Coordination Guided Reinforcement Learning

Qiangfeng Peter Lau[♠], Mong Li Lee[†] and Wynne Hsu[§]
Department of Computer Science
National University of Singapore
13 Computing Drive, Singapore 117417, Republic of Singapore
{plau[♠],leeml[†],whsu[§]}@comp.nus.edu.sg

ABSTRACT

In this paper, we propose to guide reinforcement learning (RL) with expert coordination knowledge for multi-agent problems managed by a central controller. The aim is to learn to use expert coordination knowledge to restrict the joint action space and to direct exploration towards more promising states, thereby improving the overall learning rate. We model such coordination knowledge as constraints and propose a two-level RL system that utilizes these constraints for online applications. Our declarative approach towards specifying coordination in multi-agent learning allows knowledge sharing between constraints and features (basis functions) for function approximation. Results on a soccer game and a tactical real-time strategy game show that coordination constraints improve the learning rate compared to using only unary constraints. The two-level RL system also outperforms existing single-level approach that utilizes joint action selection via coordination graphs.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, Search

General Terms

Algorithms, Performance, Experimentation

Keywords

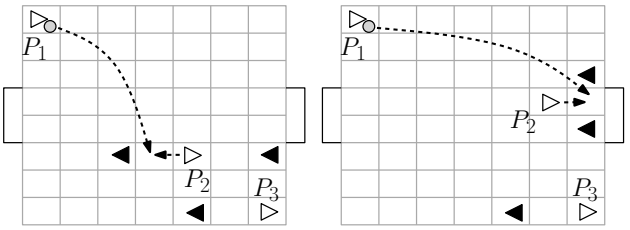
Reinforcement learning, guiding exploration, coordination constraints, factored Markov decision process

1. INTRODUCTION

Expert knowledge is commonly employed in large-scale reinforcement learning (RL) in a variety of ways. In particular, hierarchical RL handles single agent Markov decision processes (MDPs) by recursively partitioning them into smaller problems using a task hierarchy [19, 7, 1]. The task hierarchy constrains the solution space (policies) of the learning problem so that only relevant actions for a task can be selected at each time step. Learning a good task selection policy will direct exploration towards the more promising parts of the MDP.

For multi-agent problems, each agent has a set of actions whose Cartesian product forms the joint action space. This space is exponential in the number of agents and therefore, RL with naive exploration is slow. Hierarchical RL has been adapted to multi-agent problems [15, 9] by having one task hierarchy per agent where the actions are selected jointly. Once each individual agent's task is selected, it will have a constrained (reduced) set of actions to consider. However, this framework cannot be easily extended to incorporate coordination behavior among multiple agents.

Consider Fig. 1 which depicts a state in a soccer game and player P_1 has the ball. Let N, S, E, W, be the four compass directions. P_1's action set is $A_1 = \{S, E, pass2, pass3, shoot\}$ where $pass2$ and $pass3$ denote passing the ball to players P_2 and P_3 respectively, and $shoot$ denotes the action to kick the ball into the goal. Players P_2 and P_3 have the action set $A_2 = \{N, S, E, W\}$ and $A_3 = \{N, W\}$ respectively. We denote a joint action as $\langle a_1, a_2, a_3\rangle \in A_1 \times A_2 \times A_3$. The size of this joint action space is $5 \times 4 \times 2 = 40$. A closer examination reveals that much of this space does not need be explored as they are unlikely to lead to a winning state. For example, P_1 certainly should not pass the ball to P_2 if P_2 is moving adjacent to an opponent as the ball can easily be intercepted. With this simple coordination strategy, the set of disallowed joint actions is $\{pass2\} \times \{S, E, W\} \times A_3$. Similarly, P_1 should not pass the ball to P_3 and the set of disallowed joint actions is $\{pass3\} \times A_2 \times A_3$. Immediately, the size of the joint action space is reduced by 35%.



(a) Bad pass (b) Good pass

Figure 1: Example states in a simplified soccer game. The white ▷ versus black ◀ players.

In this paper, we focus on a central learner with multiple agents where communication is free. This corresponds to the scenario of a computer player managing an army in real-time strategy (RTS) games or a team of players in soccer. We aim to exploit coordination knowledge for improving the learning rate of good policies by modeling coordination among agents

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

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.

We propose a two-level RL system where the top level learns to choose the CCs to constrain the bottom level's learning of joint actions. The RL system learns to guide itself, i.e., deciding which CCs to use in various states is part of the learning process. This is a necessary flexibility because not all CCs are always useful in every state. For example, in Fig. 1a, the CC that P_1 should not pass the ball to P_2 is appropriate since there are opponents close to P_2. However, this CC may not be suitable, e.g., in Fig. 1b, where P_2 is standing directly in front of the goal as it may be better to pass to P_2 so that P_2 can try to score. Such a two-level system is different from RL for constrained MDPs [8] as the two-level system dynamically learns to use different constraints to improve learning, instead of using static constraints to prevent failure.

The proposed CCs are useful in addition to existing methods of modeling coordination with a communication structure such as a coordination graph (CG) for joint action selection [11, 12]. Unlike task-based methods where single agents are restricted based on individual tasks, CCs define these restrictions based on expert knowledge of multi-agent coordination. Such coordination is not easily expressed within single agent tasks. Existing works usually delegate them to the CG structure as features (basis functions) [15] or as static rules [16]. This further motivates us to investigate the potential of CCs in a more active role for improving learning performance.

Using CCs for online RL has three challenges. First, the system must be able to efficiently learn to activate the various CCs from its interaction with the environment. However, different combinations of activated CCs lead to large number of bottom level value functions to be learned. We address this by formulating learning equations that exploit similarities among the bottom level components of our system. Second, choosing CCs at the top level introduces an exponential top level joint action space to explore. We overcome this by identifying those CCs which can never be violated in a given state. Once identified, these CCs are deactivated, reducing the top level action space for exploration. Last, the system must integrate well with other useful methods for multi-agent learning in large state-action spaces.

To the best of our knowledge, this is the first online RL system that uses coordination to guide learning in multi-agent problems with a central controller. Our model-free approach frees the user from designing models for large MDPs with many variables. A major benefit of our system is that existing predicate definitions of features can be reused to specify CCs. This sharing of predicate components between CCs and features aids the user in encoding knowledge for the top level of the system. Experiments show that CCs give better results compared to having unary constraints or with coordination knowledge encoded only as a CG. For domains that require heavy coordination, using CCs leads to better policies, and hence better overall goal achievement.

2. TWO-LEVEL RL SYSTEM

An MDP is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of primitive actions, $\mathcal{P}(s'|s,a)$ is a transition probability model that gives the probability of going from state s to s' when action a is taken, and $\mathcal{R}(s,a,s')$ is a reward model that gives the reward of taking a in s and reaching s'. The set of available actions at state s is written as $\mathcal{A}(s)$. With N agents, the joint action space is factored as $\mathcal{A} = A_1 \times ... \times A_N$, where A_n is the action variable that corresponds to the n-th agent. $\mathcal{S}$ may also be factored into multiple variables in a similar way.

A solution to the MDP is a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ and the optimal policy π^* is one that maximizes the expected total discounted reward in any given state. Let the expected discounted sum of rewards when taking a in s at time t, observing reward r_t, and following π thereafter with discount rate γ be $Q^\pi(s,a) = E_\pi\{\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} | s, a\}$. Then $\pi^*(s) = \text{argmax}_{a \in \mathcal{A}(s)} Q^*(s,a)$. By learning Q^* directly, we obtain π^* without learning $\mathcal{P}$ and $\mathcal{R}$ [18].

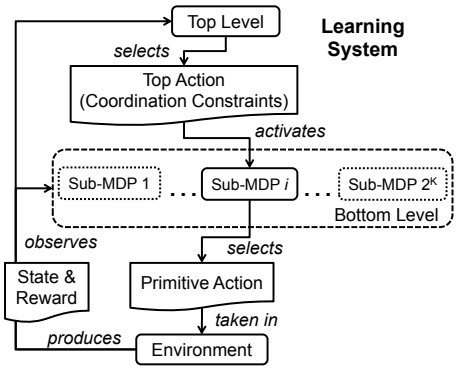


Figure 2: Two-level learning system

Fig. 2 depicts the proposed two-level learning system. The top level determines which CCs are relevant to the current state. The top level action space $\mathcal{A}_0$ consists of the *activation* or *deactivation* of each CC. With K constraints, the size of $\mathcal{A}_0$ is 2^K. However, in practice, the number of CCs that may be activated is typically small. The activated CCs restrict the bottom level to a sub-MDP, $i \in [1, 2^K]$, as shown in Fig. 2. Therefore, sub-MDP i corresponds to a unique top level action in $\mathcal{A}_0$. The joint action space of sub-MDP i, denoted by $\mathcal{A}_i$, is a subset of the original joint action space $\mathcal{A}$. This allows the bottom level to learn the original joint actions quickly. After the bottom level has taken an action in the environment, execution returns to the top level. Details of the learning equations, action selection and specification of CCs are given in Sections 2.1, 2.2 and 2.3. We also discuss how $\mathcal{A}_0$ can be reduced in Section 2.4.

2.1 Learning Equations & Updates

To model the system's two-level learning, we augment the original MDP's state space with an index i that keeps track of the position within the hierarchy. The top level corresponds to $i = 0$, and $i \in [1, 2^K]$ refers to one of the sub-MDPs. Note that when $i \in [1, 2^K]$, it also refers to a unique top level action in $\mathcal{A}_0$. The *augmented* MDP's state space is $\mathcal{S}' = [0, 2^K] \times \mathcal{S}$, and $\langle i, s \rangle \in \mathcal{S}'$ is an augmented state. The augmented action space is the union $\mathcal{A}' = \mathcal{A}_0 \cup \mathcal{A}$. Let an action in $\mathcal{A}'$ be $\tilde{a}$. Then, the transitions between levels in the hierarchy are deterministic while those between original states follow $\mathcal{P}$, resulting in the transition model,

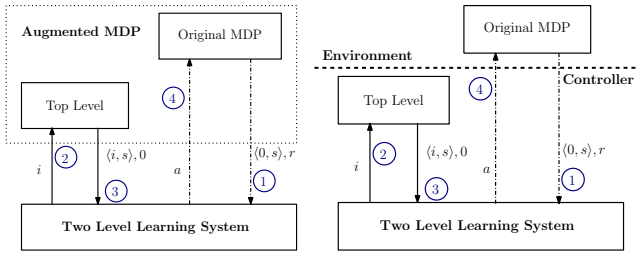
$$\mathcal{P}'(\langle i', s' \rangle | \langle i, s \rangle, \tilde{a}) = \begin{cases} \mathcal{P}(s'|s,\tilde{a}) & \text{if } i \neq 0 \wedge i' = 0 \wedge \tilde{a} \in \mathcal{A} \\ 1 & \text{if } i = 0 \wedge i' = \tilde{a} \wedge \tilde{a} \in \mathcal{A}_0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

and the reward model is

$$\mathcal{R}'(\langle i,s\rangle,\tilde{a},\langle i',s'\rangle)=\begin{cases}\mathcal{R}(s,\tilde{a},s') & \text{if } i'=0\\ 0 & \text{otherwise}\end{cases} \quad (2)$$

that is, the original reward model $\mathcal{R}$ is used when transiting between original states, all other transitions are zero. Note that the augmented MDP is also an MDP.

Fig. 3 illustrates the dynamics of the augmented MDP from the system's point of view and that of the original environment. Solid arrows indicate deterministic transitions while dotted-dashed arrows indicate non-determinism. Numbers describe a sequence of transitions between two original states. In Fig. 3a, the RL system operates as if the top level is part of the environment. Conversely in Fig. 3b, the original environment only receives original joint actions from the central controller that consists of multiple agents.



(a) RL System View (b) Environment View

Figure 3: Interactions between RL system, augmented MDP, and environment (MDP).

Let the hierarchical policy, $\pi:\mathcal{S}'\mapsto\mathcal{A}'$, that solves the augmented MDP be represented by a set of policies $\{\pi_i\}$ indexed by i such that $\pi(\langle i,s\rangle)=\pi_i(s)$. That is, π_0 denotes the top level policy, and $\pi_{i>0}$ denotes a policy constrained to the action space $\mathcal{A}_i(s)$ of the sub-MDP i. The Bellman equation of the action value function for the augmented MDP is,

$$Q^\pi(\langle i,s\rangle,\tilde{a})=\sum_{\langle i',s'\rangle\in\mathcal{S}'}\mathcal{P}'(\langle i',s'\rangle|\langle i,s\rangle,\tilde{a}) \times[\mathcal{R}'(\langle i,s\rangle,\tilde{a},\langle i',s'\rangle)+Q^\pi(\langle i',s'\rangle,\pi_i(s'))]. \quad (3)$$

Suppose we only use discounting (γ) for transitions between original states, we can rewrite and simplify Eq. 3 into two parts, $\forall i>0$, for the top level,

$$Q^\pi(\langle 0,s\rangle,i)=\sum_{s'\in S}\mathcal{P}_i(s'|s,\pi_i(s))[\mathcal{R}(s,\pi_i(s),s') +\gamma Q^\pi(\langle 0,s'\rangle,\pi_0(s'))] \quad (4)$$

and for the bottom level,

$$Q^\pi(\langle i,s\rangle,a)=\sum_{s'\in S}\mathcal{P}_i(s'|s,a)[\mathcal{R}(s,a,s') +\gamma Q^\pi(\langle 0,s'\rangle,\pi_0(s'))]. \quad (5)$$

Note that i in Eq. 4 refers to a unique top level action in $\mathcal{A}_0$, and we subscript $\mathcal{P}$ with i to indicate primitive actions that are disallowed based on i. Eq. 4 expresses the expected reward of taking a top level action i and following π_i for one step before returning to the top and following π_0 thereafter. Eq. 5 expresses the expected reward at the bottom level that returns to the top level immediately after one step and following π_0 thereafter. Now we can select greedy primitive actions in s by separately computing $\pi_0(s)=\text{argmax}_{i\in\mathcal{A}_0(s)}Q^\pi(\langle 0,s\rangle,i)$, followed by $\pi_i(s)=\text{argmax}_{a\in\mathcal{A}_i(s)}Q^\pi(\langle i,s\rangle,a)$.

Updating the 2^K $Q^\pi(\langle i,\cdot\rangle,\cdot)$ functions at the bottom level requires exponential space and time. However we observe that, although $\mathcal{P}_i$ and $\mathcal{P}_j$ for two sub-MDPs have different domains as their actions are from $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively, their probabilities are contained in the original MDP's $\mathcal{P}$. This is because the transition probability is a conditional probability where values are normalized over the state space but not the action space. Hence, given 2^K sub-MDPs, $\forall s\in\mathcal{S},i,j\in[1,2^K],a\in\mathcal{A}_i(s)\cap\mathcal{A}_j(s)$, we have

$$Q^\pi(\langle i,s\rangle,a)=Q^\pi(\langle j,s\rangle,a). \quad (6)$$

Consequently, the various sub-MDP functions are the same for their intersected domains. This leads to a single bottom level function definition: $\forall i>0,a\in\mathcal{A}_i(s)$,

$$U^\pi(s,a)=Q^\pi(\langle i,s\rangle,a) \quad (7)$$

that is independent of i. Incidentally, U^π is similar to the action value function for the original MDP but constrained to the joint actions in $\mathcal{A}_i(s)$ by the two-level policy π.

We use linear function approximation to learn the value functions $Q^\pi(\langle 0,\cdot\rangle,\cdot)$ and U^π. This employs a linear combination of m number of features (basis functions), ϕ_p, with weights, w_p, to be learned. In other words, given a function $F(s,a)$, we want to find $\vec{w}=\langle w_1,...,w_m\rangle$ such that, $F(s,a)\approx\vec{w}\cdot\vec{\phi}_{s,a}$, where $\vec{\phi}_{s,a}=\langle\phi_1(s,a),\cdots,\phi_m(s,a)\rangle$.

To obtain $\vec{w}$ for each optimal value function, we perform on-policy temporal difference (TD) updates using the hierarchical policy π given by $\{\pi_i\}$, where each π_i is a GLIE policy [7], and online samples of the form $\langle\langle 0,s\rangle,i,\langle i,s\rangle,a,r,\langle 0,s'\rangle\rangle$. The first two entries in the sample denote that the top level is in the augmented state $\langle 0,s\rangle$ and it chooses the action $\pi_0(s)=i\in\mathcal{A}_0(s)$. The next two entries in the sample indicate the state $\langle i,s\rangle$ of the bottom level and the primitive action $\pi_i(s)=a\in\mathcal{A}_i(s)$ taken by it. The final two entries indicate that both levels observe reward r and go to next state s'. Note that when the bottom level policy chooses an exploratory action, i.e., $\pi_i(s)$, it does so by choosing a random action within the constrained joint action space $\mathcal{A}_i(s)$ as specified by the top level action i.

Let $Q^\pi(\langle 0,s\rangle,i)\approx\vec{w_0}\cdot\vec{\phi_0}_{s,i}$, $U^\pi(s,a)\approx\vec{w_U}\cdot\vec{\phi_U}_{s,a}$, and α be the step size parameter that decreases over time. Then, the weights $\vec{w_0}$ and $\vec{w_U}$ are updated as follows:

$$\vec{w_0}\leftarrow\vec{w_0}+\alpha[r+\gamma Q^\pi(\langle 0,s'\rangle,\pi_0(s')) -Q^\pi(\langle 0,s\rangle,i)]\vec{\phi_0}_{s,i} \quad (8)$$

$$\vec{w_U}\leftarrow\vec{w_U}+\alpha[r+\gamma Q^\pi(\langle 0,s'\rangle,\pi_0(s')) -U^\pi(s,a)]\vec{\phi_U}_{s,a} \quad (9)$$

2.2 Action Selection

Applying the policies π_i often requires selecting some primitive action within $\mathcal{A}_i(s)$ that maximizes the value functions. For example, the ϵ-greedy bottom level policy is to select a maximal action $\pi_i(s)=\text{argmax}_{a\in\mathcal{A}_i(s)}U^\pi(s,a)$ with $1-\epsilon$ probability, or a random action within $\mathcal{A}_i(s)$ with ϵ probability. In our system, $\mathcal{A}_i(s)$ is subjected to the constraints activated by the top level. This implies that the problem

of finding a maximal action, $\text{argmax}_{a \in \mathcal{A}_i(s)} U^\pi(s,a)$, can be modeled as a constraint optimization problem (COP) over the full original action space $\mathcal{A}$ as follows:

$$\underset{a \in \mathcal{A}}{\text{argmax}}\, U^\pi(s,a), \qquad \text{subject to:} \qquad (10)$$
$$c_{i,1}(s,a), ..., c_{i,p}(s,a) \quad c_{0,1}(s,a), ..., c_{0,q}(s,a)$$

where the constraints $c_{i,j}(s,a)$ are activated by the top level action i, termed dynamic CCs, and the constraints $c_{0,j}(s,a)$ are always activated regardless of i, termed static CCs.

Let each constraint be a function on a subset of variables in $\mathcal{S}$ and $\mathcal{A}$ that returns $-\infty$ if violated, or 0 otherwise. Then the objective function to maximize for the COP is

$$U^\pi(s,a) + C_0(s,a) + C_i(s,a) \qquad (11)$$

where C_0 and C_i are the sum of their respective constraints $c_{0,j}$ and $c_{i,j}$. Note that to switch to selecting random actions within $\mathcal{A}_i(s)$, we can simply replace U^π with a random function. Likewise, the selection of action for the top level, e.g., $\text{argmax}_{i \in \mathcal{A}_0(s)} Q^\pi(\langle 0, s\rangle, i)$, can be similarly modeled.

Depending on the characteristics of the COPs, we employ different strategies to solve them. If the problem consists of features and constraints that can be additively decomposed into component functions involving up to two action variables, we can utilize the coordination graph (CG) [11] with bucket elimination for sparse CGs, or the Max-plus algorithm [14] for dense CGs. CGs are formed by having one vertex for each agent (action variable) and an edge between two agents that have a non-zero component function involving them. We further employ domain reduction techniques to prune $\mathcal{A}$. If the problem involves higher arity features and constraints, more generalized solvers can be employed [17]. Furthermore, if top level actions activating CCs are presumed to be independent, we may use features for $Q^\pi(\langle 0, \cdot\rangle, \cdot)$ that only involve the state and one action variable corresponding to one CC. Consequently, top level actions can be selected independently in $O(K)$ time while bottom level actions are selected jointly. This turns out to be sufficient for good performance shown in Section 3.

2.3 Features & Constraints

Next, we show how existing predicate definitions of features can be reused to specify the CCs. We further describe how the top and bottom levels' BFs may share predicate components in their design and highlight a type of features that may be useful for certain multi-agent problems.

Predicates are a natural way to encode expert knowledge as features for RL. For example, the expert knowledge of a bad pass can be written as the predicate:

$$BadPass(s, a_x, a_y) := HasBall(P_x) \wedge IsPass(P_y, a_x) \wedge MoveNextToOpp(s, a_y),$$

where $HasBall(P_x)$ is true if player P_x has the ball, $IsPass(P_y, a_x)$ is true if the action of player P_x, a_x, is to pass the ball to P_y, and $MoveNextToOpp(s, a_y)$ is true if the action of player P_y is to move next to an opponent.

With this predicate $BadPass$, we can derive the corresponding list of propositional features (PFs) by binding the variables P_x, P_y to specific players. For example, for P_1, P_2 we have the PF, $\phi_{BadPass_{1,2}}(s,a) = BadPass(s, a_1, a_2)$, for brevity we write $BadPass_{1,2}$. The value of a PF is in $\{0, 1\}$.

PFs are commonly employed in existing RL systems to approximate the value function [15, 13, 2]. We also utilize PFs for the bottom level function U^π. An immediate advantage is that the predicates for PFs can be reused for specifying CCs. For each PF $\phi_\rho(s,a)$, we can formulate it into a constraint that disallows the proposition ρ, i.e.:

$$c_\rho(s,a) = -\infty \cdot \phi_\rho(s,a). \qquad (12)$$

If $\phi_\rho(s,a) = 1$, $c_\rho(s,a)$ returns $-\infty$, signifying that the constraint c_ρ that represents the condition $\neg\rho$ is violated.

Reusing PFs in U^π as CCs has an added advantage during bottom level action selection. Instead of specifying individual constraints $c_{i,j}$ to sum for C_i in the objective function in Eq. 11, we can simply set the corresponding PFs' weights of the activated CCs to $-\infty$. In so doing, the set of actions that are disallowed by the CCs will never be chosen due to its $-\infty$ weight.

Note that we restore the original weights of these PFs during the updates (Eq. 9). This is because in practice, incomplete COP solvers like Max-plus may still select actions that violate certain activated CCs. When this happens, we can learn the weights for the violated constraint PFs that are useful for updating other weights. Hence PFs can be used both as constraints for guiding exploration and for function approximation.

The top level value function $Q^\pi(\langle 0, \cdot\rangle, \cdot)$ is also a linear approximation. Here, we describe how the predicates for bottom level PFs can be reused for the top level features. We observe that the activation of a constraint is often dependent on the state of the environment. Hence, we encode such state-dependent activation knowledge as the top-level PFs in the following manner: Let $Activated(c)$ be true if constraint c is activated. For each c corresponding to some PF for the bottom level, we conjunct $Activated(c)$ or its negation with selected state predicates of agents involved in c.

For example, in the soccer scenario, we would like to deactivate the $BadPass_{1,2}$ constraint if the receiving player P_2 is near the enemy goal, i.e., $NearGoal(P_2)$ is true as shown in Fig. 1b. This is because it may turn out to be better to take a chance at scoring. Hence, we define a predicate $NearGoal(P_y) \wedge \neg Activated(BadPass_{x,y})$ for each pair of players to capture this condition in the top level value function. This simple strategy allows us to design top level features easily by reusing bottom level features' predicates.

For applications where the agents are homogeneous or their quantity changes over time, a new class of features called relational features (RFs) can be utilized [20]. Here, we modify the relations used in [10, 2] for function approximation by aggregating PFs that share the same predicate into RFs. RFs are additively decomposable into components involving a subset of agents, e.g. an RF of $BadPass_{x,y}$ can be the count of true bound predicates for each pair of players, P_x, P_y. A weight is learned for the RF instead of one for each of the PFs, i.e., the update by observing the true proposition $BadPass_{1,2}$ will have its effect generalized to other pairs of players for $BadPass$. RFs can greatly reduce the number of weights for the top level PFs relating to multi-agent CCs. For example, the predicate $NearGoal(P_y) \wedge \neg Activated(BadPass_{x,y})$ can be used to construct an RF to reduce $N(N-1)/2$ weights to one weight.

2.4 Top Level Learning Efficiency

Finally, we discuss how the 2^K top level action space can be reduced for exploration and consequently, faster learning. In many domains, we observe that the top level action space,

$\mathcal{A}_0$, may be heavily constrained based on the current state, s. This yields a smaller $\mathcal{A}_0(s)$ to explore. In fact, this reduction to $\mathcal{A}_0(s)$ can be directly derived from the predicates used to create the CCs. If it can be inferred that a CC cannot be violated in the current state s, then the CC need not be activated. This can be done in $O(K)$ time as we only need to inspect the predicates of each of the K CCs.

Consider the $BadPass_{2,3}$ CC. Since only player P_1 has the ball (see Fig. 1), $HasBall(P_2)$ is false and CC for $BadPass_{2,3}$ can be deactivated. We can also deactivate $BadPass_{x,y}$ CCs for other pairs of players where P_x does not have the ball, thus reducing the quadratic number of $BadPass_{x,y}$ CCs to a linear number of $BadPass_{1,y}$ CCs.

Another observation is that agents who are very far apart do not need to coordinate. We can define a $Nearby_{x,y}$ predicate that is true if two agents are within a given distance and conjunct it with those predicates involving them. This simple strategy has proven effective in reducing $\mathcal{A}_0$ for directing top level exploration in practice.

3. EXPERIMENT RESULTS

We carried out experiments to evaluate the proposed approach on two domains: simplified soccer (Fig. 1) and tactical real-time strategy (RTS) [4]. The environments are fully observable and episodic. All learning is online as no experience is saved and replayed. We compare four RL players. *Independent player* – each agent selects their own action independently and learns a separate policy [6]. *Flat player* – RL using single level on-policy learning with coordination graph (CG) defined by features for joint action selection [12]. *Solo player* – a representative of having individual task hierarchies for each agent. Only unary constraints are used. *Coordinated player* – uses our full two-level learning system. For function approximation, the solo and coordinated players use the same features as the flat player for their bottom level value function. Hence they have the same CGs defined by the features. The independent player has only features that involve single agents.

We design three types of experiments to investigate the performance of the two-level learning system. Each experiment progressively includes other methods that make RL in multi-agent domains practical. A video of the sample runs of our policies can be viewed at: `http://youtu.be/aloAOTBEUZ4`.

3.1 Simplified Soccer Domain

In soccer, the objective is to score the first goal in the shortest time. The soccer field is a *grid world*. Soccer players can stay, move in 4 directions, or the player with the ball may pass or shoot with a probabilistic chance 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. In each time step, submitted player actions are randomly shuffled and executed. Rewards are 1 for winning, and -1 for losing.

There are three types of scripted opponents in order of difficulty: *random* players select actions at random; *defensive* players stay around the home quarter of the field and move to intercept the ball if it enters, the player with the ball does a solo counter-attack; *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. In soccer, good policies may require agents to have specific roles.

In addition to unary CCs, for pairwise CCs in soccer, we used a predicate for static collision CCs and three predicates for dynamic CCs including: $BadPass$, not jointly intercepting as opponent with the ball, and jointly blocking opponents' movements.

3.1.1 Only Exact Methods

For the first type of experiments, we examine our proposed two-level RL system without any approximation, using exact tabular functions and action selection via enumeration. This is to investigate if the extra top level is indeed useful for learning performance. A tabular function is equivalent to a linear function approximation where each feature is a boolean variable corresponding to an entry in the table. We only compare tabular flat and coordinated players.

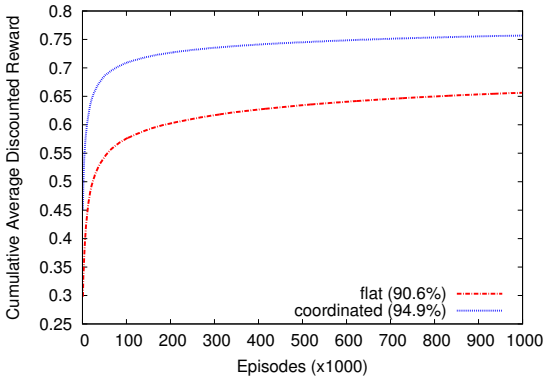
The soccer field is 6×4 units and the RL players have 2 soccer players versus 1 player for each type of scripted opponent. RL players used discounting ($\gamma = 0.99$), ϵ-greedy policies with constant $\epsilon = 0.1$, and constant step size ($\alpha = 10^{-3}$). The coordinated player used only pairwise CCs and no unary CCs for this experiment. Its top level has 3 dynamic CCs giving a top joint action space of size 2^3. With tabular functions the number of parameters to learn are more than 10,000 for each RL player. Hence we run the experiments for many episodes.

Fig. 4 shows the results for the three scripted opponents. 1 million episodes are used against random, and 10 million episodes against both defensive and aggressive opponents. From the results we see that learners have poorer goal achievement against harder opponents. The coordinated player performs consistently better than the flat player. This verifies that the top level of our system is stable and improves learning performance when there are no other factors.

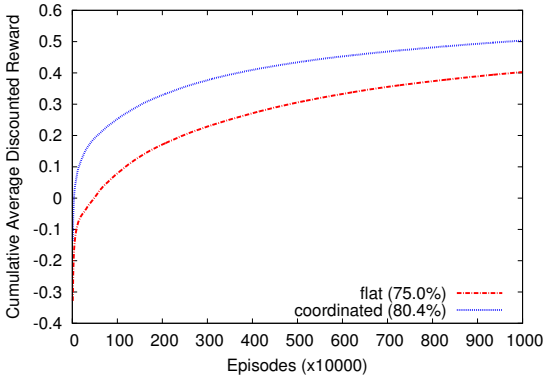
3.1.2 With Function Approximation

For the second type of experiments, we evaluate the RL system using propositional features (PFs) for linear value function approximation and exact action selection through bucket elimination. Top level action selection is independent (see Section 2.4). The soccer field is 12×8 units. The RL players have 4 soccer players versus 6 players each using the defensive and aggressive scripts. The size of the state space is at least 10^{13} and the size of the action space is 8^4. The top level has 6 static CCs and $18 (= 6 \times 3)$ dynamic pairwise CCs. The learners used discounting with $\gamma = 0.99$, and ϵ-greedy policies with decaying step size (α) and exploration (ϵ) parameters written as $param = \langle \text{initial, final, decay rate} \rangle$: all players used $\epsilon = \langle 1.0, 0.01, 0.998 \rangle$, the flat player used $\alpha = \langle 0.2, 0.1, 0.998 \rangle$, and the rest used $\alpha = \langle 0.1, 0.01, 0.998 \rangle$.

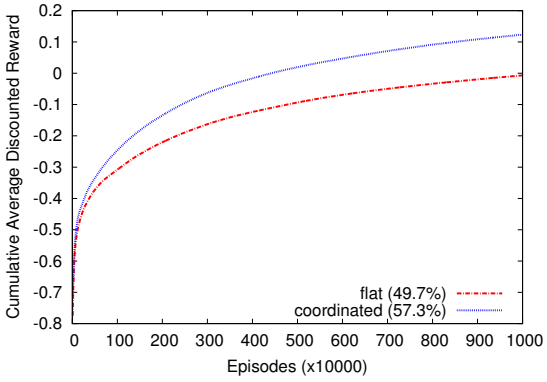
Fig. 5 shows the cumulative average discounted rewards for the soccer setup against the two different scripted strategies. Note that the plots start from the tenth episode. For all opponents, the flat player performs better than the independent player, indicating that coordination is important for simplified soccer. The solo and coordinated players' policies converged and performs better than the flat and independent players. This shows that our proposed system results in better policies than flat RL with coordination graphs. The coordinated player performs better than the solo player against both the defensive and aggressive opponents. Its benefit came from good early exploration compared to the other RL players. This indicates that CCs are effective in



(a) Random, 2 v. 1 soccer players, 1M episodes



(b) Defensive, 2 v. 1 soccer players, 10M episodes



(c) Aggressive, 2 v. 1 soccer players, 10M episodes

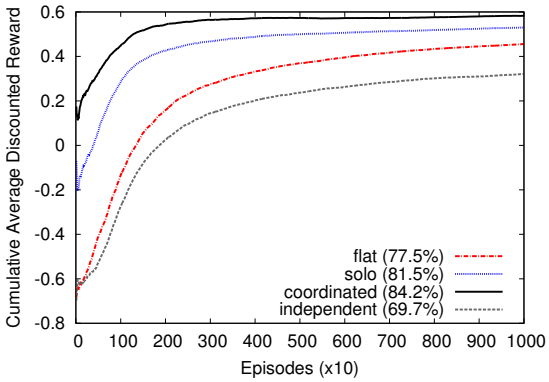
(Final win rates in brackets)

Figure 4: Soccer results for Section 3.1.1, each plot averaged over 10 runs.

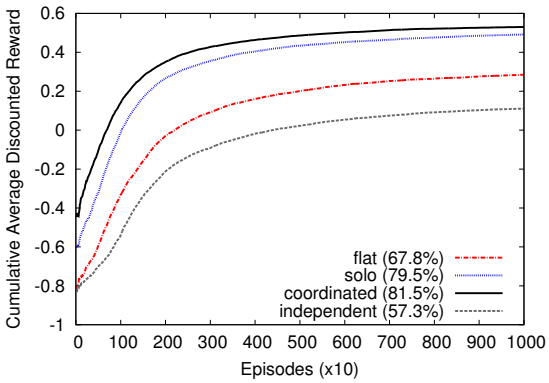
the online setting where it is generally harder to improve better policies due to the exploration-exploitation trade-off.

3.2 Tactical RTS Domain

The goal in tactical RTS is to eliminate the enemy team of marines quickly in a 240×240 *point based* map. Each marine occupies a point on the map with a fixed radius and a number of hit points. When its hit points reaches zero, it is destroyed. 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 huge state space consists of all the possible marines' positions and hit points.



(a) Defensive, 4 v. 6 soccer players.



(b) Aggressive, 4 v. 6 soccer players.

(Final win rates in brackets)

Figure 5: Soccer results for Section 3.1.2. 10K episodes averaged over 10 runs each.

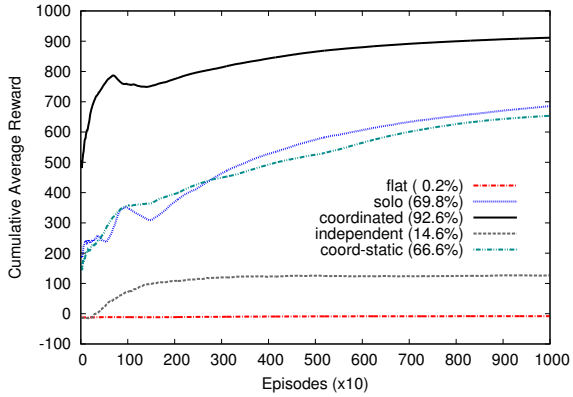
We pit the RL players against two scripted opponents: *aggressive* marines head for the nearest enemy and shoot enemies in range, *unpredictable* marines move in random directions and shoot enemies in range. The unpredictable opponent may be strong if its marines move in the same direction towards the 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. Rewards are -0.1 per time step and 10^3 for opponent team elimination.

3.2.1 *Relational Features & Inexactness*

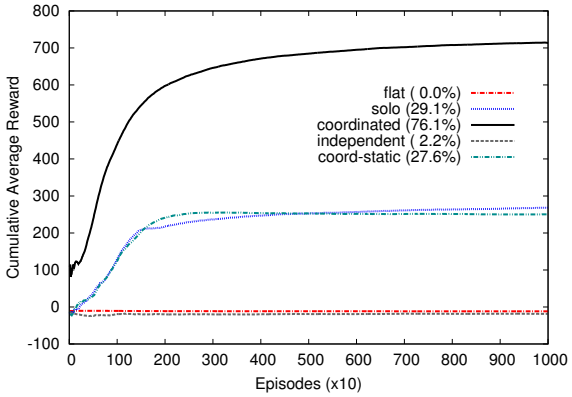
The final type of experiment integrates methods to deal with large multi-agent problems where the number of agents change over time. We incorporate the use of relational features (RFs) for generalization of learning and approximate primitive action selection using the Max-plus algorithm.

Four setups are used in our experiments: (a) 10 RL marines versus 10 aggressive marines, (b) 10 RL marines versus 13 unpredictable marines, (c) 10 RL marines versus 13 aggressive marines, (d) 10 RL marines versus 5 unpredictable super marines. The super marines in setup (d) have twice the firepower and hit points, hence coordination for the RL players is very crucial for success.

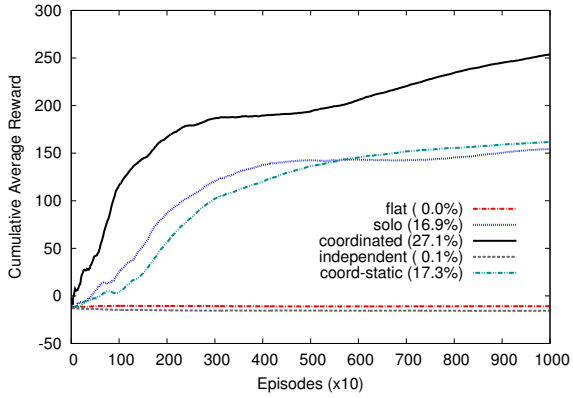
For each setup, the RL players use $\gamma = 1$ with the same decaying parameters. Setup (a), (c), (d) used $\epsilon = \langle 1, 0.01, 0.998 \rangle$, $\alpha = \langle 0.01, 10^{-4}, 0.998 \rangle$, while (b) used $\epsilon = \langle 1, 0.1, 0.998 \rangle$, $\alpha = \langle 0.01, 10^{-6}, 0.998 \rangle$. Setup (b) was given more



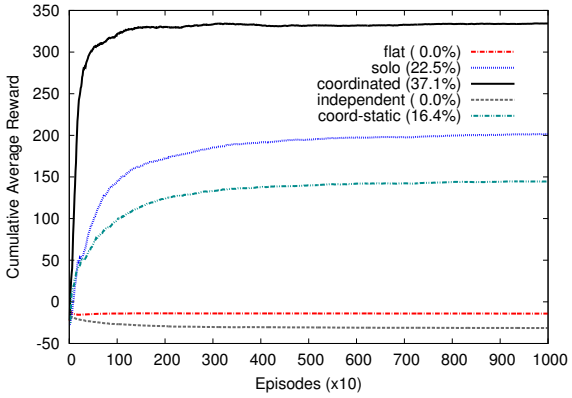
(a) Aggressive, 10 v. 10 marines.



(b) Unpredictable, 10 v. 13 marines.



(c) Aggressive, 10 v. 13 marines.



(d) Unpredictable, 10 normal v. 5 super marines.

(Final win rates in brackets. Approximate action selection using Max-plus.)

Figure 6: RTS results for Section 3.2.1. 10K episodes averaged over 10 runs each.

exploration due to the unpredictable nature of the opponent. Agents need to coordinate if they are within 30 points of each other, otherwise their pairwise features are set to zero. RFs are useful for RTS as the number of agents vary over time, except for the independent player that learns separate policies. We used 45 static collision CCs and 90 dynamic CCs from predicates for troop formation to maximize overlapping firepower, and to protect wounded teammates.

Fig. 6 shows the cumulative average reward for the RTS setups. We also included a new RL player, *coord-static*, which utilized only the static collision CCs in addition to the capabilities of the solo player. Total percentage wins are shown in brackets. We observe that all the RL players' policies converged over time. As seen in soccer, all two-level players: solo, coord-static, and coordinated; out-performed existing approaches of independent, and flat players. The coordinated player was able to learn quickly in all the four setups. The flat player experienced few winning episodes and ended up trying to lose as fast as possible to reduce the total negative reward obtained from each time step. The independent player managed to learn some strategy in Fig. 6a and wins more episodes than the flat player in the other setups. However, its reward is less than the flat player in those setups.

The results for the coord-static and solo players are mostly comparable. This is because the coord-static's marines tend to spread out more often and are destroyed easily when isolated. This occurs while ϵ is high and it has yet to learn a good formation. After sufficient exploration, the coord-static player is competitive with the solo player, although it learned a poorer policy in (d).

Conversely, the performance gains by the coordinated player with dynamic pairwise CCs are large compared to the others. Hence CCs are obviously crucial for tactical RTS. It is clear that most learning benefits came from the dynamic CCs. The coordinated player has more coordination than the solo player. This is also confirmed in our video which shows the coordinated player overcoming the enemy force simply by having better coordination among its marines. The results indicate that allowing the RL system to guide its exploration via dynamic CCs is effective for improving the learning rate in MDPs with large joint action spaces.

4. DISCUSSION & RELATED WORK

Previous works in task-based RL for multiple agents [15, 9, 16] require users to define tasks, terminating conditions, reward decomposition among tasks and agents, and new features for every level in the task hierarchy to represent them. They learn high level actions to constrain the exploration of the original MDP's actions based on single agent tasks. But it is not straightforward to incorporate coordination among agents to aid exploration. In contrast, the proposed two-level RL system employs declarative CCs and allows existing predicates for features to be reused as CCs to guide itself. Our results show that the two-level learning system outperforms task-based RL with coordination graphs when

comparing their ability at constraining exploration. This demonstrates that dynamic multi-agent CCs are important.

The work in [16] presented a two-level method where the top level assigns tasks and the bottom level learns with the task restrictions. Our work differs as our top level explores CC activations that are defined on multiple agents, and learns a value function that eliminates the need for a costly nested maximization when selecting CCs to activate. The proposed CCs are distinct from methods that learn coordination structure [13] within the value functions themselves. In our work, we use CCs to direct exploration by specifying subsets of the joint action space to be pruned. This is dynamically learned by our two-level system. The work in [5] presented a method that used constraints involving multiple agents. They require an offline phase for learning with constraints, and the constraints are static. Our work is fully online from the onset and learns to use CCs in different states. In [21], fixed heuristic supervisor agents biased base agents' policies with a coarse-grained approach. In contrast, ours employs fine-grained RL at the top level using the original reward signal and state observations.

Another branch of works deal with zero communication multi-agent problems known as Markov games where the focus is on handling the non-stationary environment due to independent learning and the setting is mostly adversarial [22]. In [3], heuristics can be provided to influence learning when the policy selects a maximal action. Their heuristics do not affect exploratory actions and are used in an adversarial setting with a much smaller action space.

5. CONCLUSION

We have investigated the use of expert coordination knowledge to improve RL via CCs for multi-agent MDPs from a centralized perspective. The proposed system's top level learns to activate CCs to guide the bottom level's exploration towards better experience. Learning to activate CCs allows flexibility in discovering good policies. Conversely, having only static CCs may lead to over-constraining the policy. We conducted experiments that progressively integrate our system with other useful methods for multi-agent problems. Our results on different domains demonstrate that the two-level RL system leads to better policies compared to existing approaches. Further, RL with CCs makes better use of early exploration, especially with multi-agent CCs. This is advantageous for online applications as overall higher goal achievement is attained. Future work involves automating the construction of CCs to reduce reliance on expert knowledge. Others include decentralized learning for distributed applications with communication costs, and fusing our method with task-based methods.

6. ACKNOWLEDGMENTS

This work was supported by A*STAR Exploit Flagship Grant ETPL/10-FS0001-NUS0.

7. REFERENCES

[1] D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *AAAI*, pages 119–125, 2002.

[2] N. Asgharbeygi, D. J. Stracuzzi, and P. Langley. Relational temporal difference learning. In *ICML*, pages 49–56, 2006.

[3] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa. Heuristic selection of actions in multiagent reinforcement learning. In *IJCAI*, pages 690–696, 2007.

[4] M. Buro. Call for AI research in RTS games. In *AAAI Workshop on AI in Games*, pages 139–141. AAAI Press, 2004.

[5] G. Chen, Z. Yang, H. He, and K. M. Goh. Coordinating multiple agents via reinforcement learning. *AAMAS*, 10:273–328, 2005.

[6] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

[7] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR*, 13:227–303, 2000.

[8] P. Geibel. Reinforcement learning for mdps with constraints. In *ECML*, volume 4212 of *LNCS*, pages 646–653, 2006.

[9] M. Ghavamzadeh, S. Mahadevan, and R. Makar. Hierarchical multi-agent reinforcement learning. *AAMAS*, 13(2):197–229, 2006.

[10] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational mdps. In *IJCAI*, pages 1003–1010, 2003.

[11] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *NIPS*, pages 1523–1530, 2001.

[12] C. Guestrin, M. G. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *ICML*, pages 227–234, 2002.

[13] J. R. Kok, P. Jan, H. Bram, and B. N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *IEEE Sym. on CIG*, pages 29–36, 2005.

[14] J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828, 2006.

[15] B. Marthi, D. Latham, S. Russell, and C. Guestrin. Concurrent hierarchical reinforcement learning. In *IJCAI*, pages 779–785, 2005.

[16] S. Proper and P. Tadepalli. Solving multiagent assignment markov decision processes. In *AAMAS*, volume 1, pages 681–688, 2009.

[17] R. Stranders, F. M. D. Fave, A. Rogers, and N. R. Jennings. A decentralised coordination algorithm for mobile sensors. In *AAAI*, pages 874–880, 2010.

[18] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[19] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[20] P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *ICML'04 Workshop on Relational Reinforcement Learning*, pages 1–9, 2004.

[21] C. Zhang, S. Abdallah, and V. Lesser. Integrating organizational control into multi-agent learning. In *AAMAS*, volume 2, pages 757–764, 2009.

[22] C. Zhang and V. R. Lesser. Multi-agent learning with policy prediction. In *AAAI*, pages 927–934, 2010.