Large Scale Coordination Transfer for Cooperative Multi-Agent Reinforcement Learning

Ethan Wang
Georgia Institute of Technology
`ewang78@cc.gatech.edu`

Binghong Chen
Georgia Institute of Technology
`binghong@cc.gatech.edu`

Le Song
Georgia Institute of Technology
`lsong@cc.gatech.edu`

Abstract

Multi-agent environments with large numbers of agents are difficult to solve due to the complexity associated with drawing sufficient samples for learning. While recent work has addressed the possibility of using transfer learning to improve sample complexities of reinforcement learning algorithms, methods for transferring knowledge in multi-agent domains across differing numbers of agents have rarely been considered. To address the bottleneck with sampling from large scale environments, we propose a joint critic structure motivated from *graph convolutional networks* and *coordination graphs* that allows for the direct transfer of parameters into environments with varying amounts of agents. We further consider *fine-tuning* the transferred policy and critic networks on the target domain and provide the motivation for doing so in cooperative environments where agent behavior is determined by a subset of the total population. Finally, we provide empirical results validating our claims on such environments, including popular multi-agent benchmark environments.

1 Introduction

Data collection for a wide variety of multi-agent environments relies on computationally expensive simulations, or in scenarios such as autonomous driving, health informatics, and robotics coordination, potentially hazardous experiments. As such, the sample complexity of reinforcement learning algorithms remains a prominent bottleneck against their applications in these systems. Since the state-action space must grow exponentially with the number of agents, popular algorithms can suffer from a curse of dimensionality termed the *curse of many agents* [10] reflected by diminishing sample efficiency as the number of agents increases. Mean-field approximation methods [24, 41, 45] as well as factorized policy [12] and value representations [35] have been proposed to counteract the exponential growth of the state-action space; however, the high computational cost of sampling from an environment with many agents and the possible consequences of iterating on real-world scenarios are still significant barriers against the use of many algorithms.

We seek to utilize *transfer learning*, a well-studied field in the domain of supervised learning [29], to reduce the need for samples in cooperative large scale environments using related environments of smaller scale. In the realm of reinforcement learning, transfer learning can take the form of *reward shaping* [2, 8, 40], where external knowledge of the reward function for the target environment is used to guide policy learning, or *learning from demonstrations* [14, 19], where expert demonstrations are used to facilitate exploration. Here, we propose a method that is most akin to *few shot learning*, where a model must learn to adapt to a target task T while training on a dataset with only a limited number of examples for T [42]. In particular, the external knowledge being transferred in this work takes the form of pre-trained policy networks and value function approximators.

Contributions: The contributions in this paper are threefold. First, we propose a joint critic structure that allows for the direct transfer of pre-trained parameters (Subsection 3.1). Commonly, the input of these critics is either the joint state or joint state-action pair, and this scales with the number of agents, preventing model parameters from being transferred between environments with varying number of agents. Secondly, we consider the possibility of *fine-tuning* (Subsection 3.2), such as in the few shot learning paradigm, across domains with differing numbers of agents using decentralized execution, a method that has not been addressed for multi-agent reinforcement learning to the best of our knowledge. Finally, we demonstrate empirically (Section 4) that this type of parameter transfer can produce the best performance on environments where individual agent behaviour is dependent on a subset of the global population. While general environments may not follow this structure, we validate our results on popular multi-agent benchmark environments.

1.1 Related Work

Single-Agent Methods: For single-agent environments, *policy transfer* refers to the use of pre-trained teacher policies from a set of source domains $\mathcal{E}_s$ that provide knowledge about some target domain E_τ. *Policy distillation* then utilizes these teacher policies, in the case of [33, 46] pre-trained on multi-task domains such as Atari 2600 games, to learn a student policy by minimizing the divergence of its action distribution with respect to the teachers'. The transfer of feature representations, termed *representation transfer*, is used in works such as [34], where *progressive neural networks* use representations from previous tasks to assist in learning a new task.

Multi-Agent Methods: *Intra-agent* transfer techniques reuse knowledge generated by an agent for new domains without requiring an explicit communication channel [6]. The authors of [5] propose a method to calculate a task mapping in an object-oriented manner, transferring value functions between tasks using this description. For environments with *sparse interactions*, where other agents effect a local agent only in certain portions of the environment, works such as [15, 51] treat the majority of the state space as a single-agent problem, reusing value functions trained on similar tasks. Similarly, the authors of [49] naturally reuse policy networks on similar road networks by using zero-shot transfer performance of learned policies as a metric to evaluate robustness for the traffic signal optimization problem.

Unlike this paper, none of the previously mentioned works consider the possibility of fine-tuning pre-trained model parameters after transferring to environments with differing amounts of *agents*. As the experiments performed in Section 4 reflect, oftentimes environment dynamics do not differ greatly with respect to the number of agents and thus can be treated analogously to the multi-task scenario, with each environment being a different "task". A similar work, [18], uses policies with graph neural network structures comparable to our proposed critic for zero-shot transfer in robotics control. However, their method assumes that an underlying communication channel is present during execution in order to aggregate features within the swarm. We focus instead on the *centralized learning with decentralized execution* paradigm, where agent policies must run independently at execution time.

2 Background

We formalize the problem as a Dec-POMDP (decentralized partially-observable Markov decision process) [28], defined by the tuple $E = (S, \{U^i\}_{i=1}^n, P, r, Z, O, n, \gamma)$ where U^i is the action space for the i^{th} agent, n is the number of agents, and $r(\mathbf{s}_t, \mathbf{u}_t)$ is the shared reward function. Here, S is the global state space, $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{u}_t) : S \times \prod_i U^i \times S \rightarrow [0, 1]$ is the transition function, and each agent receives a local observation $z_t^a \in Z$ drawn from its observation function $O : S \times \prod_i U^i \rightarrow Z$. Agents follow stochastic policies π^i conditioned on observations or an observation-action trajectory $\tau \in (Z \times U^i)$ inducing a discounted return of $G_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$. Our goal is to find an optimal joint policy π^* such that the joint action value function $Q^{\pi^*}(\mathbf{s}_t, \mathbf{u}_t) = \mathbb{E}[G_t|\mathbf{s}_t, \mathbf{u}_t] \geq Q^{\pi}(\mathbf{s}_t, \mathbf{u}_t)$ for all π.

We follow transfer learning literature and define the source domains, where we pre-train policies and critics, as $E_s = (S_s, \{U_s^i\}_{i=1}^n, P_s, r_s, Z_s, O_s, n_s, \gamma_s)$. Similarly, target domains, which will be used to evaluate zero or few-shot performance, will be defined as $E_\tau = (S_\tau, \{U_\tau^i\}_{i=1}^n, P_\tau, r_\tau, Z_\tau, O_\tau, n_\tau, \gamma_\tau)$.

2.1 Multi-Agent Reinforcement Learning

In Section 4, we will base the proposed method off of the well-known *counterfactual multi-agent policy gradients* (COMA) [9] algorithm. We also use the *individual actor-critic* (IAC) [21] algorithm as a sanity check to demonstrate that the multi-agent methods are learning as expected. We describe the algorithms below:

IAC: Although it is a single-agent method, independent actor-critic has demonstrated suprising effectiveness in multi-agent systems [32, 37, 38]. For a single agent a, IAC applies the policy gradient [36] to train an independent actor-critic, resulting in a $\big(\pi^a(u^a|s^a), Q^{\pi^a}(s^a, u^a)\big)$ pair. While able to maximize individual performance, the lack of a coordination mechanism means that IAC will most likely struggle to find optimal solutions in complex multi-agent scenarios.

COMA: COMA uses a joint critic that marginalises out a single agent's action with a counterfactual baseline to address the problem of multi-agent credit assignment. The advantage function used evaluates the contribution of agent a by computing the average of all counterfactuals $\hat{u^a}$ while keeping the actions of other agents fixed.

2.2 Graph Convolutional Networks

A *graph convolution network* (GCN) [20] takes a feature matrix summarizing the attributes of nodes $v_i \in V$ of a graph $G = (E, V)$ as input. In a manner similar to the standard convolutional neural network (CNN), the GCN then extracts locally connected features and outputs a encoded node-level feature matrix by aggregating a node's features with its neighbors'. [20] defines a graph convolution layer as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{1}$$

where $H^{(l)}$ is the feature matrix of the previous layer, l, $\tilde{A} = A + I$ is the adjacency matrix with added self-connections, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $W^{(l)}$ is the trainable weight matrix. $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}$ composes the normalized graph Laplacian, which is theorized to result in a special form of Laplacian smoothing [22] and is possibly a key reason why GCNs work.

2.3 Self-Attention

The self-attention mechanism models the dependencies between different parts of a sequence and has been used successfully in machine reading, abstractive summarization, and machine translation [47, 4, 30, 31, 25, 39]. More recently, it has been adopted for multi-agent reinforcement learning and used for communication [17] and modeling relations between groups of agents [16, 43].

Following the terminology in literature [39], we let $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ be the query, key, and value matrices respectively. Then, the attention weights are generated with respect to the input features F_{in} as follows:

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} = F_{in} \cdot (\mathbf{W_q}, \mathbf{W_k}, \mathbf{W_v}) \tag{2}$$

$$\mathbf{A} = \text{softmax}(\frac{\mathbf{QK}^\top}{\sqrt{d_k}}) \tag{3}$$

where $\sqrt{d_k}$ is the scaling factor and $\mathbf{W_q}, \mathbf{W_k}, \mathbf{W_v}$ are learnable weight matrices.

3 Large Scale Coordination Transfer

A *coordination graph* (CG) [11] is an undirected graph that models inter-agent behaviour by representing the locality of interactions within an environment. Agents are represented as nodes in the graph, and an edge between two agents represents some dependence between their behaviours. Joint actions are then encoded by the graph structure and thus can capture complex interactions between agents. In this paper, we focus on the types of environments coordination graphs have typically been used for, where an agent's action depends only on a subset of other agents. We acknowledge that this may be a limiting condition of our method; however, recent work has shown that CGs can be effective in more general environments [1].

Definition 1 *(Decomposable graph). Let $S \subset V$ be some cut of graph $G = (E, V)$, where $|S| > 2$. Call graph G decomposable if there exists some S such that all $v \in V$ can exist in cuts isomorphic to S.*

Above, we define *decomposable* graphs, a structure we exploit to underline the motivation behind our method. Decomposable coordination graphs can represent scenarios such as robotics swarm control, traffic grid optimization, and city block planning.

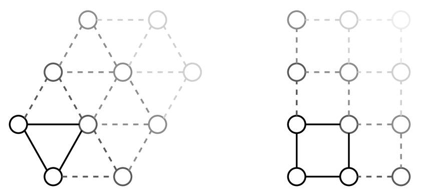


Figure 1: Decomposable graphs with cut S outlined in black. Notice that in each case, the graph is only decomposable if a portion of cut S overlaps with another cut isomorphic to S.

We consider the coordination graphs, $G_s = (V_s, E_s)$ and $G_\tau = (V_\tau, E_\tau)$, for a source and target environment respectively, where $|V_\tau| > |V_s|$. If the two environments model similar dynamics, e.g. are both traffic grid networks, but differ in the number of agents, we expect some portion of the larger coordination graph to be nearly isomorphic to a portion of the smaller one. If the larger graph is then *decomposable*, we can find its corresponding cut S and choose the source environment with a coordination graph most similar to S.

Given a decomposable coordination graph for the target environment, the motivation for transferring parameters between the source and target environments is clear; the optimal policy π_s^* that maximizes the value function in the source environment Q_s also maximizes the contribution of its corresponding group of agents to the global reward. Then, under the condition that agent behaviour in the target environment only depends on neighboring agents (with structure isomorphic to S), the mapping of π_s^* from S to cuts of G_τ will also maximize reward in the target environment.

For target environments in which a static coordination graph is not known, we instead rely on *dynamic coordination graphs* [23], where a learned graph structure is conditioned on current state. Under the assumption that an agent's actions are only influenced by its neighborhood, we expect that a source environment that envelopes this neighborhood size is sufficient to capture knowledge about these localities of interaction. Then, the arguments for transfer learning in the multi-task domain apply as the method falls under the paradigms of *policy distillation* and *representation transfer*.

Parameter sharing between agent-level policies is vital to the intuition behind either case. In areas of a decomposable CG where the cuts isomorphic to S overlap, an agent may otherwise be expected to have two different behaviours. For dynamic coordination graphs, no static mapping to facilitate transfer of policy networks is defined, and thus, it would be unclear which policy network should transfer onto a specific agent in the target environment.

3.1 Critic Structure

We propose a joint critic structure that facilitates coordination between agents and captures locality of agent interactions within an environment. Naturally, due to their ability to aggregate information over a defined graph structure, a graph convolutional network is the clear choice to model coordination graphs.

The default implementation considers the adjacency matrix for the GCN, A, to be a coordination graph with binary edge weights. This is generally a heuristic choice, but given that we only use static adjacency matrix representations for known decomposable coordination graphs, there will be an obvious graph structure to follow. Figure 2 depicts the general structure for this joint critic representation; however, instead of using the self-attention block in the *Learned Adjacency Matrix* module, we pass an adjacency matrix in directly.

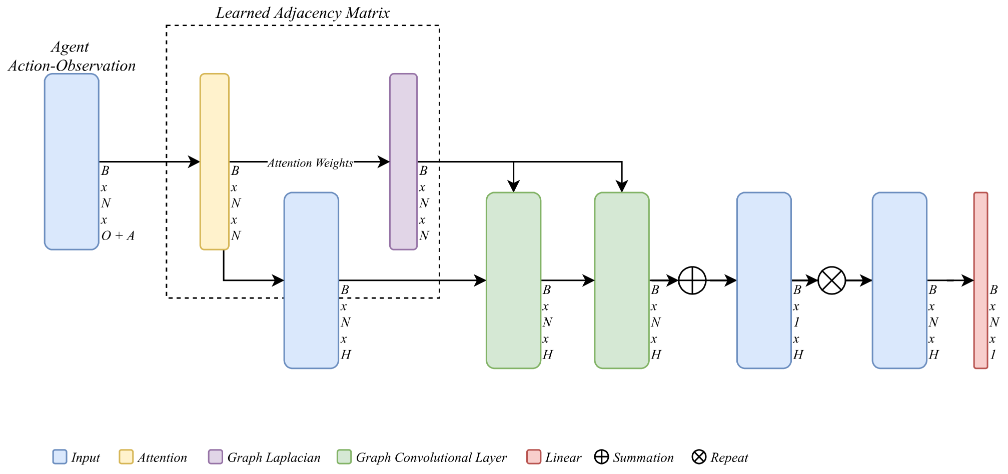


Figure 2: The proposed critic structure. Dimensions listed refer to the output shape of the corresponding layer. N is the number of agents, $O + A$ is the shape of an individual agent's observation-action pair, and H is the hidden size. The summation and repeat depicted act over the *agent* dimension.

Soft Edge Weights: For cases in which the coordination graph is *dynamic*, we must use a learned adjacency matrix representation, such as the *Learned Adjacency Matrix* module in Figure 2. Following the procedure outlined in [23], agent observations are first passed through a self-attention block, and the resultant *attention weights* are then used as a *soft* adjacency matrix. As maintaining differentiability would be difficult if edge weights were strictly binary, this allows the module to be trained end-to-end with the rest of the critic. The self-attention block also outputs its input features F_{in}, used to calculate the query $\mathbf{Q}$ and key $\mathbf{K}$ matrices, which corresponds to the agent action-observations encoded by a linear layer. F_{in} is then used as the input to the first graph convolutional layer.

We note that general joint critic structures receive the *joint* state-action pair as input and thus have variable input shape across differing numbers of agents. The proposed structure instead computes the features for agents in a single parallel batch, reducing computational cost. Since the input shape is constant, critics for environments with varying number of agents have the same number of parameters, allowing for the direct transfer of parameters from the source to the target environment.

The summation that appears towards the end of Figure 2 aggregates observation-action encoding across agents to produce a joint value $Q(\mathbf{s}, \mathbf{a})$ that is then used in training. This type of design has been shown to be effective for multi-agent domains in the form of *value decompostion networks* (VDN) [35] and on set-based data in the form of *deep sets* [48].

While this section has focused largely on the implementation of a single joint critic, the structure can easily be extended to algorithms with numerous joint critics, such as MADDPG [27]. Since agent-level policies must share parameters, it follows that these joint critics must also share parameters.

3.2 Fine-tuning

In the supervised learning domain, *fine-tuning* refers to the initialization of a new model on pre-trained parameters and the subsequent training on a new domain. If the initialized model is used on the new domain without additional training, it is called *zero-shot* learning; however, if the model learns on a limited number of examples from the new domain, it becomes *few-shot* learning.

Our intuition is that there may be some differences in the larger environment which will require agents to adjust their behaviour, as specified in Section 4. However, due to the knowledge distilled by training on the source environment, we are able to show empirically that much fewer training samples are needed to achieve similar results. As a result, we claim that the *fine-tuning* method can be viewed as *few-shot* learning. We note that the base procedure without fine-tuning may be preferable in cases where environment dynamics do not change when more agents are introduced or when the target environment is extremely expensive to sample from. We give the pseudo-code for both proposed methods in Appendix C.

4 Results

We present experiments on three environments: park lights, traffic junction [44], and cooperative navigation [27]. All environments require inter-agent coordination to receive high rewards as evidenced by the performance of the baseline methods. For each case, we choose the source environment to be the smallest non-trivial subset of agents in the target environment where key environment dynamics are captured. For example, cooperative navigation requires $n = 3$ due to the fact that we observe 2 other agents in the observation space, and traffic junction must have 4 agents (2×2) so that each possible configuration for vehicle turns exist. Environment hyper parameters and minor implementation details can be found in Appendix A.

Cooperative navigation follows a particular paradigm possible for multi-agent settings: the *nearest neighbor* observation space, where agents receive the observations of their closest neighbors as well as their own. The related problem of constructing a k-NN graph has been shown to be naively quadratic in the number of agents [3] and even with state-of-the-art optimizations, subquadratic in the worst case [13]. Thus, we claim that zero-shot performance is especially important in this environment; limiting the number of samples needed from the target environment will drastically reduce computational costs. A list of simulation times can be found in Table 1 in Appendix A.

For each environment, we compare the same implementation of COMA on two different critics: our proposed architecture, labeled with **(GCN)**, and a standard multi-layer perceptron. The fine-tuning procedure outlined in Algorithm 2 is labed with **(Fine Tune)**. We establish **IAC** and **IAC (Fine Tune)** as the baseline methods for comparison, given that there is no central coordination scheme for these methods. **IAC (Fine Tune)** initializes its parameters with a model pre-trained on the source environment and is fine-tuned on the target environment, like **COMA (Fine Tune)**; this is possible since its independent critics keep a constant input shape with respect to the number of agents. To ensure fair comparison, we keep the number of learnable parameters across all algorithms the same. Further information can be found in Appendix B.

4.1 Park Lights

We create the *park lights* environment as a model of outdoor public lighting and as a way to isolate the benefits of our transfer learning approach. Here, agents are situated on a $N \times M$ grid and model individual park lights. At each timestep, an agent receives information about its 8 adjacent neighbors, containing only knowledge about whether that agent's light is on or off, the state of its own light, and the number of timesteps its light has been in that state. Agents must then choose to either toggle their light or do nothing, with each illuminated light also giving light to its 8 adjacent cells. The global reward is then the function $T - 0.1 \sum_{i=0}^{N \times M} t_i$, where T is the total number of cells lit, either by its corresponding agent or by the 8 agents adjacent to it, and t_i is the number of timesteps agent i's light has been on.

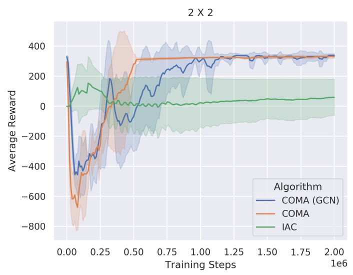
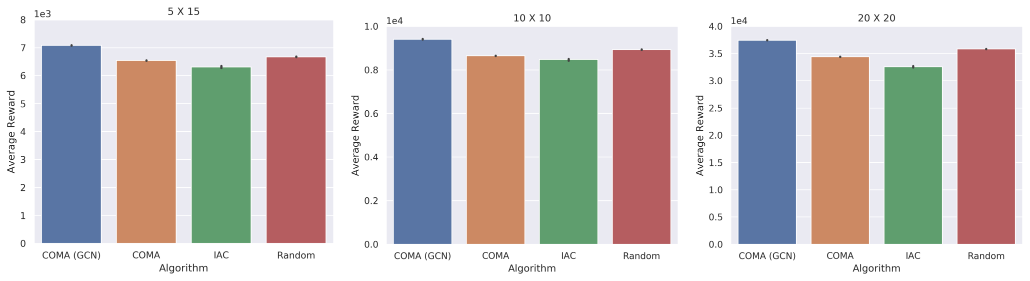


Figure 3: Training reward on the source environment. Refer to Figure 4 for the rest of the experiments.



(a) Zero-shot (b) Zero-shot (c) Zero-shot

Figure 4: Park Lights Environment (3 target environments). GCN uses an adjacency matrix with edges between agents in adjacent cells.

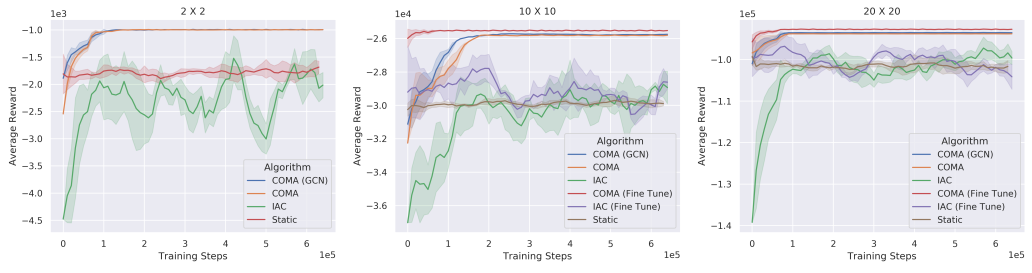
Figure 3 shows the average performance of the algorithms on episodes of length 100 during training. Notice that the optimal policy requires that two adjacent agents not have their lights on at the same time and that agents alternate illuminating adjacent squares. As a result of this essential coordination, **IAC** is unable to converge; each individual agent can only observe whether its neighbors has their lights on or off at a given timestep, so it is unable to predict whether it or its adjacent agents should toggle their light.

We see that the zero-shot performance (Figure 4) of the policy trained on the proposed GCN architecture is the best. This follows logically from the fact that *park lights* has a decomposable coordination graph since increasing the size of the grid does not change environment dynamics. Furthermore, even though the return of **COMA** is comparable to that of **COMA (GCN)** in the source environment, the performance gap widens in each of the target environments, performing about as well as **IAC**. This indicates that a simple MLP critic is not able to capture and distill knowledge about agent-level interactions and that the GCN model of the coordination graph is what facilitates knowledge transfer.

4.2 Traffic Junction

The *traffic junction* environment is a popular benchmark in multi-agent reinforcement learning where cars are randomly added to a traffic grid network and must avoid collisions as they travel to their pre-determined destinations. At every intersection of the $N \times M$ road network, an agent models the traffic light and is able to observe information within a limited range of one grid from itself. To prevent flickering of lights, agents are able to toggle the color of the traffic light every 5 timesteps.

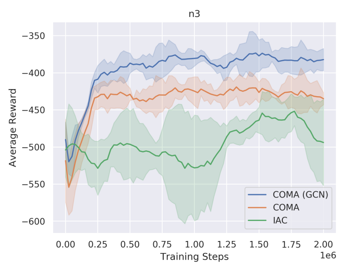
Even though the traffic junction environment is commonly used in literature, there is no universal reward function that all works agree upon. [50] suggests that a reward based on the *queue length* at traffic lights is optimal for learning, so in this work, we follow the implementation given by [49], where the reward function is a combination of *queue length*, *wait time*, and *vehicle delay*. However, to better indicate transfer performance, we switch to episodic rollouts of length 100 rather than follow an infinite horizon approach. Furthermore, we introduce the **Static** baseline, which switches the color of the traffic lights every 5 timesteps (as soon as they can be switched). This represents a common real-world implementation of traffic lights, and thus is included to better compare results.

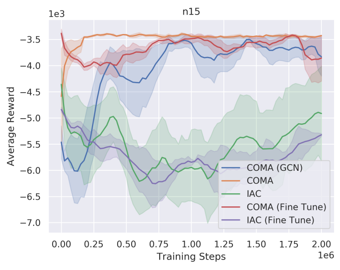


(a) Training on Source Environment (b) Fine-tuning (c) Fine-tuning

Figure 5: Traffic Junction Environment (1 source and 2 target environments). GCN uses an adjacency matrix with edges corresponding to roads between agents. All methods other than **COMA (Fine Tune)** and **IAC (Fine Tune)** are trained from scratch on the target environments.

In contrast to *park lights*, the actions of a *traffic light* agent will eventually influence the actions of agents beyond those immediately adjacent to it given that vehicles take many timesteps to travel down the road. While the coordination graph is still structured the same as the road network itself (as an agent does not interact with agents other than its neighbors *in the same timestep*), we expect that increasing the grid size has some non-trivial impact on optimal behaviors. Figure 5 demonstrates that our approach is reasonable in this environment. We observe that while the zero-shot performance of the proposed critic is high, fine-tuning it with a few samples of the target environment is able to improve its performance. As a result, our method converges much quicker than the others and generally has the best performance.


(a) Training on Source Environment


(b) Fine-tuning

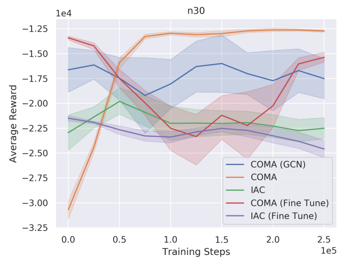

(c) Fine-tuning

Figure 6: Cooperative Navigation Environment (1 source and 2 target environments). GCN uses the learned adjacency matrix depicted in Figure 2. All methods other than **COMA (Fine Tune)** and **IAC (Fine Tune)** are trained from scratch on the target environments.

4.3 Cooperative Navigation

Cooperative navigation is a commonly used scenario for the popular multi-agent particle environment (MPE) [27]. Here, N agents are tasked with cooperatively covering N landmarks across the map and are able to observe information about the k closest landmarks and agents. We follow the implementation given by the official codebase for [26].

As demonstrated by Table 1, the k-nearest neighbors observation space causes superlinear growth in runtime with respect to the number of agents in the environment. Thus, we provide fine-tuned methods only for environments up to $n = 30$; we emphasize that zero-shot performance becomes much more critical as the number of agents grows beyond that amount. Since there is no clear coordination graph encoded into the structure of the environment, we use the *Learned Adjacency Matrix* module illustrated in Figure 2 to learn a dynamic coordination graph. To maintain fair comparison, we also include the parameters introduced here into the total count.

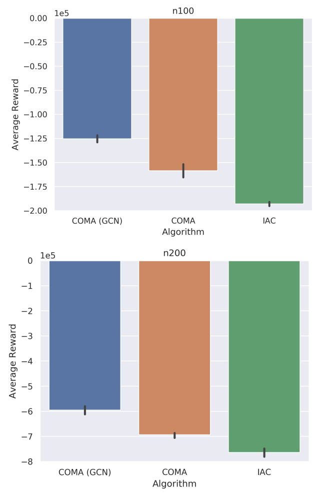


Figure 7: Cooperative Navigation Environment (2 target environments). Zero-shot performance

We observe in Figure 6 that while the GCN critic with the *Learned Adjacency Matrix* module does converge, it is less consistently monotonic when compared to the MLP critic, particularly when fine-tuning is used. Most importantly though, the zero-shot performance of the proposed critic is still high (and the highest in Figure 7), indicating that transfer is effective even in environments where a coordination graph is completely unknown. This can be explained by the fact that the dynamic coordination graph model may not adjust well to environments with a different amount of agents. The architecture learns a representation of dynamics in the smaller source environment, which corresponds to strong performance on the target environment; however, we hypothesize that suddenly introducing many more agents destabilizes the self-attention module, as the nearest agents observed by an individual varies more in the larger environments (the $n = 3$ source environment always observes the same agents). The module then needs to adjust its learned queries to account for this shift in input domain.

5 Conclusion and Future Work

In this work, we propose a joint critic architecture that is able to be transferred directly between environments with differing numbers of agents. While we demonstrate in Section 4 that it performs better than baseline critics and is able to effectively facilitate knowledge distillation, we acknowledge that its reliance on parameter sharing and a simple GCN architecture can be limiting [7]. In future works, we would like to investigate the possibilities of using more of the various critic structures found in literature, such as hypernetworks [32].

We further propose a transfer learning procedure to share knowledge between a single source environment and multiple target environments. This is effective for the environments that we experiment in; however, there may be cases where utilizing knowledge from multiple source environments can improve performance, such as in the single agent setting [33, 46]. We leave the choice of these source environments – selecting a cut S from the coordination graph of the target environment – to future work.

References

[1] Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *International Conference on Machine Learning*, pages 980–991. PMLR.

[2] Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015). Policy transfer using reward shaping. In *AAMAS*, pages 181–188.

[3] Chen, J., ren Fang, H., and Saad, Y. (2009). Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, **10**(69), 1989–2012.

[4] Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas. Association for Computational Linguistics.

[5] Da Silva, F. L. and Costa, A. H. R. (2017). Towards zero-shot autonomous inter-task mapping through object-oriented task description. In *Workshop on Transfer in Reinforcement Learning (TiRL)*.

[6] Da Silva, F. L. and Costa, A. H. R. (2019). A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, **64**, 645–703.

[7] Dehmamy, N., Barabási, A.-L., and Yu, R. (2019). Understanding the representation power of graph neural networks in learning graph topology. *arXiv preprint arXiv:1907.05008*.

[8] Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 165–172.

[9] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[10] Gronauer, S. and Dieopold, K. (2021). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, pages 1–49.

[11] Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored mdps. In *NIPS*, volume 1, pages 1523–1530.

[12] Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017). Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer.

[13] Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., and Zhang, H. (2011). Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, page 1312–1317. AAAI Press.

[14] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., *et al.* (2018). Deep q-learning from demonstrations. In *Thirty-second AAAI conference on artificial intelligence*.

[15] Hu, Y., Gao, Y., and An, B. (2014). Multiagent reinforcement learning with unshared value functions. *IEEE transactions on cybernetics*, **45**(4), 647–662.

[16] Iqbal, S. and Sha, F. (2019). Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR.

[17] Jiang, J. and Lu, Z. (2018). Learning attentional communication for multi-agent cooperation. *arXiv preprint arXiv:1805.07733*.

[18] Khan, A., Tolstaya, E., Ribeiro, A., and Kumar, V. (2020). Graph policy gradients for large scale robot control. In *Conference on robot learning*, pages 823–834. PMLR.

[19] Kim, B., Farahmand, A.-m., Pineau, J., and Precup, D. (2013). Learning from limited demonstrations. In *NIPS*, pages 2859–2867. Citeseer.

[20] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[21] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

[22] Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*.

[23] Li, S., Gupta, J. K., Morales, P., Allen, R., and Kochenderfer, M. J. (2020). Deep implicit coordination graphs for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.11438*.

[24] Li, Y., Wang, L., Yang, J., Wang, E., Wang, Z., Zhao, T., and Zha, H. (2021). Permutation invariant policy optimization for mean-field multi-agent reinforcement learning: A principled approach.

[25] Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

[26] Liu, I.-J., Yeh, R. A., and Schwing, A. G. (2020). Pic: permutation invariant critic for multi-agent deep reinforcement learning. In *Conference on Robot Learning*, pages 590–602. PMLR.

[27] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

[28] Oliehoek, F. A. and Amato, C. (2016). *A concise introduction to decentralized POMDPs*. Springer.

[29] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**(10), 1345–1359.

[30] Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

[31] Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

[32] Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR.

[33] Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

[34] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[35] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., *et al.* (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

[36] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[37] Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, **12**(4), e0172395.

[38] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.

[39] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[40] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*.

[41] Wang, L., Yang, Z., and Wang, Z. (2020a). Breaking the curse of many agents: Provable mean embedding q-iteration for mean-field reinforcement learning. In H. Daume and A. Singh, editors, *37th International Conference on Machine Learning, ICML 2020*, 37th International Conference on Machine Learning, ICML 2020, pages 10034–10045. International Machine Learning Society (IMLS). Publisher Copyright: © 2020 by the Authors.; 37th International Conference on Machine Learning, ICML 2020 ; Conference date: 13-07-2020 Through 18-07-2020.

[42] Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020b). Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, **53**(3), 1–34.

[43] Wright, M. A. and Horowitz, R. (2019). Attentional policies for cross-context multi-agent reinforcement learning. *arXiv preprint arXiv:1905.13428*.

[44] Wu, C., Kreidieh, A., Parvate, K., Vinitsky, E., and Bayen, A. M. (2017). Flow: Architecture and benchmarking for reinforcement learning in traffic control. *arXiv preprint arXiv:1710.05465*, **10**.

[45] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. (2018). Mean field multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5571–5580. PMLR.

[46] Yin, H. and Pan, S. J. (2017). Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *Thirty-First AAAI conference on artificial intelligence*.

[47] Yu, A. W., Dohan, D., Luong, M.-T., Zhao, R., Chen, K., Norouzi, M., and Le, Q. V. (2018). Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*.

[48] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2017). Deep sets. *arXiv preprint arXiv:1703.06114*.

[49] Zhang, Z., Yang, J., and Zha, H. (2019). Integrating independent and centralized multi-agent reinforcement learning for traffic signal network optimization. *arXiv preprint arXiv:1909.10651*.

[50] Zheng, G., Zang, X., Xu, N., Wei, H., Yu, Z., Gayah, V., Xu, K., and Li, Z. (2019). Diagnosing reinforcement learning for traffic signal control. *arXiv preprint arXiv:1905.04716*.

[51] Zhou, L., Yang, P., Chen, C., and Gao, Y. (2016). Multiagent reinforcement learning with sparse interactions by negotiation and knowledge transfer. *IEEE transactions on cybernetics*, **47**(5), 1238–1250.

Supplementary Material for Large Scale Coordination Transfer for Copperative Multi-Agent Reinforcement Learning

A Experiment Details

All methods for each environment are trained over 4 separate seeds and evaluated by averaging 4 evaluation episodes.

Table 1: Average environment simulation times (100 step)

Environment	Number of Agents	Training Time
Park Lights	4	4.92e-03 sec
	75	0.0812 sec
	100	0.106 sec
	400	0.425 sec
Traffic Junction	4	3.37 sec
	100	19.5 sec
	200	67.0 sec
Cooperative Navigation	3	0.0828 sec
	15	3.57 sec
	30	25.2 sec
	100	113 sec
	200	7312 sec

Table 1 depicts the simulation time of a hundred steps in the corresponding environment. While these type of explicit computational costs are not often shown in literature due to their dependence on factors such as hardware, cluster usage, and implementation, we include this table to illustrate the importance of maximizing knowledge transferred from the source environment. All times refer to the same implementation on the same hardware (Intel Xeon Silver 4116 CPU and NVIDIA GeForce RTX 2080 Ti), and overall cluster usage is kept constant to the best of our ability.

A.1 Park Lights

Agents are located within cells of an $N \times M$ grid and are given two possible actions corresponding with the state of their light: [ON, OFF]. ON lights are able to illuminate all cells within a 3×3 enclosure around them. At each timestep, an agent observes: 1) the state (ON or OFF) of the lights corresponding to agents in the 8 adjacent grids, 2) the state of their own light, and 3) the number of timesteps their light has been in the ON state. This corresponds with an observation shape of 18. All transitions are deterministic, and lights have randomly initialized state. We compare algorithms with approximately 5k policy parameters and 80k critic parameters, where policy and critic networks are share parameters across all agents.

A.2 Traffic Junction

We follow the traffic lights implementation from the official FLOW repository[1]. We follow the reward function given by the authors of [49], which corresponds to a linear combination of *queue length*, *wait time*, *vehicle delay*, *emergency stops*, *traffic light phase changes*, and the number of vehicles *passing the traffic light*. Traffic light agents are given two actions, [NO-OP, CHANGE], and are allowed to change their lights every 5 timesteps. Observation spaces and SUMO parameters likewise follow [49]. We allow agents to observe a finite distance around them, gathering information about vehicle velocity, distance to intersection, average waiting time, and queue length as well as run heterogenous

[1]https://github.com/flow-project/flow

autonomous vehicles over the road network. Here, we compare algorithms with approximately 5k policy parameters and 90k critic parameters.

A.3 Cooperative Navigation

For all cooperative navigation scenarios, we follow implementation given by [26]. However, we fix the number of observed agents in the observation space to 2 and set the number of landmarks to N, the total number of agents in the environment. The environment then uses a discrete action space, [NO-OP, LEFT, RIGHT, UP, DOWN], with each direction corresponding to noisy force on the agent resulting in movement. Finally, we compare algorithms with approximately 6k policy parameters and 250k critic parameters.

B Baseline Implementation Details

Policy networks between all methods across all agents share the same structure. Table 2 displays the corresponding layer types and shapes. We remark that this corresponds with a basic MLP policy that outputs the probability of each action in the action space.

Table 2: Policy Network Structure

Layer	Size
Linear + ReLU	64
Linear + ReLU	64
Linear + Softmax	Action Shape

Critic networks differ between MLP joint critics, MLP individual critics, and GCN critics. As is expected, **COMA** uses the MLP joint critic, **COMA (GCN)** uses the GCN critic, and **IAC** uses the MLP individual critic. All critics take observation-action pairs as input, and we scale layer sizes depending on the environment to match the number of parameters for each algorithm. Architectures are listed below:

Table 3: Joint/Individual MLP Critic Structure

Layer	Size
Linear + ReLU	Varying (Usually 256 or 512)
Linear + ReLU	Varying (Usually 256 or 512)
Linear	1

Table 4: GCN Critic Structure

Layer	Size
Linear (Attention Encoding)	Varying (Usually 256 or 512)
Linear (Attention Queries)	Varying (Usually 256 or 512)
Linear (Attention Values)	Varying (Usually 256 or 512)
Linear + ReLU	Varying (Usually 256 or 512)
Matrix Multiplication + Linear + ReLU	Varying (Usually 256 or 512)
Sum + Linear	1

COMA and IAC share the same general parameters. We update each step of the environment on a replay buffer of size one and do not use batching across environments. Table 5 provides all relevant hyperparameters for the algorithms.

Table 5: Algorithm Hyperparameters

Parameters	Value
Discount γ	0.95
Exploration Rate	0.1
Exploration Anneal	0.998
Policy learning rate	0.0001
Critic learning rate	0.001
Optimizer	Adam

C Algorithms

Algorithm 1 Multiagent Large Scale Coordination Transfer

```
Require: $T_s$ iterations on $E_s$, a policy gradient multi-agent reinforcement learning method $RL$ that uses our proposed critic structure
Initialize: Policy parameters $\theta_s$ and critic parameters $\phi_s$.
while $t < T_s$ do:
    Train agents on the source environment: $\theta_s, \phi_s = RL(\theta_s, \phi_s, E_s)$
end while
Evaluate: Measure performance of $\pi^{\theta_\tau}$ on $E_\tau$.
```

Algorithm 2 Multiagent Large Scale Coordination Transfer with Fine-Tuning

```
Require: $T_s$ iterations on $E_s$, $T_\tau$ iterations on $E_\tau$, a policy gradient multi-agent reinforcement learning method $RL$ that uses our proposed critic structure
Initialize: Policy parameters $\theta_s$ and critic parameters $\phi_s$.
while $t < T_s$ do:
    Train agents on the source environment: $\theta_s, \phi_s = RL(\theta_s, \phi_s, E_s)$
end while
Transfer: Initalize policy parameters $\theta_\tau = \theta_s$ and critic parameters $\phi_\tau = \phi_s$
while $t < T_\tau$ do:
    Train agents on the target environment: $\theta_\tau, \phi_\tau = RL(\theta_\tau, \phi_\tau, E_\tau)$
end while
Evaluate: Measure performance of $\pi^{\theta_\tau}$ on $E_\tau$.
```