Deep Implicit Coordination Graphs for Multi-agent Reinforcement Learning

Sheng Li
Stanford University
Stanford, CA
lisheng@stanford.edu

Jayesh K. Gupta
Stanford University
Stanford, CA
jkg@cs.stanford.edu

Peter Morales
Microsoft
Redmond, WA
pmorales@microsoft.com

Ross Allen
MIT Lincoln Labs
Lexington, MA
ross.allen@ll.mit.edu

Mykel J. Kochenderfer
Stanford University
Stanford, CA
mykel@stanford.edu

ABSTRACT

Multi-agent reinforcement learning (MARL) requires coordination to efficiently solve certain tasks. Fully centralized control is often infeasible in such domains due to the size of joint action spaces. Coordination graph formalizations allow reasoning about the joint action based on the structure of interactions. However, they often require domain expertise in their design and can be difficult for dynamic environments with changing coordination requirements. This paper introduces the *deep implicit coordination graph* (DICG) architecture for such scenarios. DICG consists of a module for inferring the dynamic coordination graph structure which is then used by a graph neural network module to learn to implicitly reason about the joint actions or values. DICG allows learning the tradeoff between full centralization and decentralization via standard actor-critic methods to significantly improve coordination for domains with large number of agents. We apply DICG to both centralized-training-centralized-execution and centralized-training-decentralized-execution regimes. We demonstrate that DICG solves the *relative overgeneralization* pathology in predatory-prey tasks as well as outperforms various MARL baselines on the challenging StarCraft II Multi-agent Challenge (SMAC) and traffic junction environments.

KEYWORDS

Coordination, Graph Neural Network, Deep Reinforcement Learning

ACM Reference Format:
Sheng Li, Jayesh K. Gupta, Peter Morales, Ross Allen, and Mykel J. Kochenderfer. 2021. Deep Implicit Coordination Graphs for Multi-agent Reinforcement Learning. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), London, UK, May 3–7, 2021*, IFAAMAS, 11 pages.

1 INTRODUCTION

Effective multi-agent reinforcement learning (MARL) in fully cooperative environments often requires coordination between agents on a team. One simple approach for achieving coordination is to reduce the problem to a single agent problem where the action space is the joint action space of all agents. Unfortunately, this joint action space grows exponentially with the number of agents, making it intractable for many domains of interest. To avoid this problem, a common strategy is to *decentralize* or factorize the decision policy or value function for each agent [8, 36, 38]. Each agent selects actions to maximize its corresponding utility function, with the end goal of maximizing the joint value function. However, such decentralization can be suboptimal [24]. The optimal policy is often not learnable in such a context due to a game-theoretic pathology referred to as *relative overgeneralization* [28], where the agent's reward gets confounded by penalties from random exploratory actions of other collaborating agents.

Guestrin et al. [7] introduced the framework of *coordination graph* (CG) to reason about joint value estimates from a factored representation to significantly improve computational tractability at the expense of optimality. Compared to function decomposition schemes like Value Decomposition Networks (VDN) [37], QMIX [30], and parameter sharing in decentralized policy optimization [8], the CG framework allows explicit modeling of the locality of interactions and formal reasoning about joint actions given the coordination graph structure. Kok and Vlassis [18] applied these ideas in the context of tabular reinforcement learning. The approach was later extended to function approximation with neural networks by Böhmer et al. [3]. Most of these approaches assume a domain dependent static coordination graph is given. Although the coordination graph terminology is focused on using a graph data-structure to decompose payoffs and utilities, we believe the idea of using a graph data-structure for coordinated action inference can be considered more general.

For a wide range of problems, this coordination graph structure is dynamic and state dependent. Domain heuristics like adding a graph edge with neighboring agents based on some distance metric are sometimes used [13]. The approach of Kok et al. [17] attempts to learn such structure, but it is limited to tabular settings with domain heuristics. We hypothesize that methods that learn the appropriate dynamic coordination graph to inform the selection of joint actions can help address coordination issues in MARL.

We propose the *Deep Implicit Coordination Graph* (DICG) module for multi-agent deep RL. It uses a self-attention network to determine, what we call, an implicit coordination graph structure which is then used for agent information integration through a graph convolutional network (GCN). Although "implicit coordination

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, London, UK.

graph" is not strictly the payoff-utility based coordination graph as is standard in the literature, it builds off the same idea of reasoning about the joint action based on the relatively sparse interactions between the agents. The intuition behind this architecture design is to make both the coordination graph structure and action inference over its edges differentiable so that the entire DICG module can be used inside either the actor or the critic and trained end-to-end through standard policy optimization methods. Since the module is trained to optimize the joint reward, the GCN submodule learns to implicitly reason about joint actions/values based on the structure of interaction inferred by the attention submodule.

We compare DICG with fully centralized and decentralized MARL methods in a challenging domain involving predator-prey tasks that require strong coordination. We also study performance on the StarCraft II Multi-Agent Challenge (SMAC) [32] and the traffic junction environment [36]. DICG learns the relevant dynamic coordination graph structure, allowing it to make an appropriate trade-off between centralized and decentralized methods.

2 BACKGROUND AND RELATED WORK

We formalize the problem as a Dec-POMDP [26] forming the tuple $\langle \mathcal{I}, \mathcal{S}, \{\mathcal{A}^i\}_{i=1}^n, \mathcal{T}, \mathcal{Z}, R, \mathcal{O}, \gamma \rangle$, where $\mathcal{I} = \{1, \ldots, n\}$ is the set of agents, $\mathcal{S}$ is the global state space, $\mathcal{A}^i$ is the action space of the ith agent, and $\mathcal{Z}$ is the observation space for an agent. The transition function defining the next state distribution is given by $\mathcal{T} : \mathcal{S} \times \prod_i \mathcal{A}^i \times \mathcal{S} \to [0, 1]$. The reward function is $R : \mathcal{S} \times \prod_i \mathcal{A}^i \to \mathbb{R}$, and the discount factor is $\gamma \in [0, 1)$. The observation model defining the observation distribution from the current state is $\mathcal{O} : \mathcal{S} \times \mathcal{Z} \to [0, 1]$. Each agent i has a stochastic policy π^i conditioned on its observations o_i or action-observation history $\tau^i \in (\mathcal{Z} \times \mathcal{A}^i)$. The discounted return is $G_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$, where r_t is the joint reward at step t. The joint policy π induces a value function $V^\pi(s_t) = \mathbb{E}[G_t \mid s_t]$ and an action-value function $Q^\pi(s_t, \mathbf{a}_t) = \mathbb{E}[G_t \mid s_t, \mathbf{a}_t]$, where $\mathbf{a}_t$ is the joint action. The advantage function is then $A^\pi(s_t, \mathbf{a}_t) = Q^\pi(s_t, \mathbf{a}_t) - V^\pi(s_t)$.

2.1 Policy Optimization

We use policy optimization to maximize the expected discounted return. Given policy π_θ parameterized by θ, the surrogate policy optimization objective is [34]:

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \tag{1}$$

where $\hat{A}_t$ is the advantage function estimator [33] at time step t and the expectation $\hat{\mathbb{E}}_t[\ldots]$ indicates the empirical average over a finite batch of samples. In practice, we use the clipped PPO objective [34] to limit the step size for stable updates. For the policy π_θ, we can either condition on states using a feed-forward network like multi-layer perceptron (MLP), or condition on the full history using a recurrent neural network such as an LSTM [11] or GRU [5]. In the context of centralized training but decentralized execution, a common strategy is to share the policy parameters between agents that are homogeneous [3, 8]. With shared rewards, COMA [6] critic can be useful for better credit assignment and can be easily combined with our proposed approach. However, these approaches do not model the coordination structure. Wei et al. [43] investigate relative overgeneralization in continuous action multi-agent tasks and show improvement over MADDPG [21]. OroojlooyJadid and Hajinezhad [27] provide a general overview of cooperative MARL.

2.2 Coordination Graphs

For several multi-agent domains, the outcome of an agent's action often depends only on a subset of other agents in the domain. This *locality of interaction* can be encoded in the form of a coordination graph (CG) [7]. A CG is often represented as an undirected graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$ and contains a vertex $v_i \in \mathcal{V}$ for each agent i and a set of undirected edges $\{i, j\} \in \mathcal{E}$ between vertices v_i and v_j. Guestrin et al. [7] use this CG to induce a factorization of an action-value function into *utility functions* f^i and *payoff functions* f^{ij}:

$$q^{\text{CG}}(s_t, a) = \sum_{v^i \in \mathcal{V}} f^i(a^i \mid s_t) + \sum_{\{i,j\} \in \mathcal{E}} f^{ij}(a^i, a^j \mid s_t) \tag{2}$$

Guestrin et al. [7] and Vlassis et al. [42] draw on the connections with maximum a posteriori (MAP) estimation techniques in probabilistic inference to compute the joint action from such factorizations; resulting into algorithms like Variable Elimination and Max-Plus. Kok and Vlassis [18] explored their use in the context of tabular MARL. Deep Coordination Graphs (DCG) [3] extended these ideas of factoring the joint value function of all agents according to a static coordination graph into payoffs between pairs of agents to deep MARL. They did so by estimating the payoff functions using neural networks and using message passing based on Max-Plus [42] along the coordination graph to maximize the value function, allowing training of the value function end-to-end.

In this work, however, we forgo explicitly computing the joint action through inference over factored representation with a *given* coordination graph. Instead, we use attention to learn the appropriate agent observation-dependent coordination graph structure with soft edge weights and then use message passing in a graph neural network to compute appropriate values or actions for the agents, such that full the computation graph remains differentiable.

2.3 Self-attention

Self-attention mechanism [4] emerged from the natural language processing community. It is used to relate different positions of a single sequence. The difference between self-attention and standard attention is that self-attention uses a single sequence as both its source and target sequence. It has been shown useful in image caption generation [19, 46] and machine reading [4, 45].

The attention mechanism has also been adopted recently for MARL. The relations between a group of agents can be learned through attention. Iqbal and Sha [12] use attention to extract relevant information of each agent from the other agents. Jiang and Lu [14] use self-attention to learn when to communicate with neighboring agents. Wright and Horowitz [44] use self-attention on the policy level to differentiate different types of connections between agents. Jiang et al. [13] use multi-head dot product attention to compute interactions between neighbouring agents for the purpose of enlarging agents' receptive fields and extracting latent features of observations. We use self-attention to learn the attention weights between agents, and use the attention weights to form a "soft"-edged coordination graph instead of edges with binary weights.

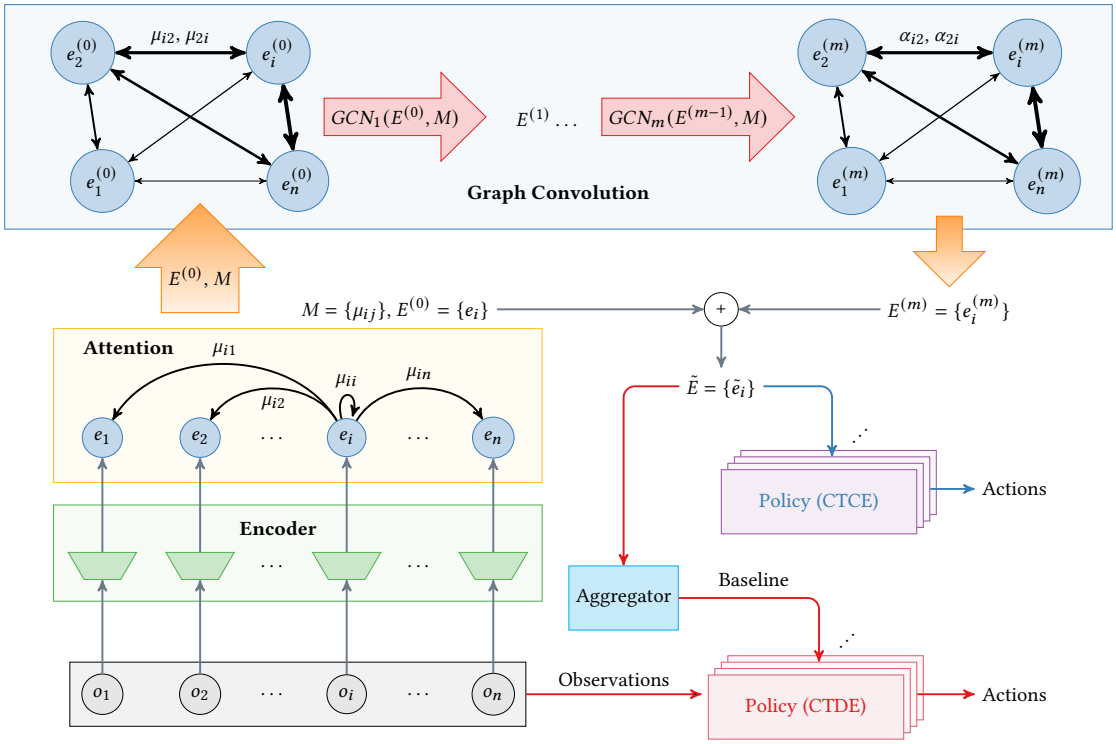

Figure 1: Network architecture of DICG. It can be used for either a centralized-training-centralized-execution (CTCE) approach or as a centralized-training-decentralized-execution (CTDE) approach. The blue arrows indicate the CTCE approach. The DICG module serves as a joint observation encoder. We use the integrated observations $\tilde{E}$ to directly obtain actions for agents through a parameter sharing policy. The baselines in CTCE are estimated by a concatenation of raw observations. The red arrows indicate the CTDE approach. We pass the integrated observations $\tilde{E}$ through an aggregator network to estimate a centralized baseline. We then use the baseline to compute the advantage to guide policy optimization.

2.4 Graph Neural Networks

Several frameworks have been proposed to extract locally connected features from arbitrary graphs [16, 39]. Given a graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$, a graph convolutional network (GCN) takes as input the feature matrix that summarizes the attributes of each node $v_i \in \mathcal{V}$ and outputs a node-level feature matrix. This is similar to how a convolution operation across local regions of the input produces feature maps in CNNs. MAGnet learns relation weights through a loss function based on heuristic rules in the form of a relevance graph [23]. Deep relational RL embeds multi-head dot-product attention as relational block into graph neural networks to learn pairwise interaction representation of a set of entities in the agent's state [47]. Recently, Liu et al. [20] combined a two-stage attention network with a graph neural network for communication between the agents to achieve state-of-the-art performance on the traffic junction domain with curriculum training [2].

3 APPROACH

Instead of the standard approach of learning the binary weights of the edges in a coordination graph, we use self-attention to learn the relation between agents and use the attention weights as soft edges of a coordination graph. These soft edges form an implicit coordination graph representing elements of its adjacency matrix, $M \in \mathbb{R}^{n \times n}_{>0}$. We use self-attention to avoid building coordination graphs using hard-coded or domain-specific heuristics so that our approach is applicable to more abstract multi-agent domains. Moreover, maintaining differentiability is difficult with binary connections. We use attention to implicitly represent the edge weights as the strength of the connection between agents to obtain the graphs's adjacency matrix. We then apply graph convolution [16] with this adjacency matrix to integrate information across agents. We use graph convolution because it is an efficient and differentiable way to pass information along the graph. With the integrated information, we can either use it as observation embeddings to directly obtain actions or use it to estimate baselines for advantage estimation during policy optimization. In summary, the DICG module consists of an encoder, an attention module, and a graph convolution module with the architecture outlined in Fig. 1.

In detail, we first pass n observations $\{o_i\}_{i=1}^n$ of the n agents through a parameter sharing encoder parameterized by θ_e. The encoder outputs n embedding vectors $\{e_i\}_{i=1}^n$, each with size d:

$$e_i = \text{Encoder}(o_i; \theta_e), \text{ for } i = 1, \ldots, n. \tag{3}$$

We then compute the attention weights from agent i to j using these embeddings as:

$$\mu_{ij} = \frac{\exp(\text{Attention}(e_i, e_j, W_a))}{\sum_{k=1}^{n} \exp(\text{Attention}(e_i, e_k, W_a))}. \quad (4)$$

where the attention module is parameterized by W_a, which is a trainable $d \times d$ weight matrix. The attention score function we adopt is general attention [22]:

$$\text{Attention}(e_i, e_j, W_a) = e_j^\top W_a e_i. \quad (5)$$

The attention module is also parameter shared among agents. We use these attention weights to form an $n \times n$ positive real valued adjacency matrix M with $M_{ij} = \mu_{ij}$, which encodes the implicit coordination graph. Since we apply soft-max for attention weights, we have $\sum_{j=1}^{n} \mu_{ij} = 1$.

We stack the embeddings to form an $n \times d$ feature matrix E with the ith row being the embedding $e_i^\top$. We denote the E before any graph convolution operations as $E^{(0)}$. With the soft adjacency matrix M and the feature matrix $E^{(0)}$, we can apply graph convolution to perform message passing and information integration across all agents. In the fast approximate GCN by Kipf and Welling [16], a graph convolution layer is

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{M} \tilde{D}^{-\frac{1}{2}} H^{(l)} W_c^{(l)}\right), \quad (6)$$

where $H^{(l)}$ is the feature matrix of convolution layer l. In our case, $H^{(0)} = E^{(0)} = [e_1^\top; e_2^\top; \ldots; e_n^\top]$. Diagonal entries of M are already positive from the self-attention weights. Therefore, unlike Kipf and Welling [16], we do not need to add an identity matrix for non-zero self-connections and can set $\tilde{M} = M$. By their definition, $\tilde{D}_{ii} = \sum_{j=1}^{n} \tilde{M}_{ij} = \sum_{j=1}^{n} \mu_{ij} = 1$, i.e. $\tilde{D}$ is simplified to an identity matrix, I_n. The $d \times d$ matrix $W_c^{(l)}$ is a trainable weight matrix associated with layer l, and σ is a non-linear activation.

Replacing $\tilde{M}$ with attention weights M and $\tilde{D}$ with I_n, the graph convolution operation simplifies to

$$H^{(l+1)} = \sigma\left(M H^{(l)} W_c^{(l)}\right). \quad (7)$$

This graph convolution operation is performed m times. We denote the output of mth layer $H^{(m)}$ as $E^{(m)}$, which is a stack of integrated embeddings.

We then use a residual connection [10] between $E^{(0)}$ and $E^{(m)}$ to obtain the final embedding matrix $\tilde{E} = E^{(0)} + E^{(m)}$. The residual connection is designed to assist gradient flow through the attention module and the encoder. The final embedding matrix $\tilde{E}$ consists of a stack of integrated embeddings $\{\tilde{e}_i\}_{i=1}^{n}$. Finally, there are two ways to use the embedding matrix $\tilde{E}$:

(a) DICG-CE: If full communication is allowed between agents, we can use the DICG module in a *centralized-training-centralized-execution* (CTCE) framework to communicate information between the agents. The output from the DICG module, $\tilde{E}$, integrates relevant information across all agents. The corresponding embedding $\tilde{e}_i$ (or its history for recurrent neural networks) can be passed through a separate or parameter-shared policy network to obtain actions for each agent (indicated with blue arrows in Fig. 1). We can then use standard actor-critic methods to train the network end-to-end. As our experiments demonstrate, embeddings obtained from DICG are superior to simply concatenating the raw observations and passing through an MLP network due to the implicit coordination structure reasoning for information integration.

(b) DICG-DE: If full communication is not allowed between agents, we can still use the DICG module to facilitate better coordination. Following the principles of *centralized-training-decentralized-execution* (CTDE), we can use the output of the DICG module, $\tilde{E}$, in a centralized critic. We pass $\tilde{E}$ through another MLP, which we refer to as an aggregator network, to estimate the centralized baseline (indicated with red arrows in Fig. 1). Separate or parameter shared policy networks can be trained for each agent using standard actor-critic methods [1, 6, 8], except using the DICG centralized baseline for advantage computation. During execution the critic is no longer required and the agents can act independently. Again, we find that the embeddings obtained by DICG are superior to simply concatenating the raw observations and passing through an MLP network due to its implicit reasoning about the dynamic coordination structure.

Key Differences from Related Approaches: Each component module like self-attention and graph convolutions that are key to our approach have been previously used in the literature. Iqbal and Sha [12] use attention mechanism in their critic to dynamically attend to other agents. However, they do not use the concept of a coordination graph and process those embeddings with a graph neural network. Moreover, they experiment with fairly small and simple particle environments. Wright and Horowitz [44] use attention mechanism in their actor to aggregate information from other agents sharing some similarities with DICG-CE. They do not use attention to create a coordination graph structure that can be used by a graph neural network to process the observation embeddings. Moreover, they are focused on a simple simulation of merging vehicles. Jiang et al. [13] use attention mechanism to obtain a relational kernel for use in a graph neural network. Moreover they are focused on value based methods and use Deep Q-learning. They do not use the attention weights to create a coordination graph. Rather, they hand craft the adjacency matrix used by the graph neural network. They experiment with particle environments with simple observations. Ryu et al. [31] use both an attention mechanism and graph neural networks. However, they focus on multi-group settings and consider the relationships between an individual agent with groups of other agents to come up with a hierarchical representation. Again, they only experiment with simple particle environments.

The key contribution of this work is to take inspiration from the coordination graph literature and combine these various components in a specific way to design a fully differentiable architecture. As we'll see in the next section, this design leads to strong performance improvements in a variety of complex multi-agent domains.

4 RESULTS

We present experiments applying DICG to three environments (shown in Fig. 2): predator-prey, StarCraft II Multi-agent Challenge (SMAC) [32], and traffic junction [36]. These environments require coordination to achieve high returns, i.e. agent interactions are not so sparse that totally decentralized approaches with partial observability can achieve high returns. They are also sufficiently complex that fully centralized approaches quickly become intractable. We compare our approach against two standard actor-critic baseline

(a) Predator-Prey (b) SMAC (3s_vs_5z) [40, 41] (c) Traffic Junction (hard mode) [20, 36]

Figure 2: Experiment environments. (a) Predators are marked in blue, and prey are marked in red. The cyan grids are the capture range of predators. An example of successful capture is predator 2 and 6 capturing prey 3. An example of a single-agent capture attempt that will cause penalty is predator 3 capturing prey 8 alone. (b) SMAC scenario 3s_vs_5z. (c) An illustration of the hard mode traffic junction environment with 18×18 grids and 4 two-way routes.

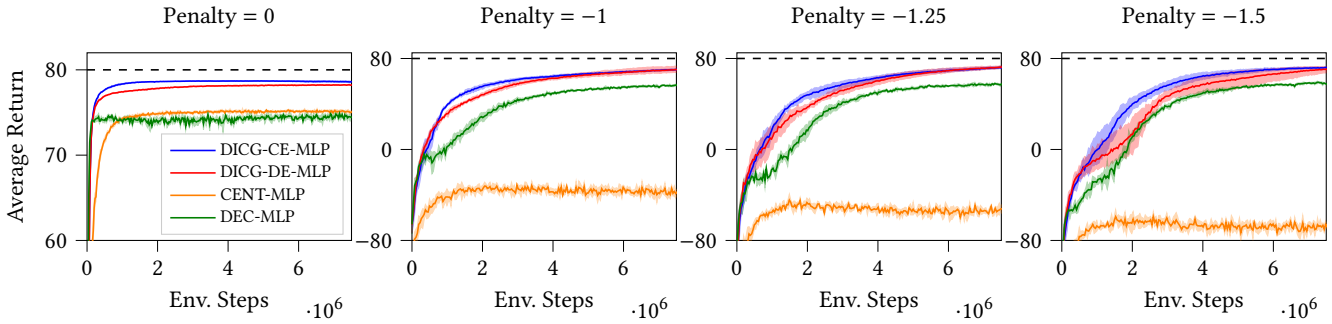

Figure 3: Average return of predator-prey with increasing penalty for single-agent capture attempt.

approaches: 1) fully decentralized architecture (referred to as DEC) with only local observation as input to policy; and 2) centralized architecture (referred to as CENT) with a direct concatenation of all observations as input to policy. Due to the dimensionality of the action space, we still need to factorize the policy [8, 36] in the centralized architecture so that we output separate action distributions for each agent. They both use a centralized critic with full observation to create a baseline estimate, and they use PPO for policy optimization [1]. Benchmarking against them can justify the effectiveness of coordination learning and information integration of DICG. We also compare with results reported by other MARL approaches. All results are averaged over 5 seeds. Environment and network details are in Appendix A. Code is available here[1].

4.1 Predator-Prey

We use an environment similar to that described by Böhmer et al. [3]. The environment consists of a 10×10 grid world with 8 predators and 8 prey. We control the movement of predators to capture prey. The prey move by hard-coded and randomized rules to avoid predators. If a prey is captured, the agents receives a reward of 10. However, the environment penalizes any single-agent attempt to capture prey with a negative reward p; at least two agents are required to be present in the neighboring grid cells of a prey for a successful capture. We set the episode length to 200 steps, and impose a step cost of -0.1. Cooperation is necessary to achieve a high return in this environment. We use an MLP policy for all the architectures. Fig. 2a shows the environment and illustrations of a successful capture and a single-agent capture attempt to be penalized. A typical relative overgeneralization pathology could arise from imposing the single-agent capture attempt and a lack of proper coordination is that all agents crowd to a corner of the grids, failing to explore strategies.

Figure 3 shows the average return for test episodes for varying penalties p averaged over 5 runs. Overall, DICG performs the best and solves relative overgeneralization with its implicit coordination. Without any penalty ($p = 0$), fully centralized (CENT-MLP) and fully decentralized (DEC-MLP) architectures have similar performance. However, they require more steps to capture all prey than the DICG approaches. As we increase the penalty, only DICG is able to reliably and quickly converge to optimality. DEC-MLP has a characteristic slowdown in the learning curves before it is able to approach DICG. It finally converges to suboptimal performance with relative overgeneralization due to the lack of coordination across agents. The fully centralized approach CENT-MLP can only achieve positive returns in non-penalized setting. With a negative penalty, CENT-MLP cannot learn to capture prey appropriately due to two reasons: 1) simple concatenation of observations in CENT-MLP leads to a large joint observation space, and 2) concatenation of observations is not an efficient way to integrate information and learn coordination across agents.

[1]Code of this work: https://github.com/sisl/DICG

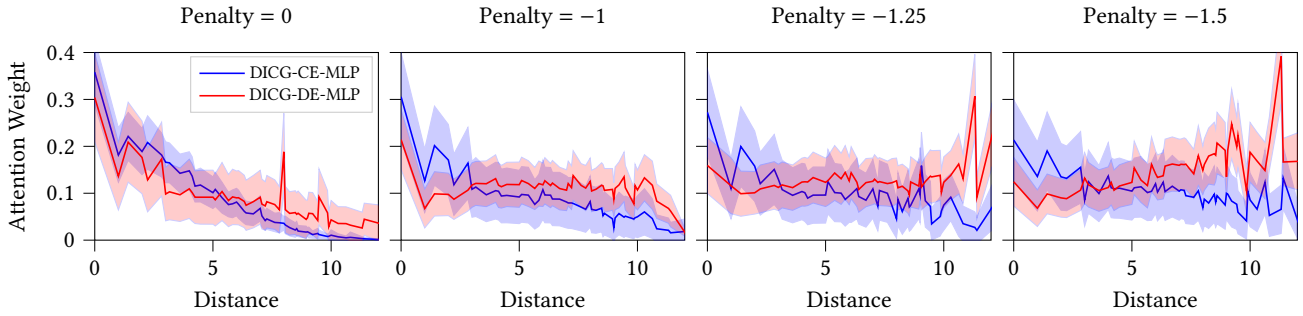

Figure 4: Attention weights, i.e. graph edge strengths of DICG under different distance between two agents (average of 5 seeds). Zero distance indicates an agent's attention towards itself. As the penalty for single-agent capture attempts increases, more attention is paid to farther agents.

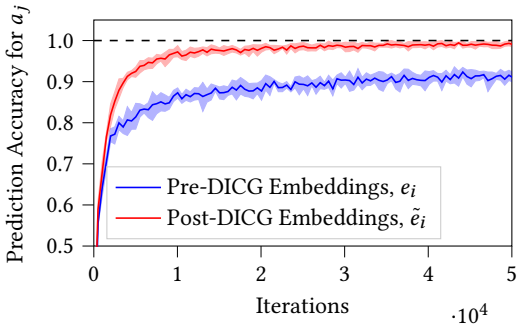

Figure 5: Predicting agent j's actions (a_j) using agent i's pre-DICG (e_i) and post-DICG embeddings ($\tilde{e}_i$). Data are collected from the evaluation trajectories of five random pairs of agents (i, j).

Analyzing Implicit Coordination Graph

We examine the effects of the implicit CG in the following experiments.

Semantics of implicit CG: To understand how the DICG learns to coordinate, we perform attention weight analysis, i.e. we study the strength of soft edges of the implicit coordination. A heuristic of what affects the strength of connection between agents is the distance between agents. Figure 4 shows the relationship between attention weight and distance between agents learned by DICG. Zero distance corresponds to the attention weight of an agent to itself. As the penalty increases, agents tend to increase the attention weight towards agents further away. This phenomenon coincides with the coordination requirements imposed by the increase of penalty that agents should pay more attention to form groups with each other to capture prey as a team than moving alone.

Effectiveness of GCN at information integration: To examine whether information is effectively integrated across agents along the implicit coordination graph, we design an experiment to predict agent j's actions a_j only using agent i's information with $i \neq j$. This is formulated as a supervised learning problem: $\hat{a}_j = f(x_i; \phi)$ with loss $L = CrossEntropy(\hat{a}_j, a_j)$. Theoretically, with a finite amount of data, if x_i is more correlated with a_j, the classifier $f(\cdot; \phi)$ can produce a higher prediction accuracy. Therefore, we set x_i to the pre-DICG embeddings e_i and the post-DICG embeddings $\tilde{e}_i$. We use a simple MLP classifier parameterized by ϕ, with a single 64-unit hidden layer and ReLU as activation. Results of five random pairs of (i, j) are shown in Fig. 5. The post-DICG embeddings can predict other agent's actions with a higher accuracy than the pre-DICG embeddings. We can interpret this as the DICG architecture allowing individual agents to learn other agents' intentions.

4.2 StarCraft II Multi-agent Challenge (SMAC)

StarCraft II, and the StarCraft II Learning Environment (SC2LE), has provided an environment for some of the most important reinforcement learning work in recent years [40, 41]. However most of this work has resided in the single-agent domain where reinforcement learning is used to train a single, centralized decision-making agent how to play the entire StarCraft II game involving the control of a large number of units within the game. The StarCraft Multi-Agent Challenge (SMAC) extends SC2LE by providing a collection of reinforcement learning benchmarks designed specifically for *multi-agent environments* [32].

Within SMAC, each unit is controlled by its own separate learning agent whose actions must be conditioned on local observations and not the global game state. SMAC scenarios are designed to explore *micromanagement*, e.g. precise movements and coordinated targeting, of relatively small groups of units. A scenario where 3 agent controlled stalkers combating with 5 computer controlled zealots is shown in Fig. 2b. In each episode, positive rewards are given for the positive health point difference between the controlled agent team and the computer controlled opponent team, otherwise, the agent team receive zero or negative rewards. Large positive terminal reward is given for winning the episode by eliminating the opponents (zero for being eliminated).

We test DICG on SMAC's asymmetric and "micro-trick" scenarios such as `8m_vs_9m`, `3s_vs_5z`, and `6h_vs_8z` [32]. The opponent AI difficulty is set to "hard", "hard", and "super hard", respectively. We use an LSTM policy for all architectures in SMAC. Results of SMAC are in Fig. 6. In `8m_vs_9m`, DICG-DE-LSTM outperforms all the other approaches; DICG-CE-LSTM and DEC-LSTM have similar performance; CENT-LSTM performs the worst. In `3s_vs_5z`,

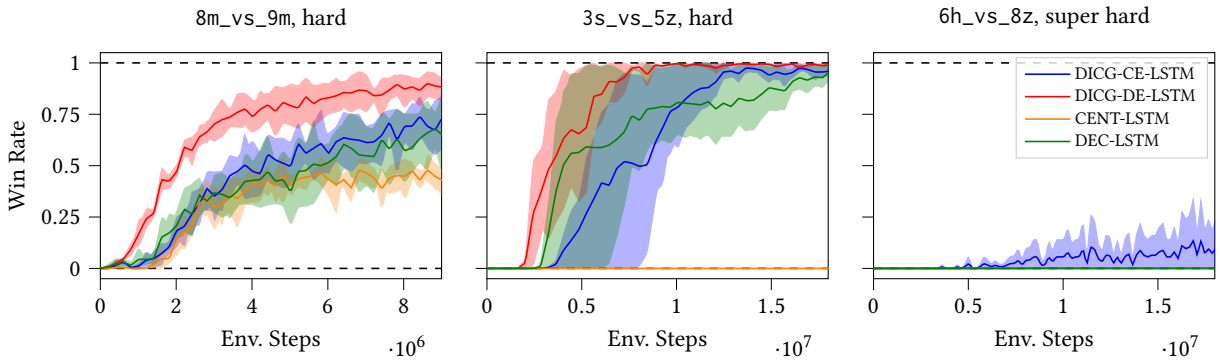

Figure 6: Evaluation win rate during training in SMAC maps.

Table 1: SMAC win rate comparison.

Approach	8m_vs_9m	3s_vs_5z	6h_vs_8z
DCG [3]	55 ± 10%	85 ± 3%	**10** ± 5%
VDN [37]	49 ± 5%	72 ± 10%	0
QMIX [30]	60 ± 11%	95 ± 1%	5 ± 5%
CENT-LSTM	42 ± 6%	0	0
DEC-LSTM	65 ± 16%	94 ± 5%	0
DICG-CE-LSTM	72 ± 11%	96 ± 3%	9 ± 9%
DICG-DE-LSTM	**87** ± 6%	**99** ± 1%	0

Table 2: Traffic junction success rate comparison.

Approach	Easy	Medium	Hard
CommNet [36]	93.0 ± 4.2%	54.3 ± 14.2%	50.2 ± 3.5%
IC3Net [35]	93.0 ± 3.7%	89.3 ± 2.5%	72.4 ± 9.6%
GA-Comm [20]	**99.7**%	**97.6**%	**82.3**%
CENT-MLP	97.7 ± 0.9%	0	0
DEC-MLP	90.2 ± 6.5%	81.3 ± 4.8%	69.4 ± 4.9%
DICG-CE-MLP	**98.1 ± 1.9**%	80.5 ± 6.8%	22.8 ± 4.6%
DICG-DE-MLP	95.6 ± 1.5%	90.8 ± 2.9%	**82.2 ± 6.0**%

DICG-DE-LSTM shows the highest and the most stable win rate, as well as the most sample efficient learning; DICG-CE-LSTM has more stable performance than DEC-LSTM, but CENT-LSTM fails to learn. From game replays, we observe that DICG learns a particular circular movement strategy in 3s_vs_5z. Due to the asymmetric setup, the 3 stalker agents controlled by DICG cannot overcome the opposing 5 zealots with force. The DICG agents learn to split into two groups, each attracting a number of opponents. Each group moves along the edges of the square map in a circle, and damages opponents using a learned hit-and-run tactic with their high speed. When the opponents are sufficiently weak, the two groups reunite to eliminate the opponents. This highly coordinated tactic demonstrates the effectiveness of DICG. In 6h_vs_8z, a very difficult map, DICG-CE-LSTM is the only approach to win against the opponent AI.

A comparison of SMAC win rate under the same difficulty setting with DCG [3], VDN [37] and QMIX [30] is in Table 1. DICG outperforms DCG and others *without using the privileged state information* in 8m_vs_9m and 3s_vs_5z, and having comparable but noisier win rate for 6h_vs_8z. DICG outperforms VDN and QMIX in all the mentioned scenarios requiring high coordination. In many multi-agent tasks, we do not have access to privileged full state information even during training and our algorithm needs to just work with observations. DICG demonstrates the advantage of integrated information to prevent relative overgeneralization in such scenarios.

4.3 Traffic Junction

The traffic junction environment, introduced by Sukhbaatar and Fergus [36], is a multi-agent environment where cars are randomly added to traffic junctions with pre-assigned routes who need to avoid collision with each other and reach their destinations. Each agent only has a limited vision of one grid from itself. The reward function consists of a collision penalty to discourage collision and a step cost to discourage congestion. There are easy, medium, and hard difficulty modes in the traffic junction environment. The environment configurations are adopted from Singh et al. [35]. Figure 2b shows the traffic junction environment in hard mode. We use MLP policies for the traffic junction environment.

The results are in Fig. 7 and a comparison with other baselines using the same environment configurations is in Table 2. DICG-CE-MLP performs better than DICG-DE-MLP in easy mode. CENT-MLP also performs well in easy mode. This is because easy mode has fewer number of agents and small observation space. Centralized execution can outperform decentralized approaches in relatively small domains. However, DICG-CE-MLP has the privilege of more efficient agent information integration over CENT-MLP.

In medium and hard mode, where the number of agents and the dimension of observation space increases, centralized approaches fail to perform well. DICG-DE-MLP outperforms decentralized and centralized baselines in medium and hard mode. Even though we *do not use any curriculum* [2] based training, DICG's performance is close to that of GA-Comm [20] which employs curriculum learning. Note that the results of GA-Comm fall within the uncertainty range of our results in easy and hard mode. We provide brief descriptions of these baselines in Appendix B.

Ablation. To examine the effectiveness of various components in the DICG architecture, we perform ablation experiments by adjusting the attention module and the graph convolution module. Two variants of DICG-DE-MLP are designed:

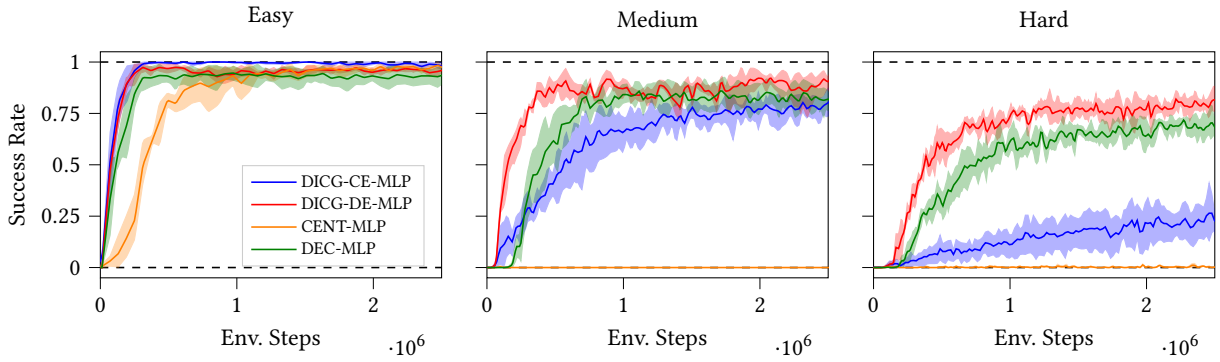

Figure 7: Evaluation success rate during training of different difficulty levels in traffic junction.

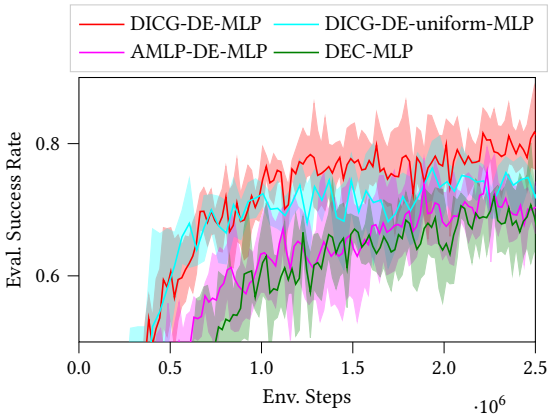

Figure 8: Zoomed-in ablation experiment results in hard mode traffic junction environment (average of 5 random seeds).

(1) Replacing learned attention weights with uniform attention weights, i.e. for n agents, the attention weights become $1/n$. We denote this variant as DICG-DE-uniform-MLP.
(2) Replacing the graph convolution module with MLP. We concatenate the embeddings and the attention weights of all the agents and feed the concatenation through an MLP to estimate the baseline. We denote this variant as AMLP-DE-MLP.

We test the variants in the hard mode traffic junction environment where the number of agents is large. We expect the attention module and graph convolution modules to play relatively more important roles in coordinating agents. The results are shown in Fig. 8 (averaged over 5 random seeds) in a zoomed-in view. AMLP-DE-MLP shows similar performance as DEC-MLP. This indicates that MLP cannot integrate agents' information as effectively as graph convolution. DICG-DE-uniform-MLP has slightly worse performance than DICG-DE-MLP. This indicates that learned attention weights can better emphasize coordination among agents than uniformly spreading attention. Table 3 compares the success rate with other approaches.

Table 3: Traffic junction success rate comparison with ablation experiments in hard mode.

Approach	Success Rate
CommNet [36]	50.2 ± 3.5%
IC3Net [35]	72.4 ± 9.6%
GA-Comm [20]	**82.3%**
CENT-MLP	0
DEC-MLP	69.4 ± 4.9%
DICG-CE-MLP	22.8 ± 4.6%
DICG-DE-MLP	**82.2 ± 6.0%**
DICG-DE-uniform-MLP	72.0 ± 1.3%
AMLP-DE-MLP	70.3 ± 3.8%

5 CONCLUSIONS AND FUTURE WORK

In this work, we present the DICG architecture that uses self-attention to implicitly build a coordination graph and then perform message passing with graph convolution layers to compute appropriate baseline values (DE) or actions (CE) for the agents while keeping the computational graph differentiable. We demonstrate that DICG solves the relative overgeneralization pathology in predator-prey tasks, as well as various MARL baselines including the challenging StarCraft II micromanagement tasks and traffic junction tasks. DICG is shown to be an effective architecture for implicitly and dynamically learning multi-agent coordination that achieves an appropriate tradeoff between fully centralized and fully decentralized approaches. For future work, we aim to improve the sample efficiency of DICG. To achieve this, we may incorporate the DICG architecture into off-policy learning algorithms such as deep Q-learning [25] and soft actor-critic [9, 43].

ACKNOWLEDGMENTS

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

REFERENCES

[1] Ross E Allen, Javona White Bear, Jayesh K Gupta, and Mykel J Kochenderfer. 2019. Health-Informed Policy Gradients for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:1908.01022* (2019).

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *International Conference on Machine Learning (ICML)*. 41–48.

[3] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. 2019. Deep Coordination Graphs. *arXiv preprint arXiv:1910.00091* (2019).

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733* (2016).

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[6] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[7] Carlos Guestrin, Daphne Koller, and Ronald Parr. 2002. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems (NeurIPS)*. 1523–1530.

[8] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Springer, 66–83.

[9] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning (ICML)*. 1861–1870.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[12] Shariq Iqbal and Fei Sha. 2018. Actor-attention-critic for multi-agent reinforcement learning. *arXiv preprint arXiv:1810.02912* (2018).

[13] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. 2020. Graph Convolutional Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=HkxdQkSYDB

[14] Jiechuan Jiang and Zongqing Lu. 2018. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems (NeurIPS)*. 7254–7264.

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[17] Jelle R Kok, Eter Jan Hoen, Bram Bakker, and Nikos Vlassis. 2005. Utile coordination: Learning interdependencies among cooperative agents. In *IEEE Symposium on Computational Intelligence and Games*. 29–36.

[18] Jelle R Kok and Nikos Vlassis. 2004. Sparse cooperative Q-learning. In *International Conference on Machine Learning (ICML)*. ACM, 61.

[19] Xihui Liu, Hongsheng Li, Jing Shao, Dapeng Chen, and Xiaogang Wang. 2018. Show, tell and discriminate: Image captioning by self-retrieval with partially labeled data. In *European Conference on Computer Vision (ECCV)*. 338–354.

[20] Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. 2019. Multi-Agent Game Abstraction via Graph Attention Neural Network. *arXiv preprint arXiv:1911.10715* (2019).

[21] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS)*. 6379–6390.

[22] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[23] Aleksandra Malysheva, Tegg Taekyong Sung, Chae-Bong Sohn, Daniel Kudenko, and Aleksei Shpilman. 2018. Deep multi-agent reinforcement learning with relevance graphs. *arXiv preprint arXiv:1811.12557* (2018).

[24] Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. 2012. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *The Knowledge Engineering Review* 27, 1 (2012), 1–31. https://doi.org/10.1017/S0269888912000057

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[26] Frans A Oliehoek and Christopher Amato. 2016. *A concise introduction to decentralized POMDPs*. Springer.

[27] Afshin OroojlooyJadid and Davood Hajinezhad. 2019. A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963* (2019).

[28] Liviu Panait, Sean Luke, and R Paul Wiegand. 2006. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation* 10, 6 (2006), 629–645.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[30] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. 4295–4304.

[31] Heechang Ryu, Hayong Shin, and Jinkyoo Park. 2020. Multi-Agent Actor-Critic with Hierarchical Graph Attention Network.. In *AAAI Conference on Artificial Intelligence (AAAI)*. 7236–7243.

[32] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The Starcraft multi-agent challenge. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2186–2188.

[33] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[35] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. 2018. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755* (2018).

[36] Sainbayar Sukhbaatar and Rob Fergus. 2016. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2244–2252.

[37] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, and Thore Graepel. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2085–2087.

[38] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning (ICML)*. 330–337.

[39] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations (ICLR)*.

[40] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[41] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. 2017. Starcraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* (2017).

[42] Nikos Vlassis, Reinoud Elhorst, and Jelle R Kok. 2004. Anytime algorithms for multiagent decision making using coordination graphs. In *IEEE International Conference on Systems, Man and Cybernetics*. 953–957.

[43] Ermo Wei, Drew Wicke, David Freelan, and Sean Luke. 2018. Multiagent soft Q-learning. In *AAAI Spring Symposium*.

[44] Matthew A Wright and Roberto Horowitz. 2019. Attentional Policies for Cross-Context Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:1905.13428* (2019).

[45] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. QANet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541* (2018).

[46] Jun Yu, Jing Li, Zhou Yu, and Qingming Huang. 2019. Multimodal transformer with multi-view visual representation for image captioning. *IEEE Transactions on Circuits and Systems for Video Technology* (2019).

[47] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. 2018. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830* (2018).

A EXPERIMENT DETAILS

Table 4: PPO [34] and optimizer parameters.

Parameter	Value
Likelihood ratio clip range	0.2
Optimizer	Adam [15] (PyTorch [29])
Policy learning rate	3×10^{-4}
GAE-λ	0.97
Discount γ	0.99
Trajectory length	Environment dependent
Batch size	Environment dependent
Policy entropy coefficient	Environment dependent

This section describes the environment settings, hyperparameters of the policy optimization algorithm, and network sizes. Table 4 shows the hyperparameters used for PPO shared by all experiments. Batch size is in the unit of environment steps. All the nonlinear activation functions in this work are hyperbolic tangent (tanh). The aggregator network used for DICG-DE approach is a single layer linear mapping from the embedding dimension to one dimension. Environment specific detailed settings are listed in following subsections.

Code of this work is available at this link[2].

A.1 Predator-Prey

We use the implementation of predator-prey from open source `ma-gym`.[3] The original implementation does not include other predators in the observation of a predator. We modify the environment by making the other agents visible in the field of view of a predator. A predator has a field of view of 5×5 grids with itself at center.

We train with network settings shown in Table 5. The maximum number of steps in an episode is set to 200. We use a batch size of 6×10^4 and a policy entropy coefficient of 0.1. Except for DICG-DE-MLP, all other approaches use an MLP baseline (critic). Two layers of GCN are used for DICG.

A.2 StarCraft II Multi-agent Challenge (SMAC)

We use the open source SMAC implementation by Samvelyan et al. [32]. We use the SMAC implementation[4] by Samvelyan et al. [32]. The number of controlled agents and opponents, their unit types as well as maximum time steps of an episode are redefined by the authors of the micromanagement scenarios. The detailed map configurations and the PPO parameters we use are listed in Table 6. For the network architecture, two layers of GCN are used for DICG. Other network architecture details are listed in Table 7. Except for DICG-DE-LSTM, all other approaches use an MLP feature baseline (critic).

A.3 Traffic Junction

We use the open source traffic junction environment implementation by Sukhbaatar and Fergus [36]. We use the maximum agent add rate (the most difficult task setting) from their curriculum framework and skip curriculum training. We use a policy entropy coefficient of 0.02. Except for DICG-DE-MLP, all other approaches use an MLP baseline (critic). The detailed environment settings are listed in Table 8. The vision range of cars (agents) is set to one grid. The network configurations we used are in Table 9.

[2]Code of this work: https://github.com/sisl/DICG

[3]https://github.com/koulanurag/ma-gym/tree/master/ma_gym/envs/predator_prey

[4]https://github.com/oxwhirl/smac

B DESCRIPTIONS FOR COMPARED BASELINE APPROACHES

We compared with the following baseline approaches:

- Value-decomposition Networks (VDN) [37]: Value-based off-policy method that factorizes the joint value function as a sum of individual agent value functions.
- QMIX [30]: Value-based off-policy method that factorizes the joint value function as a monotonic function of the individual agent-value functions.
- Deep Coordination Graphs (DCG) [3]: Value-based off-policy method to learn factored value representations based on predetermined static coordination graph.
- CommNet [36]: A multi-agent communication model uses multi-step centralized and aggregated (sum and mean) communication channels to share information among agents. This model increases the receptive field of agents through communication. An on-policy method.
- IC3Net [35]: An extension of CommNet with more sophisticated communication model. An on-policy method.
- GA-Comm [20]: An on-policy communication modeling method. It first uses hard attention to select which agents to communicate with, then uses soft attention and GCN to get contribution of other agents.

Table 5: Predator-prey network architectures.

Approach	DICG encoder sizes	DICG embedding size	MLP policy sizes	Baseline (critic) sizes
DICG-CE-MLP	[128]	64	[128, 64, 32]	[64, 64, 32]
DICG-DE-MLP	[128]	64	[128, 64, 32]	[64, 64, 32]
DEC-MLP	N/A	N/A	[128, 64, 32]	[64, 64, 32]
CENT-MLP	N/A	N/A	[512, 128, 64]	[64, 64, 32]

Table 6: SMAC scenario (map) information and training settings.

Map	Difficulty	Controlled agents	Opponents	Max steps	Batch size	Policy entropy coeff.
`8m_vs_9m`	Hard	8 marines	9 marines	120	8×10^4	0.01
`3s_vs_5z`	Hard	3 stalkers	5 zealots	250	6×10^4	$0.025/0.1^*$
`6h_vs_8z`	Super hard	6 hydralisks	8 zealots	150	6×10^4	$0.025/0.1^*$

* For DICG-DE-LSTM.

Table 7: SMAC network architectures.

Approach	DICG encoder sizes	DICG embedding size	Policy encoder sizes	LSTM hidden size	Baseline (critic) sizes
DICG-CE-LSTM	[128]	128	[128]	64	[64, 64, 64]
DICG-DE-LSTM	[128]	64	[128]	64	[64, 64, 64]
DEC-LSTM	N/A	N/A	[128]	64	[64, 64, 64]
CENT-LSTM	N/A	N/A	[256]	128	[64, 64, 64]

Table 8: Traffic junction environment configurations [35].

Difficulty	# roads	# directions	Road dim.	# junctions	$n_{\max}$	Car add rate	Max steps	Batch size
Easy	2	1	7	1	5	0.3	20	6×10^4
Medium	4	2	14	1	10	0.2	40	6×10^4
Hard	8	2	18	4	20	0.05	60	8×10^4

Table 9: Traffic junction network architectures.

Approach	DICG encoder sizes	DICG embedding size	MLP policy sizes	Baseline (critic) sizes
DICG-CE-MLP	[128]	128	[128, 64, 32]	[64, 64, 64]
DICG-DE-MLP	[128, 128]	128	[256, 128, 64]	N/A
DEC-MLP	N/A	N/A	[256, 128, 64]	[64, 64, 64]
CENT-MLP	N/A	N/A	[512, 128, 64]	[64, 64, 64]