

Distributed Coordination Guidance in Multi-Agent Reinforcement Learning

Qiangfeng Peter Lau, Mong Li Lee and Wynne Hsu
Department of Computer Science, National University of Singapore
 {plau, leeml, whsu}@comp.nus.edu.sg

Abstract—In this paper we present a distributed reinforcement learning system that leverages on expert coordination knowledge to improve learning in multi-agent problems. We focus on the scenario where agents can communicate with their neighbors but this communication structure and the number of agents may change over time. We express coordination knowledge as constraints to reduce the joint action space for exploration. We introduce an extra learning level to learn when to make use of these constraints. This extra level is decentralized among the agents, making it suitable for our communication restrictions. Experiment results on tactical real-time strategy and soccer games show that our system is effective in online learning as opposed to existing methods that use individual constraints on agents and coordinated action selection.

Keywords—learning; coordination; guiding exploration;

I. INTRODUCTION

Learning to make sequential decisions for multiple collaborating agents is a difficult problem in general. This is especially so when agents require coordination to achieve a common goal. The space of their action combinations is exponential in the number of agents, quickly rendering reinforcement learning (RL) with naive exploration impractical. For online RL to be viable, we need to optimize the number of interactions with the environment. This motivates us to develop a solution to improve the learning rate of good policies in multi-agent RL (MARL).

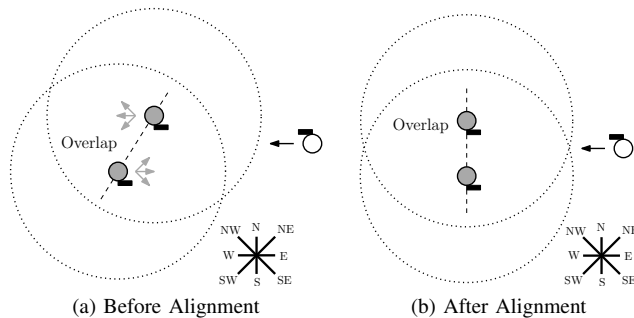


Figure 1. Simplified states of tactical real time strategy game. Solid circles represent marines (2 for gray team and 1 for white team). Dotted circles indicate the range of their rifle (black bar). Black arrow indicates movement direction for white marine. Dashed line shows alignment of gray marines. Gray arrows in (a) are movement actions that may result in the state in (b).

Consider the simplified view of states from a tactical real time strategy (RTS) game [1] in Fig. 1. Two marines from the gray team are shown with an oncoming enemy marine from the white team. The objective is to destroy the enemy.

Each marine may move in 8 compass directions as shown in the figure or stay put and shoot at the enemies within range of its rifle. In Fig. 1(a), the joint action space of the gray team is $9^2 = 81$. Careful examination reveals that much of this space is of less importance for exploration as they may not quickly lead to the goal of destroying the enemy. For example, the gray marines should not move in such a way that prevents both from shooting at the enemy at the same time. An ideal situation is illustrated in Fig. 1(b) where the overlapped shooting range is aligned to the enemy’s approach. Once the shooting begins, the white marine may only shoot at one gray marine while both gray marines can shoot at it. With this simple coordination strategy, the joint action space in Fig. 1(a) is reduced by 81% to that of $\{NW, W, SW, stay\} \times \{NE, E, SE, stay\} - \{(stay, stay)\}$ which has a size of 15.

In this work, we aim to exploit coordination knowledge to improve the learning rate of useful policies by modeling coordination among agents as hard constraints. We refer to these hard constraints as Coordination Constraints (CCs), and use CCs to limit the joint action space for exploration. Unary constraints defined on single agents are a special case of CCs.

Previous works on MARL have assumed varying degrees of communication among agents. One approach for directing exploration is to incorporate the concept of procedural tasks from single agent hierarchical RL [2]–[4] into a multi-agent setting. At each time step, agents are individually constrained to the actions allowed for the current task they are assigned to. Task assignment may be part of the learning process [5], [6] or derived separately [7]. However, the multi-agent coordination expressed by CCs do not fit well within procedural task definitions. Existing works usually delegate such coordination knowledge as basis functions (features) [5] or as static rules [7]. Another line of works uses static (fixed) heuristics to bias the policies of the original agent towards better exploration [8], [9]. Their communication structure is fixed at the start.

In contrast, we use a problem formulation where communication is limited to only between current neighboring agents. The number of agents and their communication structure may change dynamically according to the state. This corresponds to the scenario where agents have limited communication range and are physically mobile.

We propose a distributed two-level MARL system where each agent’s top level learns to select the appropriate CCs

to constrain its bottom level’s learning of joint actions with its neighbors. Selecting CCs is important as not all CCs are useful in every state. Deciding which CCs to employ in different states is part of the learning process, enabling the RL system to learn to guide itself during exploration. For example, in Fig. 1, the CC that indicates the gray marines movements should be constrained may not be suitable if one of the gray marines is badly wounded. It may make more sense for the healthy gray marine to engage the enemy first while the wounded marine supports from behind.

Developing this two-level MARL system entails three main challenges. First, since the communication between agents and the number of agents may change, the top level of the system must not critically reside in only selected agents. Second, the additional execution and exploration requirements of the top level must not hinder and should be beneficial to the overall learning rate. Third, learning at the top level should not detract from solving the original problem. We should make use of the original reward signal instead of defining new rewards that may not reflect the original goals of the system.

We address the first and third challenges by formulating decomposed update equations that take into account the communication restrictions for distributed local updates. For the second challenge, execution is handled via existing works in decentralized constraint optimization [10] in conjunction with the use of basis functions. Then, the exploration requirements for learning CC selections are managed by inspecting the semantics of CCs themselves. As an added benefit of our method, CC definitions may be used to provide a richer function approximation for learning.

To the best of our knowledge, this is the first distributed MARL system that learns to use coordination knowledge to guide learning under dynamic communication changes. The system is decentralized in the sense that learned parameters are split among the agents. Hence the loss of one agent does not render the system ineffective. The communication structure forms a coordination graph [11] among the agents so that they may coordinate if they can communicate. As such, the proposed approach is different from methods that learn when such coordination is (un)necessary [12].

The rest of the paper is organized as follows. Section 2 gives the problem formulation and background work. Section 3 describes the distributed two level system and how CCs are defined. Finally, we demonstrate our system’s effectiveness on RTS and simplified soccer game domains. Particularly for the former, we show that the system remains effective even when agents’ communication are lost.

II. PRELIMINARIES

For our MARL system, we make use of the decentralized Markov decision process (DEC-MDP) formulation [13] as follows. A DEC-MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ such that given N number of agents:

- 1) $\mathcal{S} = S_0 \times S_1 \times \dots \times S_N$ is the state space that is factored into state variable S_0 that is observable by all agents and S_1, \dots, S_N variables that are local for each agent. A state is $\mathbf{s} = \langle s_0, s_1, \dots, s_N \rangle \in \mathcal{S}$.
- 2) $\mathcal{A} = A_1 \times \dots \times A_N$ is the joint action space factored into N action variables, one for each agent. A joint action is $\mathbf{a} = \langle a_1, \dots, a_N \rangle \in \mathcal{A}$.
- 3) $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the global transitional probability model, i.e., $\mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = Pr(\mathbf{s}' | \mathbf{s}, \mathbf{a})$ where $\mathbf{s} \in \mathcal{S}$ is the current state, $\mathbf{a} \in \mathcal{A}$ is the current joint action and $\mathbf{s}' \in \mathcal{S}$ is the next state.
- 4) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the global reward function that gives the reward received for taking action \mathbf{a} in state \mathbf{s} and reaching state \mathbf{s}' . \mathcal{R} is decomposable into a sum of local rewards, $\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \sum_{n=1}^N R_n(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.

We assume that under different state values in \mathcal{S} , agents have different communication structure. That is, $\mathbf{s} \in \mathcal{S}$ defines a coordination graph (CG) such that a vertex exists for each agent and an edge exists between agents that can communicate in \mathbf{s} . Each agent, n , may identify its neighbors in the CG using state values $s_0 \in S_0$ and $s_n \in S_n$ and send messages to them. Hence agents may coordinate and access the local state variables of their neighbors. Based on this communication restrictions, we define \mathbf{s}_n to be the set of state variables accessible by agent n and \mathbf{a}_n to be the set of action variables of agents that are neighbors of n . Note that in general the CG may consists of disjoint sub-graphs and agents within the same sub-graph may synchronize their actions by sending messages.

A solution to the DEC-MDP is a global policy $\pi : \mathcal{S} \mapsto \mathcal{A}$. This policy can be expressed in terms of a global action value function that is the expected discounted sum of rewards, r , when taking \mathbf{a} in \mathbf{s} at time t and discount rate γ , i.e., $Q^\pi(\mathbf{s}, \mathbf{a}) = E_\pi \{ \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} | \mathbf{s}, \mathbf{a} \}$. This is usually written as the Bellman equation, $Q^\pi(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}') [\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q^\pi(\mathbf{s}', \pi(\mathbf{s}'))]$. Then, we may express the maximizing policy as $\pi(\mathbf{s}) = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}(\mathbf{s})} Q^\pi(\mathbf{s}, \mathbf{a})$, where $\mathcal{A}(\mathbf{s}) \subseteq \mathcal{A}$ indicates that certain actions may be disallowed in the current state. To learn an optimal policy π^* we need only to learn the optimal action value function Q^* directly, without actually learning the models \mathcal{T} and \mathcal{R} [14]. This is useful in the case of DEC-MDP as it frees the user from defining a decomposable transition model for \mathcal{T} which can be difficult.

For distribution, the global policy π needs to be expressed as local parts, one for each agent. This can be done by decomposing Q^π into a sum of local components [15], i.e., $Q^\pi(\mathbf{s}, \mathbf{a}) = \sum_{n=1}^N Q_n^\pi(\mathbf{s}_n, \mathbf{a}_n)$, where $\mathbf{s}_n, \mathbf{a}_n$ contain the state and action values of agent n and its current neighbors. Then, the global policy can be learned by updating local value functions Q_n^π using the local interactions with the environment (i.e. agent based decomposition in [10]) via, $Q_n^\pi(\mathbf{s}_n, \mathbf{a}_n) \leftarrow Q_n^\pi(\mathbf{s}_n, \mathbf{a}_n) + \alpha [r_n + \gamma Q_n^\pi(\mathbf{s}'_n, \mathbf{a}'_n) - Q_n^\pi(\mathbf{s}_n, \mathbf{a}_n)]$, where \mathbf{a}'_n is derived from the global joint

action \mathbf{a}' chosen from the global policy π , r_n is the reward observed by agent n , and α is the step size parameter. In our problem formulation, the CG is not fixed. This indicates that the state and action variables in $\mathbf{s}'_n, \mathbf{a}'_n$ may not correspond to those in the previous time step, $\mathbf{s}_n, \mathbf{a}_n$. We explain how this is handled as we describe our system next.

III. TWO LEVEL MARL SYSTEM

A conceptual overview of our proposed distributed coordination guided MARL (DistCGRL) system architecture is shown in Fig. 2 for four hypothetical agents, A_1, \dots, A_4 , that are able to communicate with their adjacent agents (double headed arrows). Each white box represents a learning component, generally grouped into global top and bottom components (dotted boxes). At each time step, each agent's top level component decides on the CCs to *activate* for the bottom level component. Then, the bottom level component takes into account the CCs in selecting the actual joint action $\langle a_1, a_2, a_3, a_4 \rangle$ to be taken in the environment. Both levels' components observe the next local state and local reward to update themselves internally.

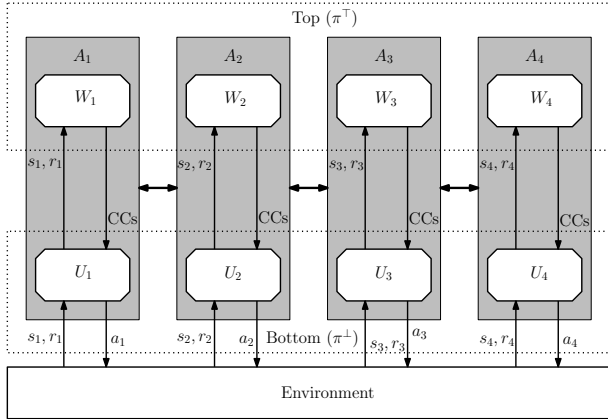


Figure 2. Conceptual architecture of two level DistCGRL system with four agents. Gray boxes are the agent boundaries. Within each agent, two white boxes indicate its learning components. Those at the top are grouped as the top level and similarly for the bottom. For each agent, top level components observe the same state and reward as the bottom level.

Our two level system makes use of coordination constraints (CCs) to improve the learning rate of a policy that solves the original DEC-MDP. Each of these CCs may involve a subset of the original number of agents. For example, the alignment movement constraint in Fig. 1 may be defined as one N -ary constraint involving all agents, or broken down into smaller pairwise constraints among agents. In general, the latter is preferred as it allows fine-grained control over the CCs to be employed in different states for various pairs of agents.

In order to make the learning system aware of the CCs, we augment the DEC-MDP with additional actions and hierarchical information in the following way. Let K be the

number of CCs. Then, there are K action variables such that $A_1^0 \times \dots \times A_K^0 = \mathcal{A}^0$, the joint action space for the top level of the system. Each A_k^0 corresponds to the k -th CC and consists of two actions: *activation* (1) and *deactivation* (0). We refer to a top level joint action as $\mathbf{b} \in \mathcal{A}^0$. Let Θ be a special indicator tuple of size K to denote that the system is in the top level. Then, the new state space augmented with the two level hierarchy is $\mathcal{S}^\Theta = (\mathcal{A}^0 \cup \{\Theta\}) \times \mathcal{S}$, and a state is $\langle \mathbf{b}^\Theta, \mathbf{s} \rangle \in \mathcal{S}^\Theta$. The solution to the DEC-MDP is a joint hierarchical policy, $\pi : \mathcal{S}^\Theta \mapsto \mathcal{A} \cup \mathcal{A}^0$, that can be represented by,

$$\pi(\langle \mathbf{b}^\Theta, \mathbf{s} \rangle) = \begin{cases} \pi^\top(\mathbf{s}) \in \mathcal{A}^0 & \text{if } \mathbf{b}^\Theta = \Theta \\ \pi^\perp(\mathbf{b}, \mathbf{s}) \in \mathcal{A} & \text{if } \mathbf{b}^\Theta = \mathbf{b} \end{cases} \quad (1)$$

where π^\top is a top policy that returns top level actions in \mathcal{A}^0 and π^\perp is a bottom policy that returns original primitive actions in \mathcal{A} from the DEC-MDP.

Next, we present learning equations for the augmented DEC-MDP.

A. Value Functions & Policy

To develop the learning equations, we first specify the global action value function Q^π . Subsequently, we decompose Q^π so that it is suitable for local updating. Similar to Eq. 1, we define Q^π in two parts,

$$Q^\pi(\langle \mathbf{b}^\Theta, \mathbf{s} \rangle, a) = \begin{cases} W(\mathbf{s}, a) & \text{if } \mathbf{b}^\Theta = \Theta, \text{ where } a \in \mathcal{A}^0 \\ U(\mathbf{s}, a) & \text{if } \mathbf{b}^\Theta \in \mathcal{A}^0, \text{ where } a \in \mathcal{A} \end{cases} \quad (2)$$

where W and U are the top and bottom level functions respectively, such that their Bellman equations are,

$$W(\mathbf{s}, \mathbf{b}) = \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{T}(\mathbf{s}, \chi, \mathbf{s}') [\mathcal{R}(\mathbf{s}, \chi, \mathbf{s}') + \gamma W(\mathbf{s}', \pi^\top(\mathbf{s}'))] \quad (3)$$

where $\chi = \pi^\perp(\mathbf{b}, \mathbf{s}) \in \mathcal{A}$, and,

$$U(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}') [\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma W(\mathbf{s}', \pi^\top(\mathbf{s}'))] \quad (4)$$

Eq. 3 handles the case when the augmented state is in the top level as indicated by Θ in $Q^\pi(\langle \Theta, \mathbf{s} \rangle, \mathbf{a}^0)$. It expresses a single evaluation of the hierarchical policy from the top into the bottom level for one primitive action, χ . Then the system returns to the top level and continues with the next action $\pi^\top(\mathbf{s}')$.

Eq. 4 is used when the augmented state is in the bottom level indicated by \mathbf{b} in $Q^\pi(\langle \mathbf{b}, \mathbf{s} \rangle, \mathbf{a})$. In this case, primitive action \mathbf{a} is taken and the system returns to the top level in the policy and continues from there with Eq. 3.

With Eq. 3 and 4, the top and bottom sub-policies can now be specified with respect to them. For example, in the case of greedy policies, $\pi^\top(\mathbf{s}) = \operatorname{argmax}_{\mathbf{b} \in \mathcal{A}^0(\mathbf{s})} W(\mathbf{s}, \mathbf{b})$ and $\pi^\perp(\mathbf{b}, \mathbf{s}) = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}(\mathbf{b}, \mathbf{s})} U(\mathbf{s}, \mathbf{a})$, where $\mathcal{A}^0(\mathbf{s}) \subseteq \mathcal{A}^0$ and $\mathcal{A}(\mathbf{b}, \mathbf{s}) \subseteq \mathcal{A}$ indicate that the joint action spaces they are maximizing may be constrained to a subspace. In particular, the bottom policy's $\mathcal{A}(\mathbf{b}, \mathbf{s})$ takes into account

the CCs activated by the top level specified by \mathbf{b} . Note that during the selection of exploratory actions, we select random actions, $\pi^\top(\mathbf{s}) \in \mathcal{A}^0(\mathbf{s})$ and $\pi^\perp(\mathbf{b}, \mathbf{s}) \in \mathcal{A}(\mathbf{b}, \mathbf{s})$.

To distribute the hierarchical policy, we decompose Eq. 3 and 4 into a sum of local components, one for each agent, giving $W(\mathbf{s}, \mathbf{b}) = \sum_{n=1}^N W_n(\mathbf{s}_n, \mathbf{b}_n)$ and $U(\mathbf{s}, \mathbf{a}) = \sum_{n=1}^N U_n(\mathbf{s}_n, \mathbf{a}_n)$ respectively, where \mathbf{s}_n contains the local state values of the state variables observable by agent n , \mathbf{b}_n is the local top level action corresponding to CCs that involve n and its neighbors, and \mathbf{a}_n is the local primitive action that involve n and its neighbors. The observable values and neighbors depend on the current CG. Fig. 2 depicts the local functions in the various components (white boxes) of the agents that together make up the joint top level and bottom level policy. Next we discuss these local states and actions and how they are involved in updating the local functions W_n and U_n .

B. Local Update Equations

For dynamic communication, the variables that make up \mathbf{s}_n , \mathbf{a}_n , and \mathbf{b}_n may change at each time step. For each agent, the joint values, \mathbf{s}_n and \mathbf{a}_n , contain values from the current neighbors that are defined based on the CG in the current state. Likewise, the action variables that represent CCs in the local top level action \mathbf{b}_n for agent n will only be present only if the agents that each CC refers to are current neighbors of the agent. For example in Fig. 1, suppose the variables in \mathbf{b}_n for each marine consists of unary CCs and the pairwise alignment CC with its neighbor. If the two marines move out of communication range, their top level actions will no longer include the CC for pairwise alignment.

Linear function approximation is widely used in RL to reduce the number of learning parameters as opposed to a tabular representation of value functions. It requires defined basis functions (BFs), ϕ_1, \dots, ϕ_J , and corresponding weights, w_1, \dots, w_J to be learned. A BF, ϕ_j is a function that returns a real number with a subset of the action and state variables as its domain. For example, for A_1 and A_2 in Fig. 2, we may have BF $\phi_j(s_1, s_2, a_1, a_2) \in \mathbb{R}$. For convenience, we write $\phi_j(\mathbf{s}, \mathbf{a})$ and we define $scope(\phi_j)$ to be the set of variables involved in the domain of ϕ_j . Let $\vec{\phi}_{\mathbf{s}, \mathbf{a}} = \langle \phi_1(\mathbf{s}, \mathbf{a}), \dots, \phi_J(\mathbf{s}, \mathbf{a}) \rangle$ and $\vec{w} = \langle w_1, \dots, w_J \rangle$. Then, the approximation for a function F is the dot product, $F(\mathbf{s}, \mathbf{a}) \approx \vec{w} \cdot \vec{\phi}_{\mathbf{s}, \mathbf{a}}$.

Now, we update each local value function with the local temporal difference error. We define the local update equations for each $W_n(\mathbf{s}_n, \mathbf{b}_n) \approx \vec{w}_{W_n} \cdot \vec{\phi}_{W, \mathbf{s}_n, \mathbf{b}_n}$ and $U_n(\mathbf{s}_n, \mathbf{a}_n) \approx \vec{w}_{U_n} \cdot \vec{\phi}_{U, \mathbf{s}_n, \mathbf{a}_n}$ as,

$$\vec{w}_{W_n} \leftarrow \vec{w}_{W_n} + \alpha [r_n + \gamma W_n(\mathbf{s}'_n, \mathbf{b}'_n) - W_n(\mathbf{s}_n, \mathbf{b}_n)] \vec{\phi}_{W, \mathbf{s}_n, \mathbf{b}_n} \quad (5)$$

$$\vec{w}_{U_n} \leftarrow \vec{w}_{U_n} + \alpha [r_n + \gamma W_n(\mathbf{s}'_n, \mathbf{b}'_n) - U_n(\mathbf{s}_n, \mathbf{a}_n)] \vec{\phi}_{U, \mathbf{s}_n, \mathbf{a}_n} \quad (6)$$

where \mathbf{b}'_n is the top level joint action of agent n and its neighbors derived from the global top level action, $\mathbf{b}' = \pi^\top(\mathbf{s}')$. We use ϵ -greedy policies for π^\top and π^\perp where an exploratory action is chosen with ϵ probability and a maximal action with $1 - \epsilon$ probability. These policies are GLIE when ϵ decreases over time [3]. Note that we have not used the primitive action \mathbf{a}' selected for \mathbf{s}' by the policy as W_n already encodes the current local estimate of the expected return.

The variables whence \mathbf{s}_n , \mathbf{a}_n , and \mathbf{b}_n come from may be different in two consecutive states as the agents' neighbors may have changed. To handle this problem of dynamic communication changes, we only require each BF ϕ_j to return *zero* when there exists variables in the $scope(\phi_j)$ that do not have corresponding values present in \mathbf{s}_n , \mathbf{a}_n , or \mathbf{b}_n . Hence, the weights of BFs in Eq. 5 and 6 that involve more than one agent will only be updated for the duration in which the agents remain as neighbors.

C. Action Selection

We now examine the finer details of the global policies π^\top and π^\perp . In particular, the primitive action selection for π^\perp is constrained to $\pi^\perp(\mathbf{b}, \mathbf{s}) \in \mathcal{A}(\mathbf{b}, \mathbf{s}) \subseteq \mathcal{A}$ where $\mathbf{b} = \langle b_1, \dots, b_K \rangle \in \mathcal{A}^0$. The values in \mathbf{b} represent if a CC is activated (1) or not (0). We can formulate choosing a maximum action in the constrained space $\mathcal{A}(\mathbf{b}, \mathbf{s})$ as a constraint optimization problem (COP) over the original space \mathcal{A} by specifying the objective function,

$$OBJ_{\mathbf{b}}(\mathbf{s}, \mathbf{a}) = U(\mathbf{s}, \mathbf{a}) + \sum_{k=1}^K b_k \cdot c_k(\mathbf{s}, \mathbf{a}) \quad (7)$$

where each c_k is the corresponding CC that returns $-\infty$ if the CC is violated and 0 otherwise. If the CC is not activated in \mathbf{b} , $b_k = 0$ and it has no impact on $OBJ_{\mathbf{b}}$. The bottom policy is then, $\pi^\perp(\mathbf{b}, \mathbf{s}) = \arg\max_{\mathbf{a} \in \mathcal{A}} OBJ_{\mathbf{b}}(\mathbf{s}, \mathbf{a})$ over the original space \mathcal{A} . To select an exploratory random action within $\mathcal{A}(\mathbf{b}, \mathbf{s})$, we replace U with a random function.

COPs using the the local functions U_n can be solved by distributed COP (DCOP) solvers. This typically involves message passing among neighbors. If each c_k and U_n can be additively decomposed into components involving up to two agents in their scope, the max-plus algorithm can be used directly on the current CG [10]. Otherwise, for components involving more agents, the max-sum method can be used [16]. These methods are approximate for CG with cycles, hence some CCs may not be enforceable. Nevertheless they have been shown to be effective in practice.

Next, we deal with the question of who makes the decisions for CCs that involve multiple agents. Consider Fig. 3 where each agent is represented as vertex in the original CG. Suppose that each edge corresponds to a pairwise CC. Then, each vertex in the top level CG corresponds to the top level action variable for one CC. The edges in the top level

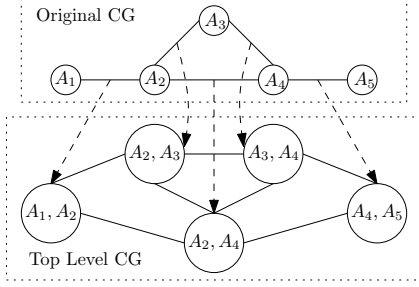


Figure 3. Example top level coordination graph derived from the original.

CG indicates that there exists some pairwise BF of the top level function W that involves two CCs. Hence a similar objective function to Eq. 7 can be used to compute the top level action, $\pi^\top(\mathbf{s})$.

For each vertex of the top level CG, we use a scheme where one predetermined agent in the vertex computes the activation of the CC while other the agents relay their local values of W_n that is relevant to the CC as well as messages from their neighboring agents. With this scheme, the described DCOP solvers can be used to compute $\pi^\top(\mathbf{s})$ using local W_n values. Note that our requirement that only neighboring agents may communicate still holds. If no component in W exists that is defined on multiple top level action variables, the top level CG has no edges and the activation of each CC can be decided independently by each group of involved agents. This turns out to be effective as shown by the experiment results in Section IV.

D. Basis Functions & Constraints

Predicates are commonly used to define BFs for the value functions [5], [12], [17]. They can be used directly as binary BFs that return 1 if the predicate is true and 0 otherwise. Because predicates are logical encodings of domain knowledge, we can use them to encode CCs as well. This allows us to make use of existing predicate BFs (PBFs) to encode CCs, and conversely, to allow newly specified CCs to enrich the existing set of PBFs for the U_n functions.

The sharing of predicates allows us to make use of PBFs in the following way. Given a PBF, ϕ_k , we can use it as a CC, c_k , by having, $c_k(\mathbf{s}, \mathbf{a}) = -\infty \cdot \phi_k(\mathbf{s}, \mathbf{a})$. An immediate benefit of this definition is that given activations \mathbf{b}_n for each agent, CCs can be represented in the objective function (Eq. 7) merely by setting the corresponding weight w_k for some ϕ_k used in U_n to the value of $-\infty$ before action selection by π^\perp . Then, the global OBJ_b (Eq. 7) becomes,

$$\sum_{n=1}^N \left[\underbrace{\sum_j w_j \cdot \phi_j(\mathbf{s}_n, \mathbf{a}_n)}_{\text{Normal BFs \& deactivated CCs}} + \underbrace{\sum_{k \in \mathbf{b}_n} -\infty \cdot \phi_k(\mathbf{s}_n, \mathbf{a}_n)}_{\text{CCs activated in } \mathbf{b}_n} \right] \quad (8)$$

U_n , the local function of agent n

After selecting \mathbf{a} , we restore the original value of w_j during updates (Eq. 5 and 6) so that weights can be updated in the

event that the DCOP solver fails to enforce the CC.

Note that if a CC is violated, its predicate is true. To illustrate, we give an example definition of the alignment CC for two marines in Fig. 1 below,

$$\begin{aligned} \text{NotAligned}(\mathbf{s}_1, \mathbf{s}_2, a_1, a_2) = & \text{SameNearestEnemy}(\mathbf{s}_1, \mathbf{s}_2) \\ & \wedge \text{Distance}\Delta(\mathbf{s}_1, a_1, \mathbf{s}_2, a_2) > 0 \end{aligned}$$

where SameNearestEnemy is true if the two marines' nearest enemy is the same. Let the nearest enemy be E . Then, $\text{Distance}\Delta$ returns the magnitude of the difference in distance between pairs of marines $\langle A_1, E \rangle$, and $\langle A_2, E \rangle$, after both marines have each taken actions a_1 and a_2 in \mathbf{s} . Now, the predicate NotAligned can be used as a PBF for local functions U_1 and U_2 , and also as a CC for both the marines as it will be true if the marines move further out of alignment.

Note that the NotAligned CC defines a constraint that state what should not be. This type of expert knowledge is able to remove large parts of the primitive action space from exploration in certain states, but the learning process maintains the flexibility of figuring out good solutions.

PBFs for the top level functions W_n can also be based on predicates that are used for U_n . For example, each marine has a number of hit points that reduces when damaged by the enemy, and let $\text{NotAligned}_{1,2}$ be the NotAligned CC for marines A_1 and A_2 . Then, top level PBFs may be $\text{Wounded}(\mathbf{s}_1) \wedge \text{Activated}(\text{NotAligned}_{1,2})$ and $\text{Wounded}(\mathbf{s}_2) \wedge \text{Activated}(\text{NotAligned}_{1,2})$ for W_1 and W_2 respectively, where $\text{Wounded}(s_x)$ is true if marine x has less than half its hit points remaining and $\text{Activated}(c)$ is true if the top level activates c . Then, the MARL system can now dynamically learn if it is desirable to activate the NotAligned CC when one of the marines is wounded.

E. Improving Top Level Learning Efficiency

We have introduced top level actions that can be used to improve the exploration of primitive actions in the DEC-MDP. However, the top level actions' exploration should not hinder the overall learning rate. In practice, efficient exploration of CC activations can be achieved by inspecting the semantics of CCs themselves. This typically yields a smaller top level action space $\mathcal{A}^0(\mathbf{s}) \subseteq \mathcal{A}^0$ to explore. For example, for the NotAligned CC, it can only be true if both marines have the same nearest enemy. Hence, if SameNearestEnemy is false, the NotAligned can immediately be deactivated as it is impossible to violate the CC. Furthermore, there is no activation of CCs between agents that are currently out of communicable distance. Each agent may determine which CCs can be deactivated in time linear in number of CCs that it is involved in.

F. Agent Execution

Finally, we summarize our DistCGRL system in the form of an algorithm (Fig. 4) that each agent executes. This is a

form of on-policy temporal difference learning. Neighboring agents communicate their local values of W_n and U_n during action selection in Lines 2, 5, 9, and 12. Then, they may observe their own and their neighbors’ resultant actions of π^\top and π^\perp for local updating at Lines 13–16. The updates involve the locally observed reward r_n . Local updates may be computed in parallel by the agents. As mentioned in Section III-D, \vec{w}_{U_n} is backed up and set to $-\infty$ for those CCs activated by \mathbf{b}_n (see Lines 3 and 10). But they are restored before updating the value functions at Line 15.

```

1: Observe initial state  $\mathbf{s}_n$ 
2:  $\mathbf{b}_n \leftarrow \pi^\top(\mathbf{s})$            {select top action, communicate}
3:  $\vec{w}'_{U_n} \leftarrow \vec{w}_{U_n}$        {backup weights}
4: Set weights in  $\vec{w}_{U_n}$  to  $-\infty$  if activated in  $\mathbf{b}'_n$ .
5:  $\mathbf{a}_n \leftarrow \pi^\perp(\mathbf{b}, \mathbf{s})$    {select primitive action, communicate}
6:  $\vec{w}_{U_n} \leftarrow \vec{w}'_{U_n}$        {restore weights}
7: while  $\neg \text{terminal}(\mathbf{s}_n)$  do
8:   Take action  $a_n$ , observe reward  $r_n$  and state  $\mathbf{s}'_n$ 
9:    $\mathbf{b}'_n \leftarrow \pi^\top(\mathbf{s}')_n$    {select top action, communicate}
10:   $\vec{w}'_{U_n} \leftarrow \vec{w}_{U_n}$        {backup weights}
11:  Set weights in  $\vec{w}_{U_n}$  to  $-\infty$  if activated in  $\mathbf{b}'_n$ .
12:   $\mathbf{a}'_n \leftarrow \pi^\perp(\mathbf{b}'_n, \mathbf{s}')_n$  {select primitive action, communicate}
13:   $q \leftarrow r_n + \gamma W_n(\mathbf{s}'_n, \mathbf{b}'_n)$  {local agent updates}
14:   $\vec{w}_{W_n} \leftarrow \vec{w}_{W_n} + \alpha[q - W_n(\mathbf{s}_n, \mathbf{b}_n)]\phi_{W, \mathbf{s}_n, \mathbf{b}_n}$ 
15:   $\vec{w}_{U_n} \leftarrow \vec{w}'_{U_n}$        {restore weights}
16:   $\vec{w}_{U_n} \leftarrow \vec{w}_{U_n} + \alpha[q - U_n(\mathbf{s}_n, \mathbf{a}_n)]\phi_{U, \mathbf{s}_n, \mathbf{a}_n}$ 
17:   $\mathbf{s}_n \leftarrow \mathbf{s}'_n$ ;  $\mathbf{b}_n \leftarrow \mathbf{b}'_n$ ;  $\mathbf{a}_n \leftarrow \mathbf{a}'_n$ 
18: end while

```

Figure 4. DistCGRL algorithm for a particular agent, n . If the episode has ended or agent n is removed from the episode, \mathbf{s}_n is terminal.

Note that in the case whereby agents split into two or more disjoint sub-CGs, the selection of actions by π^\top and π^\perp will only involve the variables and functions that are present in each sub-CG respectively. Similarly if agents are removed, e.g., when marines are destroyed in RTS, they no longer perform updates or participate in selecting actions. The system has no learned values that critically reside in any one agent as every agent carries its share of the top and bottom value functions.

IV. EXPERIMENT RESULTS

We carry out experiments to evaluate our proposed DistCGRL approach on two domains: tactical RTS [1] and simplified soccer. All learning is online as no experience is saved and replayed. We compare the performance of the following distributed RL players:

- 1) *Independent* player. Each agent uses the independent learners approach [18] by selecting their own action and learning their policy independently.
- 2) *Coordinated* player. Adapted from the agent based decomposition [10], this learner uses on-policy learning with the coordination graph (CG) for joint action selection among neighbors.

- 3) *DistCGRL-Solo* player. Only unary constraints are used in DistCGRL. This is a representative of having individual task hierarchies for each agent. Tasks are combinations of the unary constraints that lasts for one step.
- 4) *DistCGRL-Full* player. This player uses our full two-level learning system with all CCs.

The DistCGRL players use the same BFs as the coordinated player for their bottom level value function. Communication structure (i.e. CGs) changes over time within each episode. The independent player has only BFs that involve one agent. We have used the max-plus algorithm for action selection. For each top and primitive action selection, max-plus was limited to 10 iterations.

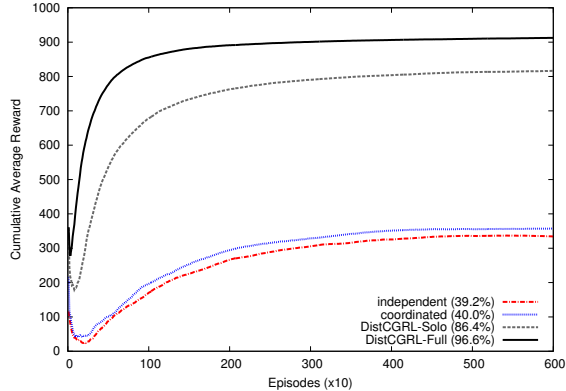
A. Tactical RTS

The goal in tactical RTS [1] is to eliminate the enemy team quickly in a 240×240 *point based* map. Each marine occupies a point on the map with an initial number of hit points. It is destroyed when its hit points reaches zero. A marine’s action domain consists of the 8 compass directions, an attack action for each possible enemy, and idle. The size of the action space is at least 10^{10} while the state space consists of all the possible marines’ positions and hit points. Marines can only communicate if they are within 30 points of each other and cannot communicate for the rest of the episode once destroyed. Marines can sense all enemies, and have access to some summary state variables (e.g. average hit points) of their team, while actual hit points are local to each marine.

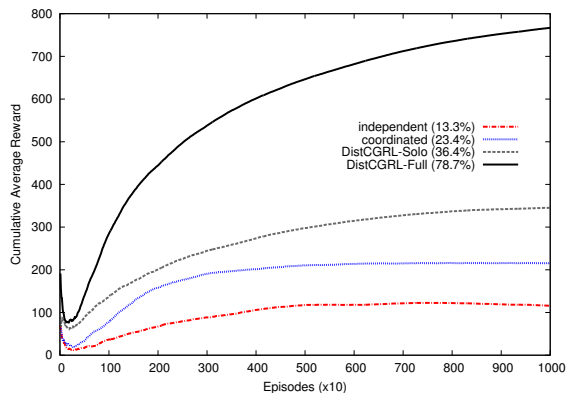
We pit the RL players against two scripted opponents: *unpredictable* marines move in random directions and shoot enemies in range, *aggressive* marines head for the nearest enemy and shoot enemies in range. The unpredictable opponent may be strong if its marines move in the same direction towards the enemy, or weak if they scatter. Opponents’ marines are able to shoot and move at the same time giving them an advantage. RL players must quickly learn to shoot and exploit teamwork as marines die easily. This makes it difficult to explore winning episodes. We use two setups in our experiments: (a) 10 RL marines versus 10 unpredictable marines, (b) 10 RL marines versus 10 aggressive marines.

Global rewards are -0.1 per time step and 10^3 for eliminating the opposing team. They are equally divided among the current number of surviving marines. For each setup, the RL players use $\gamma = 1$ with optional decaying step size (α) and exploration (ϵ) parameters written as $param = \{\text{initial, final, decay rate}\}$. Both RTS setups use $\epsilon = \{0.5, 0.01, 0.998\}$ and constant $\alpha = 10^{-4}$. DistCGRL-Full uses 90 pairwise CCs for troop formation to maximize overlapping firepower on enemies and to protect weaker team members. DistCGRL-Solo has these as BFs.

The RTS results are shown in Fig. 5. We show cumulative average reward as no discounting is used, and in the online



(a) Unpredictable Opponent Team (6,000 episodes)



(b) Aggressive Opponent Team (10,000 episodes)

Figure 5. RTS results averaged over 10 runs each. Final win rates for all episodes shown in brackets.

setting, we wish to maximize rewards even while learning. For both types of opponents, all RL players are able to learn and converge. All DistCGRL players learn more efficiently than the independent and coordinated players, demonstrating that our local updating methods are stable and effective. In particular, the DistCGRL-Full player has the best performance and starts out with high reward for both experiment setups. This shows that allowing the system to dynamically learn to employ CCs derived from coordination is effective for good performance as opposed to only having unary CCs.

B. Simplified Soccer

Fig. 6 shows an example state in simplified soccer. The goal is to score first in the shortest time. The soccer field is a 15×10 grid world. Soccer players can stay, move in 4 directions, or the player with the ball may pass or shoot with a probability of success weighted by distance. The ball changes ownership to the opposing team if the player with the ball collides with any player or if a pass fails. A failure to score a goal results in the ball going to the nearest player to the goal. Communication is only allowed for a Manhattan distance of 5 grid squares. The size of the action space is at least 6^8 . Only the player that scores a goal receives a reward of 1, and when defeated, the reward of -1 is equally divided

among soccer players that are nearest to the home goal.

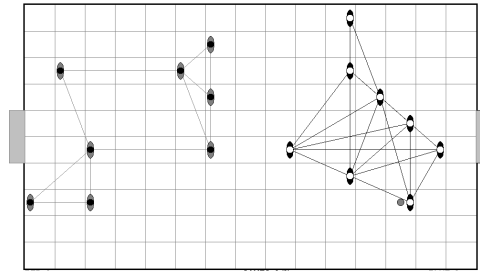


Figure 6. Example simplified soccer initial state with two teams of 8 players. Lines between players show the CG for the state.

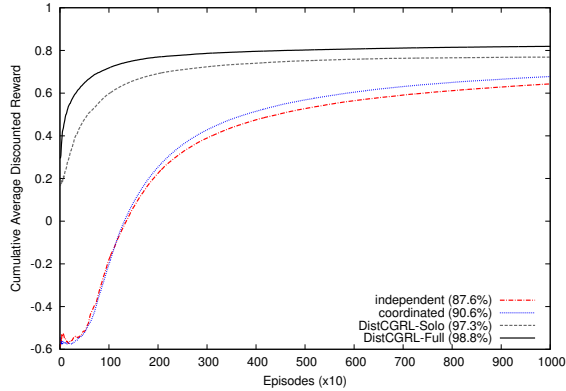
Each team has 8 soccer players. We pit the RL players against each of the two scripted strategy opponents: *defensive* players stay around the home quarter of the field and move to intercept the ball if it enters, the player with the ball counter-attacks; *aggressive* players always go for the ball, once attained, the player with the ball heads for the goal while each of the other players stays near (marks) their respective enemy players. The learners use discounting with $\gamma = 0.99$, $\epsilon = \{1.0, 0.01, 0.998\}$, and $\alpha = \{0.01, 10^{-4}, 0.998\}$. DistCGRL-Full uses 84 pairwise CCs including: passing to a player moving next to an opponent, not jointly intercepting the opponent with the ball, and jointly blocking opponents' movements. As before, DistCGRL-Solo uses only unary CCs but with pairwise BFs.

Fig. 7 presents the results. We plot cumulative average discounted reward achieved by the various players. The results for DistCGRL follow a similar trend previously observed in the RTS experiments. In Fig. 7(a), the independent player is comparable to the coordinated player. This may indicate that less coordination is required against a defensive opponent. However in Fig. 7(b), the coordinated player does much better than the independent player. Correspondingly, DistCGRL-Full's performance improvement over DistCGRL-Solo is larger for Fig. 7(b) compared to Fig. 7(a). Hence our proposed approach performs better when more coordination is required.

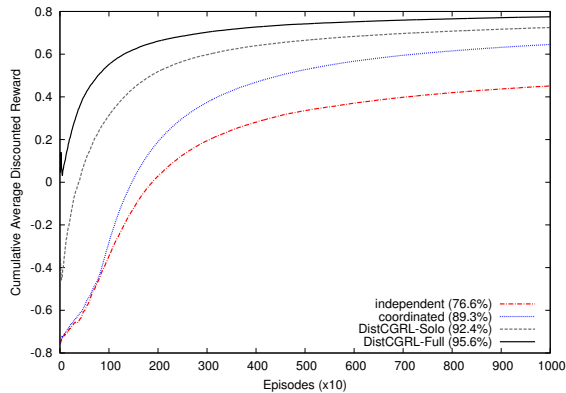
V. DISCUSSION & RELATED WORK

The experiment results show that fine-grained learning to employ CCs in different states is effective in improving the overall learning rate for a variety of different setups. This is as opposed to having only unary CCs defined on individual agents like task-based methods. Further benefit is observed over existing methods of learning independently or with coordination.

The DistCGRL system is different from [19] in that all learning is done online without any offline computation. Furthermore, DistCGRL does not require extra pseudo rewards to be defined for top level learning [2], [3]. Instead it learns from the original decomposed rewards of the DEC-MDP.



(a) Defensive Opponent



(b) Aggressive Opponent

Figure 7. Soccer results for 10,000 episodes averaged over 10 runs each. Final win rates over all episodes shown in brackets.

In [8], a framework was introduced where supervisor agents bias the policies of worker agents that select the primitive actions. However, the learning interactions of the supervisors were not defined. Our work can be viewed from the perspective that each agent carries local ‘supervision’ knowledge through CCs as our top level is distributed into the agent level. This is more applicable for domains where communication structure among agents change.

The work in [7] presented a centralized two level method for assigning agents to tasks. The original action value function is used for assignment. But, task assignment requires a costly nested maximization which they overcome using sampling methods. In our case, we avoid this problem by learning a separate top level value function to maximize for CC selection. Then, the top and bottom level value functions interact through the update equations.

VI. CONCLUSION

In this paper, we have presented the DistCGRL system that makes use of coordination knowledge by learning to use CCs to improve RL performance. Results show that this system is effective in the online setting where we wish to maximize rewards even while learning. The distributed update equations we have presented allows each agent to

carry a portion of the learned top level policy that decides on CCs to use in different states. This allows DistCGRL to operate in environments where the communication structure varies among agents and the number of agents may change.

Future work includes learning the representation of the CCs themselves to reduce dependence on expert knowledge, fusing the use of CCs with task-based methods that make use of temporal abstraction, and extending DistCGRL to domains with more restrictions on communication.

REFERENCES

- [1] M. Buro, “Call for AI research in RTS games,” in *AAAI Workshop on AI in Games*, 2004, pp. 139–141.
- [2] R. S. Sutton, D. Precup, and S. P. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [3] T. G. Dietterich, “Hierarchical reinforcement learning with the MAXQ value function decomposition,” *JAIR*, vol. 13, pp. 227–303, 2000.
- [4] D. Andre and S. J. Russell, “State abstraction for programmable reinforcement learning agents,” in *AAAI*, 2002, pp. 119–125.
- [5] B. Marthi, D. Latham, S. Russell, and C. Guestrin, “Concurrent hierarchical reinforcement learning,” in *IJCAI*, 2005, pp. 779–785.
- [6] M. Ghavamzadeh, S. Mahadevan, and R. Makar, “Hierarchical multi-agent reinforcement learning,” *AAMAS*, vol. 13, no. 2, pp. 197–229, 2006.
- [7] S. Proper and P. Tadepalli, “Solving multiagent assignment Markov decision processes,” in *AAMAS*, vol. 1, 2009, pp. 681–688.
- [8] C. Zhang, S. Abdallah, and V. Lesser, “Integrating organizational control into multi-agent learning,” in *AAMAS*, vol. 2, 2009, pp. 757–764.
- [9] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, “Heuristic selection of actions in multiagent reinforcement learning,” in *IJCAI*, 2007, pp. 690–696.
- [10] J. R. Kok and N. Vlassis, “Collaborative multiagent reinforcement learning by payoff propagation,” *Journal of Machine Learning Research*, vol. 7, pp. 1789–1828, 2006.
- [11] C. Guestrin, D. Koller, and R. Parr, “Multiagent planning with factored mdps,” in *NIPS*, 2001, pp. 1523–1530.
- [12] J. R. Kok, P. Jan, H. Bram, and B. N. Vlassis, “Utile coordination: Learning interdependencies among cooperative agents,” in *IEEE Sym. on CIG*, 2005, pp. 29–36.
- [13] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, “The complexity of decentralized control of markov decision processes,” *Math. Oper. Res.*, vol. 27, pp. 819–840, 2002.
- [14] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
- [15] C. Guestrin, M. G. Lagoudakis, and R. Parr, “Coordinated reinforcement learning,” in *ICML*, 2002, pp. 227–234.
- [16] R. Stranders, F. M. D. Fave, A. Rogers, and N. R. Jennings, “A decentralised coordination algorithm for mobile sensors,” in *AAAI*, 2010.
- [17] N. Asgharbeygi, D. J. Stracuzzi, and P. Langley, “Relational temporal difference learning,” in *ICML*, 2006, pp. 49–56.
- [18] C. Claus and C. Boutilier, “The dynamics of reinforcement learning in cooperative multiagent systems,” in *AAAI/IAAI*, 1998, pp. 746–752.
- [19] G. Chen, Z. Yang, H. He, and K. M. Goh, “Coordinating multiple agents via reinforcement learning,” *AAMAS*, vol. 10, pp. 273–328, 2005.