, we study the performance of the proposed model-free algorithm known as LID-RLPS in the context of packet scheduling in wireless multi-hop networks.

A wireless multi-class network of 20 mobile nodes in a 1,000m by 1,000m area is simulated in the NS2 simulator [7]. We use the IEEE 802.11 Distributed Coordination Function (DCF) for the MAC. For the routing protocol, the Ad hoc On-Demand Distance Vector (AODV) protocol is used. A two-ray ground reflection model is used for the radio propagation model. The nodes are simulated with a speed of 0 to 10m/s with a random way-point mobility model and varying pause times. The simulation is done for 3,000 seconds.

We define three traffic classes and simulate eight long-lived Constant Bit Rate (CBR) connections with the characteristics shown in Table I. We choose CBR flows since this type of flows captures the worst case and average long term performance. Data packets are marked as Class I, II or III, where Class I packets receive the highest priority and experience the lowest delay. The control packets from the routing protocol are marked as Class I and the data packet size is 64 bytes.

TABLE I
Traffic Source Characteristics

Traffic Source	Traffic Class	Rate (kbps)
Nodes 1 & 2	I	128
Nodes 3 & 4	III	32
Nodes 5 & 6	II	100
Nodes 7 & 8	III	128

The DEC-POMDP formulation for this simulation scenario is similar to the earlier formulation in Section II-C. However, the global state descriptor is defined as:

$$S(t) := \overrightarrow{x}(t) = \{x_1^j(t), .., x_N^j(t)\},\ \forall j = 1, 2, 3$$

The global state is thus the concatenation of the queue length or congestion level of all traffic classes in all nodes. Each agent i executes its policy by allocating bandwidth to its local class queues. The immediate cost function is defined as:

$$C(\overrightarrow{x}(t), \overrightarrow{a}(t)) = \sum_{i=1}^{N} \sum_{j=1}^{3} x_i^j(t) \tag{14}$$

Starting with any global state $S(0) := s_0 = \overrightarrow{x}(0)$, the main goal is to find the joint optimal scheduling policy $\overrightarrow{w}$ for the 20 mobile nodes that minimizes the average congestion level defined as:

$$J(\overrightarrow{w}, s_0) := \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_{s_0}^{\overrightarrow{w}} \{C(\overrightarrow{x}(t), \overrightarrow{a}(t))\} \tag{15}$$

We compare the performance of LID-RLPS with a single-agent based RL algorithm similar to [19]. We refer to the latter as Independent RL Provisioning (IRLP), where each node only considers its own locally-observed MDP independently and does not communicate its policies with any other agent.

Figure 2 shows the normalized average long term cost or congestion level under different scenarios and pause times. As expected, IRLP incurs higher congestion level even though each agent learns and adapts its policy. LID-RLPS achieves significantly lower congestion level since each agent coordinates among its neighbors before the winner agent changes its policy, as explained in Algorithm 1.

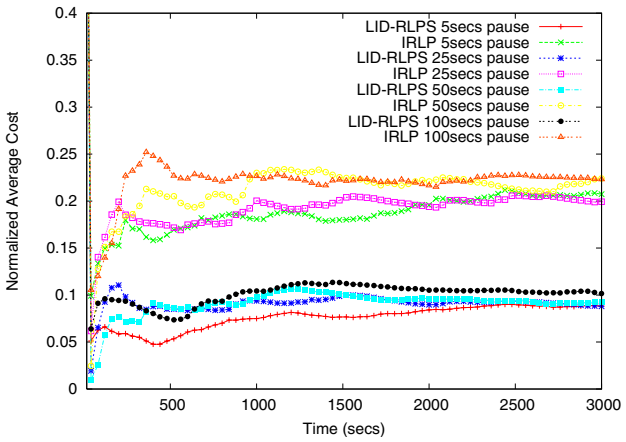

Fig. 2. Normalized average cost under varying pause times

V. Conclusion

We have considered the problem of decentralized stochastic control in a multi-agent system under the DEC-POMDP framework. Due to the non-trivial complexities of solving a DEC-POMDP, we have proposed a model-free algorithm known as LID-RLPS that performs a cooperative decentralized optimization. LID-RLPS employs a multi-agent finite state controller together with the concepts of locality of interaction and local neighborhood utility, to approximate the joint optimal policy of the agents.

In the context of resource allocation and scheduling for a communication network, simulation results have shown that the proposed scheme is able to attain its objective of optimizing the average long term cost, as compared to independent agents. To the best of the authors' knowledge, our proposed algorithm is the first attempt to use a coordinated RL mechanism to solve a multi-agent DEC-POMDP, while considering the locality of interaction of agents. For future work, we intend to apply the proposed algorithm in other applications and study its convergence properties.

References

[1] M. L. Puterman, *Markov Decision Processes*. New York, USA: Wiley Interscience, 1994.

[2] C. Boutilier, "Sequential optimality and coordination in multiagent systems," in *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999, pp. 478–485.

[3] L. Peshkin, K.-E. Kim, N. Meuleau, and L. Kaelbling, "Learning to cooperate via policy search," in *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, 2000, pp. 489–496.

[4] D. Pynadath and M. Tambe, "The communicative multiagent team decision problem: Analyzing teamwork theories and models," *Journal of Artificial Intelligence Research*, vol. 16, pp. 389–423, 2002.

[5] D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The Complexity of Decentralized Control of Markov Decision Process," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.

[6] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, "Networked distributed POMDPs: A synthesis of Distributed Constraint Optimization and POMDPs," in *Proc. of the 20th National Conference on Artificial Intelligence*, 2005, pp. 133–139.

[7] K. Fall and K. Varadhan, editors. The ns manual. The VINT Project, UC Berkeley, LBL, USC/ISI and XEROX PARC. [Online]. Available: http://www.isi.edu/nsnam/ns/ns-documentation.html

[8] J. Baxter and P. Bartlett, "Infinite-Horizon Policy Gradient Estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[9] D. Aberdeen, "Policy-Gradient Algorithms for Partially Observable Markov Decision Process," Ph.D. dissertation, Australian National University, Australia, Apr. 2003.

[10] A. Cassandra, M. Littman, and N. Zhang, "Incremental Pruning: A simple, fast, exact method for Partially Observable Markov Decision Process," in *Proc. of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, San Francisco, CA, 1997, pp. 54–61.

[11] T. Jaakkola, S. Singh, and M. Jordan, "Reinforcement Learning Algorithm for Partially Observable Markov Decision Process," in *Advances in Neural Information Processing Systems*, vol. 7, 1995, pp. 345–352.

[12] E. Hansen, D. Bernstein, and S. Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games," in *Proc. of the 19th National Conference on Artificial Intelligence*, 2004, pp. 709–715.

[13] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella, "Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings," in *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

[14] D. Aberdeen and J. Baxter, "Scaling Internal-State Policy-Gradient Methods for POMDPs," in *Proc. 19th International Conf. on Machine Learning*, Sydney, Australia, July 2002, pp. 1–12.

[15] B. Bonet, "An ϵ-optimal grid-based algorithm for Partially Observable Markov Decision Processes," in *Proc. 19th International Conf. on Machine Learning*, C. Sammut and A. Hoffmann, Eds., Sydney, Australia, 2002, pp. 51–58.

[16] M. Yokoo and K. Hirayama, "Distributed breakout algorithm for sovling distributed constraint satisfaction problems," in *Proc. ICMAS*, 1996, pp. 401–408.

[17] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo, "An asynchronous complete method in distributed constraint optimization," in *Proc. 2nd International Conf. on Autonmous Agents and Multi-Agent Systems*, Sydney, Australia, 2003.

[18] C. K. Tham, "Online Function Approximation for Scaling Up Reinforcement Learning," Ph.D. dissertation, Univ. of Cambridge, UK, Oct. 1994.

[19] C. Pandana and K. J. R. Liu, "Near-Optimal Reinforcement Learning Framework for Energy-Aware Sensor Communications," *IEEE J. Select. Areas Commun.*, vol. 23, no. 4, Apr. 2005.