Deep Coordination Graphs

Wendelin Böhmer [1] **Vitaly Kurin** [1] **Shimon Whiteson** [1]

Abstract

This paper introduces the *deep coordination graph* (DCG) for collaborative multi-agent reinforcement learning. DCG strikes a flexible trade-off between representational capacity and generalization by factoring the joint value function of all agents according to a coordination graph into payoffs between pairs of agents. The value can be maximized by local message passing along the graph, which allows training of the value function end-to-end with Q-learning. Payoff functions are approximated with deep neural networks that employ parameter sharing and low-rank approximations to significantly improve sample efficiency. We show that DCG can solve predator-prey tasks that highlight the *relative overgeneralization* pathology, as well as challenging StarCraft II micromanagement tasks.

1. Introduction

One of the central challenges in cooperative *multi-agent reinforcement learning* (MARL, Oroojlooy jadid & Hajinezhad, 2019) is coping with the size of the joint action space, which grows exponentially in the number of agents. For example, just eight agents, each with six actions, yield a joint action space with more than a million actions. Efficient MARL methods must thus generalize over large joint action spaces, in the same way that convolutional neural networks allow deep RL to generalize over large visual state spaces.

MARL often addresses the issue of large joint observation and action spaces by assuming that the learned control policy is fully *decentralized*, that is, each agent acts independently based on its own observations only. For example, Figure 1a shows how the joint value function can be factored into *utility functions* that each depend only on the actions of one agent (Sunehag et al., 2018; Rashid et al., 2018). Consequently, the joint value function can be efficiently maximized if each agent simply selects the action that maximizes its corresponding utility function. This factorization can represent any deterministic (and thus at least one optimal) joint policy. However, that policy may not be learnable due to a game-theoretic pathology called *relative overgeneralization*[1] (Panait et al., 2006): during exploration other agents act randomly and punishment caused by uncooperative agents may outweigh rewards that would be achievable with coordinated actions. If the employed value function does not have the representational capacity to distinguish the values of coordinated and uncoordinated actions, an optimal policy cannot be learned.

A higher-order value factorization can be expressed as an undirected *coordination graph* (CG, Guestrin et al., 2002a), where each vertex represents one agent and each (hyper-) edge one *payoff function* over the joint action space of the connected agents. Figure 1b shows a CG with pairwise edges and the corresponding value factorization. Here the value depends non-trivially on the actions of *all* agents, yielding a richer representation. Although the value can no longer be maximized by each agent individually, the greedy action can be found using message passing along the edges (also known as *belief propagation*, Pearl, 1988). *Sparse cooperative Q-learning* (Kok & Vlassis, 2006) applies CGs to MARL, but does not scale to real-world tasks, as each payoff function (f^{12} and f^{23} in Figure 1b) is represented as a table over the state and joint action space of the connected agents. Castellini et al. (2019) use neural networks to approximate payoff functions in the simplified case of non-sequential one-shot games. Moreover, neither approach shares parameters between the approximated payoff functions, so agents in each factor, represented by an edge, must experience all corresponding action combinations. This can require visiting a large subset of the joint action space.

While decentralization can be a requirement of the task at hand, for example when communication between agents is impossible, many important applications that allow for centralized or distributed controllers face the same issues. Examples are power, water or heat grids (Correa-Posada & Sánchez-Martin, 2015), electronic trading (Bacoyannis et al., 2018), computer games (Vinyals et al., 2019), auto-

[1]Department of Computer Science, Oxford University, United Kingdom. Correspondence to: Wendelin Böhmer <wendelinboehmer@gmail.com>.

Accepted at the 37^{th} International Conference on Machine Learning (2020).

[1] Not to be confused with the general term *generalization* in the context of function approximation mentioned earlier.

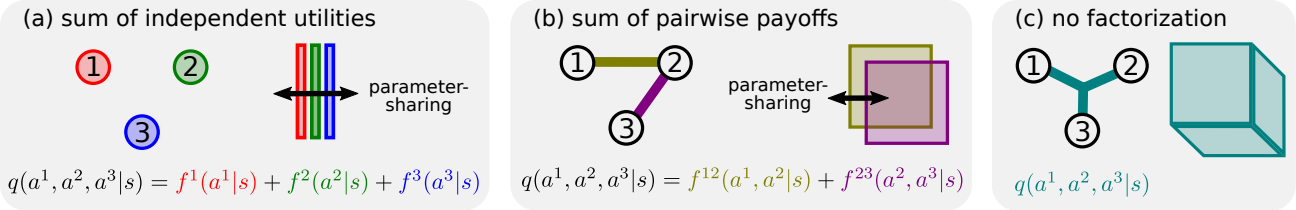

Figure 1. Examples of value factorization for 3 agents: (a) sum of independent utilities (as in VDN, Sunehag et al., 2018) corresponds to an unconnected CG. QMIX uses a monotonic mixture of utilities instead of a sum (Rashid et al., 2018); (b) sum of pairwise payoffs (Castellini et al., 2019), which correspond to pairwise edges; (c) no factorization (as in QTRAN, Son et al., 2019) corresponds to one hyper-edge connecting all agents. Factorization allows parameter sharing between factors, shown next to the CG, which can dramatically improve the algorithm's sample complexity.

matic factories (Dotoli et al., 2017), drone swarms (Alonso-Mora et al., 2017a), driver simulations (Behbahani et al., 2019), and routing of taxi fleets (Alonso-Mora et al., 2017b).

To address relative overgeneralization for centralized or distributed controllers, we propose the *deep coordination graph* (DCG), a deep RL algorithm that scales CG for the first time to the large state and action spaces of modern benchmark tasks. DCG represents the value function as a CG with pairwise payoffs[2] (Figure 1b) and individual utilities (Figure 1a). This improves the representational capacity beyond state-of-the-art value factorization approaches like VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018). To achieve scalability, DCG employs parameter sharing between payoffs and utilities. Parameter sharing between agents has long been a staple of decentralized MARL. Methods like VDN and QMIX condition an agent's utility on its history, that is, its past observations and actions, and share the parameters of all utility functions. Experiences of one agent are thus used to train all. This can dramatically improve the sample efficiency compared to methods with unfactored values (Foerster et al., 2016; 2018; Lowe et al., 2017; Schröder de Witt et al., 2019; Son et al., 2019), which correspond to a CG with one hyper-edge connecting all agents (Figure 1c). DCG takes parameter sharing one step further by approximating *all* payoff functions with the same neural network. To allow unique outputs for each payoff, the network is conditioned on a learned embedding of the participating agents' histories. This requires only one linear layer more than VDN and fewer parameters than QMIX.

DCG is trained end-to-end with deep Q-learning (DQN, Mnih et al., 2015), but uses message passing to coordinate greedy action selection between *all* agents in the graph. For k message passes over n agents with m actions each, the time complexity of maximization is only $\mathcal{O}(km(n+m)|\mathcal{E}|)$, where $|\mathcal{E}| \leq \frac{n^2-n}{2}$ is the number of (pairwise) edges, compared to $\mathcal{O}(m^n)$ for DQN without factorization.

[2] The method can be easily generalized to CGs with hyper-edges, i.e., payoff functions for more than 2 agents.

We compare DCG's performance with that of other MARL Q-learning algorithms in a challenging family of predator-prey tasks that require coordinated actions and hard StarCraft II micromanagement tasks. In the former, DCG is the only algorithm that solves the harder tasks and in the latter DCG outperforms state-of-the-art QMIX in some levels. An open-source implementation of DCG and all discussed algorithms and tasks is available for full reproducibility[3].

2. Background

In this paper we assume a Dec-POMDP for n agents $\langle \mathcal{S}, \{\mathcal{A}^i\}_{i=1}^n, P, r, \{\mathcal{O}^i\}_{i=1}^n, \{\sigma^i\}_{i=1}^n, n, \gamma \rangle$ (Oliehoek & Amato, 2016). $\mathcal{S}$ denotes a discrete or continuous set of environmental states and $\mathcal{A}^i$ the discrete set of actions available to agent i. At discrete time t, the next state $s_{t+1} \in \mathcal{S}$ is drawn from transition kernel $s_{t+1} \sim P(\cdot|s_t, \boldsymbol{a}_t)$, conditioned on the current state $s_t \in \mathcal{S}$ and joint action $\boldsymbol{a}_t \in \mathcal{A} := \mathcal{A}^1 \times \ldots \times \mathcal{A}^n$ of all agents. A transition yields collaborative reward $r_t := r(s_t, \boldsymbol{a}_t)$, and $\gamma \in [0, 1)$ denotes the discount factor. Each agent i observes the state only partially by drawing observations $o_t^i \in \mathcal{O}^i$ from its observation kernel $o_t^i \sim \sigma^i(\cdot|s_t)$. The history of agent i's observations $o_t^i \in \mathcal{O}^i$ and actions $a_t^i \in \mathcal{A}^i$ is in the following denoted as $\tau_t^i := (o_0^i, a_0^i, o_1^i, \ldots, o_{t-1}^i, a_{t-1}^i, o_t^i) \in (\mathcal{O}^i \times \mathcal{A}^i)^t \times \mathcal{O}^i$. Without loss of generality, this paper considers episodic tasks, which yield episodes $(s_0, \{o_0^i\}_{i=1}^n, \boldsymbol{a}_0, r_0, \ldots, s_T, \{o_T^i\}_{i=1}^n)$ of varying (but finite) length T.

2.1. Deep Q-learning

The goal of collaborative multi-agent reinforcement learning (MARL) is to find an optimal policy $\pi^* : \mathcal{S} \times \mathcal{A} \to [0, 1]$, that chooses joint actions $\boldsymbol{a}_t \in \mathcal{A}$ such that the expected discounted sum of future rewards is maximized. This can be achieved by estimating the optimal Q-value function[4]:

[3] https://github.com/wendelinboehmer/dcg

[4] We overload the notation $f(y|x)$ to also indicate the inputs x and multivariate outputs y of multivariate functions f.

$$q^*(\boldsymbol{a}|s) := \mathbb{E}_{\pi^*}\Big[\sum_{t=0}^{T-1}\gamma^t r_t \,\Big|\, \substack{s_0=s \\ \boldsymbol{a}_0=\boldsymbol{a}}\Big] \quad (1)$$
$$= r(s,\boldsymbol{a}) + \gamma \int P(s'|s,\boldsymbol{a}) \max q^*(\cdot|s')\, ds' .$$

The optimal policy $\pi^*(\cdot|s_t)$ chooses greedily the action $\boldsymbol{a} \in \mathcal{A}$ that maximizes the corresponding optimal Q-value $q^*(\boldsymbol{a}|s_t)$. In fully observable discrete state and action spaces, q^* can be learned in the limit from interactions with the environment (Watkins & Dayan, 1992). For large or continuous state spaces, q^* can only be approximated, e.g., with a deep neural network q_θ (DQN, Mnih et al., 2015), parameterized by θ, by minimizing the mean-squared Bellman error with gradient descent:

$$\mathcal{L}_{\text{DQN}} := \mathbb{E}\Big[\frac{1}{T}\sum_{t=0}^{T-1}\Big(r_t + \gamma \max q_{\bar{\theta}}(\cdot|s_{t+1}) - q_\theta(\boldsymbol{a}_t|s_t)\Big)^2\Big] .$$

The expectation is estimated with episodes from an experience replay buffer holding previously observed episodes (Lin, 1992), and $\bar{\theta}$ denotes the parameter of a separate target network, which is periodically replaced with a copy of θ to improve stability. Double Q-learning further stabilizes training by choosing the next action greedily w.r.t. the current network q_θ, i.e., $q_{\bar{\theta}}(\arg\max q_\theta(\cdot|s_{t+1})|s_{t+1})$ instead of the target network $\max q_{\bar{\theta}}(\cdot|s_{t+1})$ (van Hasselt et al., 2016).

In partially observable environments, the learned policy cannot condition on the state s_t. Instead, Hausknecht & Stone (2015) approximate a Q-function that conditions on the agent's history $\boldsymbol{\tau}_t := \{\tau_t^i\}_{i=1}^n$, i.e., $q_\theta(\boldsymbol{a}|\boldsymbol{\tau}_t)$, by conditioning a recurrent neural network (e.g., a GRU, Chung et al., 2014) on the agents' observations $\boldsymbol{o}_t := (o_t^1, \dots, o_t^n)$ and last actions $\boldsymbol{a}_{t-1}$, that is, $q_\theta(\boldsymbol{a}|\boldsymbol{h}_t)$ conditions on the recurrent network's hidden state $h_\psi(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{o}_t, \boldsymbol{a}_{t-1})$, $\boldsymbol{h}_0 = \boldsymbol{0}$.

Applying DQN to multi-agent tasks quickly becomes infeasible, due to the combinatorial growth of state and action spaces. Moreover, DQN values cannot be maximized without evaluating all actions. To allow efficient maximization for MARL Q-learning, various algorithms based on value factorization have been developed. We derive IQL (Tan, 1993), VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2018) and QTRAN (Son et al., 2019) in Appendix A.1.

Sunehag et al. (2018) define the VDN value function $q^{\text{VDN}}(s_t, \boldsymbol{a}) := \sum_{i=1}^n f^i(a^i|s_t)$ and introduce parameter sharing between the agents' utility functions $f^i(a^i|s_t) \approx f_\theta^v(a^i|\tau_t^i)$ to dramatically improve the sample efficiency of VDN. The utility function f_θ^v has a fixed number of outputs $A := |\cup_{i=1}^n \mathcal{A}^i|$, but agent i can restrict maximization to $\mathcal{A}^i$ by setting the utilities of unavailable actions to $-\infty$. Specialized behavior between agents can be represented by conditioning f_θ^v on the agent's role, or more generally on the agent's ID (Foerster et al., 2018; Rashid et al., 2018).

2.2. Coordination graphs

An undirected *coordination graph* (CG, Guestrin et al., 2002a) $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ contains a vertex $v_i \in \mathcal{V}$ for each agent $1 \le i \le n$ and a set of undirected edges $\{i, j\} \in \mathcal{E}$ between vertices v_i and v_j. The graph is usually specified before training, but Guestrin et al. (2002b) suggest that the graph could also depend on the state, that is, each state can have its own unique CG. A CG induces a factorization[5] of the Q-function into *utility functions* f^i and *payoff functions* f^{ij} (Fig. 1a and 1b):

$$q^{\text{CG}}(s_t, \boldsymbol{a}) := \frac{1}{|\mathcal{V}|}\sum_{v^i \in \mathcal{V}} f^i(a^i|s_t) + \frac{1}{|\mathcal{E}|}\sum_{\{i,j\}\in\mathcal{E}} f^{ij}(a^i, a^j|s_t) . \quad (2)$$

The special case $\mathcal{E} = \varnothing$ yields VDN, but each additional edge enables the value representation of the joint actions of a pair of agents and can thus help to avoid relative overgeneralization. Prior work also considers higher order coordination where the payoff functions depend on the actions of larger sets of agents (Guestrin et al., 2002a; Kok & Vlassis, 2006; Castellini et al., 2019), corresponding to graphs with hyper-edges (Figure 1c). For the sake of simplicity we restrict ourselves here to pairwise edges, which yield at most $|\mathcal{E}| \le \frac{1}{2}(n^2 - n)$ edges, in comparison to up to $\frac{n!}{d!\,(n-d)!}$ hyper-edges of degree d. The induced Q-function q^{CG} can be maximized locally using *max-plus*, also known as *belief propagation* (Pearl, 1988). At time t each node i sends messages $\mu_t^{ij}(a^j) \in \mathbb{R}$ over all adjacent edges $\{i, j\} \in \mathcal{E}$. In a tree topology, this message contains the maximized contributions of the sender's sub-tree given that the receiver chooses $a^j \in \mathcal{A}^j$. Messages can be computed locally as:

$$\mu_t^{ij}(a^j) \leftarrow \max_{a^i}\Big\{\frac{1}{|\mathcal{V}|}f^i(a^i|s_t) + \frac{1}{|\mathcal{E}|}f^{ij}(a^i, a^j|s_t) + \sum_{\{k,i\}\in\mathcal{E}} \mu_t^{ki}(a^i) - \mu_t^{ji}(a^i)\Big\} . \quad (3)$$

This process repeats for a number of iterations, after which each agent i can locally find the action a_*^i that maximizes the estimated joint Q-value $q^{\text{CG}}(s_t, \boldsymbol{a}_*)$:

$$a_*^i := \arg\max_{a^i}\Big\{\frac{1}{|\mathcal{V}|}f^i(a^i|s_t) + \sum_{\{k,i\}\in\mathcal{E}} \mu_t^{ki}(a^i)\Big\} . \quad (4)$$

Convergence of messages is only guaranteed for acyclic CGs (Pearl, 1988; Wainwright et al., 2004). However, subtracting a normalization constant $c_{ij} := \sum_a \mu_t^{ij}(a) \,/\, |\mathcal{A}^i|$ from each message μ^{ij} before it is sent often leads to convergence in cyclic graphs as well (Murphy et al., 1999; Crick & Pfeffer, 2002; Yedidia et al., 2003). See Algorithm 3 in the appendix.

[5] The normalizations $\frac{1}{|\mathcal{V}|}$ and $\frac{1}{|\mathcal{E}|}$ are not strictly necessary but allow the potential transfer of learned DCG to other topologies.

3. Method

We now introduce the *deep coordination graph* (DCG), which learns the utility and payoff functions of a coordination graph $\langle \mathcal{V}, \mathcal{E} \rangle$ with deep neural networks. In their state-free implementation, Castellini et al. (2019) learn a separate network for each function f^i and f^{ij}. However, properly approximating these Q-values requires observing the joint actions of each agent pair in the edge set $\mathcal{E}$, which can be a significant subset of the joint action space of all agents $\mathcal{A}$. We address this issue by focusing on an architecture that shares parameters across functions and restricts them to locally available information, i.e., to the histories of the participating agents. DCG takes inspiration from highly scalable methods (Yang et al., 2018; Chen et al., 2018) and improves upon Kok & Vlassis (2006) and Castellini et al. (2019) by incorporating the following design principles:

i. Restricting the payoffs $f^{ij}(a^i, a^j | \tau_t^i, \tau_t^j)$ to local information of agents i and j only;
ii. Sharing parameters between all payoff and utility functions through a common recurrent neural network;
iii. Low-rank approximation of joint-action payoff matrices $f^{ij}(\cdot, \cdot | \tau_t^i, \tau_t^j)$ in large action spaces;
iv. Allowing transfer/generalization to different CGs (as suggested in Guestrin et al., 2002b); and
v. Allowing the use of privileged information like the global state during training.

Restricting the payoff's input (i) and sharing parameters (ii), i.e., $f_\theta^i(u^i|\tau_t^i) \approx f_\theta^v(u^i|\boldsymbol{h}_t^i)$ and $f^{ij}(a^i, a^j|\tau_t^i, \tau_t^j) \approx f_\phi^e(a^i, a^j|\boldsymbol{h}_t^i, \boldsymbol{h}_t^j)$, improves sample efficiency significantly. Both utilities and payoffs share further parameters through a common RNN $\boldsymbol{h}_t^i := h_\psi(\cdot|\boldsymbol{h}_{t-1}^i, o_t^i, a_{t-1}^i)$, initialized with $\boldsymbol{h}_0^i := h_\psi(\cdot|\boldsymbol{0}, o_0^i, \boldsymbol{0})$. Note the difference to Castellini et al. (2019), which do not condition on the state or the agents' histories, and learn independent functions for each payoff.

The payoff function f_ϕ^e has A^2 outputs, $A := |\cup_{i=1}^n \mathcal{A}^i|$, one for each possible joint action of the agent pair. For example, each agent in a StarCraft II map with 8 enemies has 13 actions (SMAC, Samvelyan et al., 2019), which yields 169 outputs of f_ϕ^e. As only executed action-pairs are updated during Q-learning, the parameters of many outputs remain unchanged for long stretches of time, while the underlying RNN h_ψ keeps evolving. This can slow down training and affect message passing. To reduce the number of parameters and improve the frequency in which they are updated, we propose a *low-rank approximation*[6] of the payoff (iii) with rank K, similar to Chen et al. (2018):

$$f_\phi^e(a^i, a^j|\boldsymbol{h}_t^i, \boldsymbol{h}_t^j) := \sum_{k=1}^{K} \hat{f}_{\hat{\phi}}^k(a^i|\boldsymbol{h}_t^i, \boldsymbol{h}_t^j)\, \bar{f}_{\bar{\phi}}^k(a^j|\boldsymbol{h}_t^i, \boldsymbol{h}_t^j). \quad (5)$$

The approximation can be computed in one forward pass with $2KA$ outputs and parameters $\phi := \{\hat{\phi}, \bar{\phi}\}$. Note that a rank $K = \min\{|\mathcal{A}^i|, |\mathcal{A}^j|\}$ approximation does not restrict the output's expressiveness, while lower ranks share parameters and updates to speed up learning.

To support further research in transfer between tasks (iv), the represented value function must generalize to new topologies (i.e., zero-shot transfer). This requires DCG to be invariant to reshuffling of agent indices. We solve this by averaging payoffs computed from both agents' perspectives.[7] However, this paper does not evaluate (iv) and we leave the transfer of a learned DCG onto different graphs/topologies to future work. The DCG Q-value function is:

$$q_{\theta\phi\psi}^{\text{DCG}}(\boldsymbol{\tau}_t, \boldsymbol{a}) := \frac{1}{|\mathcal{V}|} \sum_{i=1}^{n} \overbrace{f_\theta^v(a^i|\boldsymbol{h}_t^i)}^{f_{i,a^i}^{\text{V}}} + \frac{1}{2|\mathcal{E}|} \sum_{\{i,j\}\in\mathcal{E}} \Big(\underbrace{f_\phi^e(a^i, a^j|\boldsymbol{h}_t^i, \boldsymbol{h}_t^j) + f_\phi^e(a^j, a^i|\boldsymbol{h}_t^j, \boldsymbol{h}_t^i)}_{f_{\{i,j\},a^i,a^j}^{\text{E}}} \Big). \quad (6)$$

However, some tasks allow access to privileged information like the global state $s_t \in \mathcal{S}$ during training (but not execution). We therefore propose in (v) to use this information in a *privileged bias function* $v_\varphi : \mathcal{S} \to \mathbb{R}$ with parameters φ:

$$q_{\theta\phi\psi\varphi}^{\text{DCG-S}}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}) := q_{\theta\phi\psi}^{\text{DCG}}(\boldsymbol{\tau}_t, \boldsymbol{a}) + v_\varphi(s_t). \quad (7)$$

We call this approach DCG-S (similar to VDN-S from Rashid et al., 2018) and train both variants end-to-end with the DQN loss in Section 2.1 and Double Q-learning (van Hasselt et al., 2016). Given the tensors (multi-dimensional arrays) $\boldsymbol{f}^{\text{V}} \in \mathbb{R}^{|\mathcal{V}|\times A}$ and $\boldsymbol{f}^{\text{E}} \in \mathbb{R}^{|\mathcal{E}|\times A\times A}$, where all unavailable actions are set to $-\infty$, the Q-value can be maximized by message passing as defined in (3) and (4). The detailed procedures of computing the tensors (Algorithm 1), the Q-value (Algorithm 2) and greedy action selection (Algorithm 3) are given in the appendix. Note that we do not propagate gradients through the message passing loop, as DQN maximizes the value computed by the target network.

The key benefit of DCG lies in its ability to prevent *relative overgeneralization* during the exploration of agents: take the example of two hunters who have cornered their prey. The prey is dangerous and attempting to catch it alone can lead to serious injuries. From the perspective of each hunter, the expected reward for an attack depends on the actions of the other agent, who initially behaves randomly. If the punishment for attacking alone outweighs the reward for catching the prey, agents that cannot represent the value for joint actions (QMIX, VDN, IQL) cannot learn the optimal policy. However, estimating a value function over the joint action space (as in QTRAN) can be equally prohibitive,

[6] Similar to how singular values and vectors represent matrices.

[7] Permutation invariance requires the payoff matrix $\boldsymbol{f}^{ij}$, of dimensionality $|\mathcal{A}^i|\times|\mathcal{A}^j|$, to be the same as $(\boldsymbol{f}^{ji})^\top$ with swapped inputs. We enforce this by taking the average of both. This retains the ability to learn asymmetric payoff matrices $\boldsymbol{f}^{ij} \neq (\boldsymbol{f}^{ij})^\top$.

as it requires many more samples for the same prediction quality. DCG provides a flexible function class between these extremes that can be tailored to the task at hand.

4. Related Work

Oroojlooy jadid & Hajinezhad (2019) provide a general overview of cooperative deep MARL. Independent Q-learning (IQL Tan, 1993) decentralizes the agents' policy by modeling each agent as an independent Q-learner. However, the task from the perspective of a single agent becomes nonstationary as other agents change their policies. Foerster et al. (2017) show how to stabilize IQL when using experience replay buffers. Another approach to decentralized agents is *centralized training and decentralized execution* (Kraemer & Banerjee, 2016) with a *factored value function* (Koller & Parr, 1999). Value decomposition networks (VDN, Sunehag et al., 2018) perform central Q-learning with a value function that is the sum of independent utility functions for each agent (Figure 1a). The greedy policy can be executed by maximizing each utility independently. QMIX (Rashid et al., 2018) improves upon this approach by combining the agents' utilities with a mixing network, which is monotonic in the utilities and depends on the global state. This allows different mixtures in different states and the central value can be maximized independently due to monotonicity. All of these approaches are derived in Appendix A.1 and can use parameter sharing between the value/utility functions. However, they represent the joint value with independent values/utilities and are therefore susceptible to relative overgeneralization. We demonstrate this by comparing DCG with all the above algorithms.

Another straightforward way to decentralize in MARL is to define the joint policy as a product of independent agent policies. This lends itself to the actor-critic framework, where the critic is discarded during execution and can therefore condition on the global state and all agents' actions during training. Examples are MADDPG (Lowe et al., 2017) for continuous actions and COMA (Foerster et al., 2018) for discrete actions. Wei et al. (2018) specifically investigate relative overgeneralization in continuous multi-agent tasks and show improvement over MADDPG by introducing policy entropy regularization. MACKRL (Schröder de Witt et al., 2019) follows the approach in Foerster et al. (2018), but uses *common knowledge* to coordinate agents during centralized training. Son et al. (2019) define QTRAN, which also has a centralized critic but uses a greedy actor w.r.t. a VDN factorized function. The corresponding utility functions are distilled from the critic under constraints that ensure proper decentralization. Böhmer et al. (2019) present another approach to decentralize a centralized value function, which is locally maximized by coordinate ascent and decentralized by training IQL agents from the same replay buffer. Centralized joint Q-value functions do not allow parameter sharing to the same extent as value factorization, and we compare DCG to QTRAN to demonstrate the advantage in sample efficiency. Nonetheless, DCG value factorization can in principle be applied to any of the above centralized critics to equally improve sample efficiency at the same cost of representational capacity.

Other work deals with huge numbers of agents, which requires additional assumptions to reduce the sample complexity. For example, Yang et al. (2018) introduce *mean-field multi-agent learning* (MF-MARL), which factors a tabular value function for hundreds of agents into pairwise payoff functions between neighbors in a uniform grid of agents. These payoffs share parameters similar to DCG. Chen et al. (2018) introduce a value factorization (FQL) for a similar setup based on a low-rank approximation of the joint value. This approach is restricted by uniformity assumptions between agents, but otherwise uses parameter sharing similar to DCG. The value function cannot be maximized globally and must be locally maximized with coordinate ascent. These techniques are designed for much larger sets of agents and the specific assumptions and design choices are not well suited to the tasks considered in this paper. To demonstrate this, we compare against a low-rank joint value decomposition (called `LRQ`), similar to Chen et al. (2018).

Coordination graphs (CG) have been extensively studied in multi-agent robotics with given payoffs (e.g. Rogers et al., 2011; Yedidsion et al., 2018). Van der Pol & Oliehoek (2016) learn a pairwise payoff function for traffic light control of connected intersections with DQN, which is used for all edges in a CG of intersections. Sparse cooperative Q-learning (SCQL, Kok & Vlassis, 2006) uses CG in discrete state and action spaces by representing all utility and payoff functions as tables. However, the tabular approach restricts practical application of SCQL to tasks with few agents and small state and action spaces. Castellini et al. (2019) use neural networks to approximate payoff functions, but only in non-sequential games, and require a unique function for each edge in the CG. DCG addresses for the first time the question how CG can efficiently solve tasks with large state and action spaces, by introducing parameter sharing between all payoffs (as in VDN/QMIX), conditioning on local information (as in MF-MARL) and using low-rank approximation of the payoffs' outputs (as in FQL).

Graph Neural Networks (GNN, Battaglia et al., 2018) are architectures to approximate functions on annotated graphs by *learning* some message passing over the graph's edges. GNN can thus be used to estimate a joint Q-value in MARL, which conditions on graphs annotated with the observations of all agents. In contrast to the fixed message passing of CG, however, GNN would have to *learn* the joint maximization required for Q-learning, which would require additional

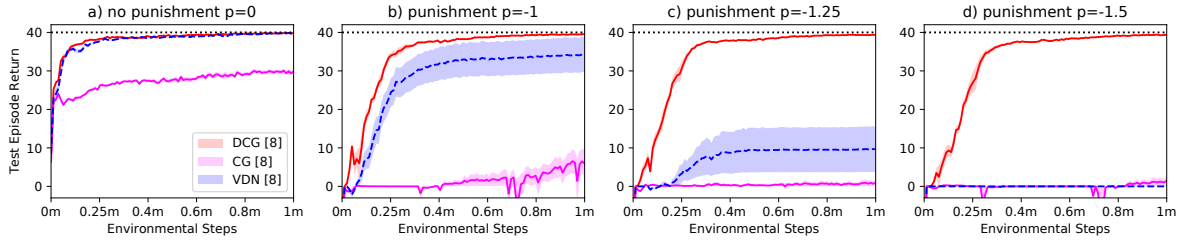

Figure 2. Influence of punishment p for attempts to catch prey alone on greedy test episode return (mean and shaded standard error, [number of seeds]) in the *relative overgeneralization* task where 8 agents hunt 8 prey (dotted line denotes best possible return). Fully connected DCG (DCG) are able to represent the value of joint actions, which leads to a better performance for larger p, where DCG without edges (VDN) has to fail eventually ($p < -1$). CG without parameter sharing (CG), learn very slowly due to sample inefficiency.

losses and might not be feasible in practice. Current MARL literature uses GNN therefore either as independent (IQL) value functions (Jiang et al., 2020; Luo et al., 2019) or as a joint critic in actor-critic frameworks (Tacchetti et al., 2019; Liu et al., 2019; Malysheva et al., 2019).

5. Empirical Results

In this section we compare the performance of DCG with various topologies (see Table 1) to the state-of-the-art algorithms QTRAN (Son et al., 2019), QMIX (Rashid et al., 2018), VDN (Sunehag et al., 2018) and IQL (Tan, 1993). We also compare a CG baseline without parameter sharing between payoffs and utilities (CG, an extension of Castellini et al., 2019), and a low-rank approximation of the joint value of all agents (LRQ, similar to Chen et al., 2018) that is maximized by coordinate ascent. Both baselines condition on a shared RNN that summarizes all agents' histories. Lastly, we investigate how well the DCG algorithm performs without any parameter sharing (DCG (nps)). All algorithms are implemented in the PYMARL framework (Samvelyan et al., 2019); a detailed description can be found in the appendix.

We evaluate these methods in two complex grid-world tasks and challenging Starcraft II micromanagement tasks from the StarCraft Multi-Agent Challenge (SMAC, Samvelyan et al., 2019). The first grid-world task formulates *relative overgeneralization* as a family of predator-prey tasks and the second investigates how *artificial decentralization* can hurt tasks that demand non-local coordination between agents. In the latter case, decentralized value functions (QMIX, VDN, IQL) cannot learn coordinated action selection between agents that cannot see each other directly and thus converge to a suboptimal policy. StarCraft II presents a challenging real-world problem with privileged information during training, and we compare DCG and DCG-S on 6 levels with varying complexity.

DCG	$\mathcal{E} := \big\{\{i,j\} \,\big|\, 1 \leq i < n, i < j \leq n\big\}$
CYCLE	$\mathcal{E} := \big\{\{i,(i \bmod n)+1\} \,\big|\, 1 \leq i \leq n\big\}$
LINE	$\mathcal{E} := \big\{\{i,i+1\} \,\big|\, 1 \leq i < n\big\}$
STAR	$\mathcal{E} := \big\{\{1,i\} \,\big|\, 2 \leq i \leq n\big\}$
VDN	$\mathcal{E} := \varnothing$

Table 1. Tested graph topologies for DCG.

5.1. Relative Overgeneralization

To model relative overgeneralization, we consider a partially observable grid-world predator-prey task: 8 agents have to hunt 8 prey in a 10×10 grid. Each agent can either move in one of the 4 compass directions, remain still, or try to catch any adjacent prey. Impossible actions, i.e., moves into an occupied target position or catching when there is no adjacent prey, are treated as unavailable. The prey moves by randomly selecting one available movement or remains motionless if all surrounding positions are occupied. If two adjacent agents execute the *catch* action, a prey is caught and both the prey and the catching agents are removed from the grid. An agent's observation is a 5×5 sub-grid centered around it, with one channel showing agents and another indicating prey. Removed agents and prey are no longer visible and removed agents receive a special observation of all zeros. An episode ends if all agents have been removed or after 200 time steps. Capturing a prey is rewarded with $r = 10$, but unsuccessful attempts by single agents are punished by a negative reward p. The task is similar to one proposed by Son et al. (2019), but significantly more complex, both in terms of the optimal policy and in the number of agents.

To demonstrate the effect of relative overgeneralization, Figure 2 shows the average return of greedy test episodes for varying punishment p as mean and standard error over 8 independent runs. In tasks without punishment ($p = 0$ in Figure 2a), fully connected DCG (DCG, solid) performs as well as DCG without edges (VDN, dashed). However, for stronger punishment VDN becomes more and more unreliable, which is visible in the large standard errors in Figures 2b and 2c, until it fails completely for $p \leq -1.5$

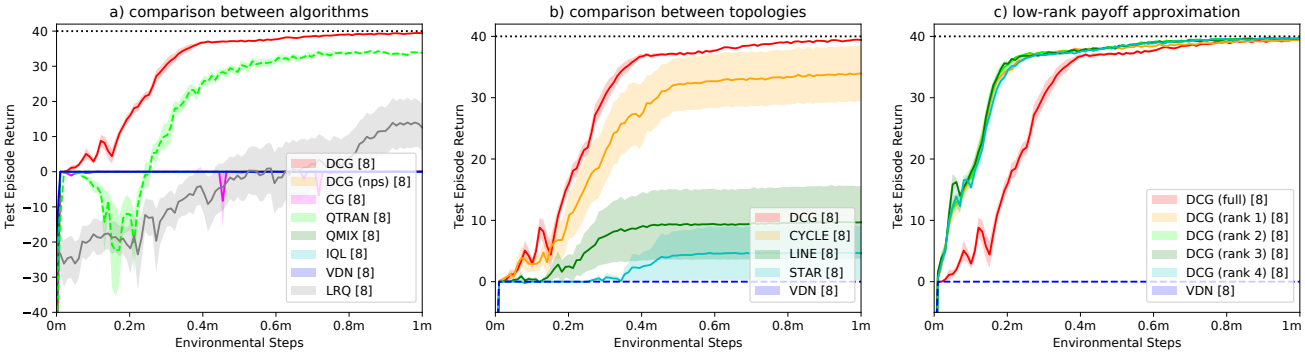

Figure 3. Greedy test episode return for the coordination task of Figure 2 with punishment $p = -2$. Comparison (a) to baseline algorithms; (b) between DCG topologies; (c) of different low-rank payoff approximations. Note that QMIX, IQL and VDN (dashed) do not solve the task (return 0) due to *relative overgeneralization*. CG, QTRAN and LQR ($K = 64$) *could* represent the joint value, but are sample inefficient due to the large joint action spaces. Note that without parameter sharing, DCG (nps) suffers the same fate. The reliability of DCG depends on the CG-topology: all seeds with fully connected DCG solved the task, but the high standard error for CYCLE, LINE and STAR topologies is caused by some seeds succeeding while others fail completely. Low-rank approximation (DCG (rank K)) dramatically improves sample efficiency without any significant impact on performance.

in Figure 2d. This is due to relative overgeneralization, as VDN cannot represent the values of joint actions during exploration. Note that a coordination graph (CG), where utilities and payoffs condition on all agents' observations, *can* represent the value but struggles to learn the task without parameter sharing. DCG, on the other hand, converges reliably to the optimal solution (dotted line).

Figure 3a shows how well DCG performs in comparison to the baseline algorithms in Appendix A.1 for a strong punishment of $p = -2$. Note that QMIX, IQL and VDN completely fail to learn the task (return 0) due to their restrictive value factorization. While CG could in principle learn the same policy as DCG, the lack of parameter sharing hurts performance as in Figure 2. QTRAN estimates the values with a centralized function, which conditions on all agents' actions, and can therefore learn the task. However, QTRAN requires more samples before a useful policy can be learned than DCG, due to the size of the joint action space. This is in line with the findings of Son et al. (2019), which required significantly more samples to learn a task with four agents than with two and also show the characteristic dip in performance with more agents. LRQ can also represent the joint value but learns extremely slow and with large deviations due to imperfect maximization by coordinate ascend. In comparison with QTRAN, CG and LRQ, fully connected DCG (DCG) learns near-optimal policies quickly and reliably.

We also investigate the performance of various DCG topologies defined in Table 1. Figure 3b shows that in particular the *reliability* of the achieved test episode return depends strongly on the graph topology. While all seeds of fully connected DCG succeed (DCG), DCG with CYCLE, LINE and STAR topologies have varying means with large standard errors. The high deviations are caused by some runs finding near-optimal policies, while others fail completely (return 0). One possible explanation is that for the failed seeds the rewarded experiences, observed in the initial exploration, are only amongst agents that do not share a payoff function. Due to relative overgeneralization, the learned greedy policy no longer explores *catch* actions and existing payoff functions cannot experience the reward for coordinated actions anymore. It is therefore not surprising that fully connected graphs perform best, as they represent the largest function class and require the fewest assumptions. The topology also has little influence on the runtime of DCG, due to efficient batching on the GPU.

The tested fully connected DCG only considers pairwise edges. Hyper-edges between more than two agents (Figure 1c) would yield even richer value representations, but would also require more samples to sufficiently approximate the payoff functions. This effect can be seen in the slower learning QTRAN and LRQ results in Figure 3a.

5.2. Artificial Decentralization

The choice of decentralized value functions is in some cases purely artificial: it is motivated by the huge joint action spaces and not because the task actually requires decentralized execution. While this often works surprisingly well, we want to investigate how existing algorithms deal with tasks that cannot be fully decentralized. One obvious case in which decentralization must fail is when the optimal policy cannot be represented by utility functions alone. For example, decentralized policies behave sub-optimally in tasks where the optimal policy would condition on multiple agents' observations in order to achieve the best return. Payoff functions in DCG, on the other hand, condition on pairs of agents and can thus represent a richer class of policies.

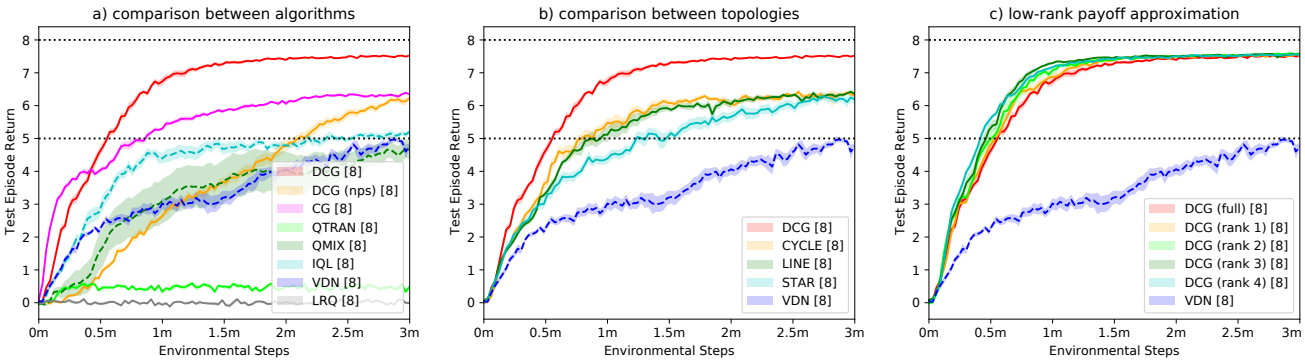

Figure 4. Greedy test episode return (mean and shaded standard error, [number of seeds]) in a *non-decentralizable task* where 8 agents hunt 8 prey: (a) comparison to baseline algorithms; (b) comparison between DCG topologies; (c) comparison of low-rank payoff approximations. The prey turns randomly into punishing ghosts, which are indistinguishable from normal prey. The prey status is only visible at an indicator that is placed randomly at each episode in one of the grid's corners. `QTRAN`, `QMIX`, `IQL` and `VDN` learn decentralized policies, which are at best suboptimal in this task (around lower dotted line). Fully connected `DCG` and `CG` can learn a near-optimal policy (upper dotted line denotes best possible return), but without parameter sharing `DCG (nps)` and `CG` yield sub-optimal performance in comparison to `DCG`. In this task low-rank approximations only marginally increase sample efficiency.

Note that dependencies on more agents can be modeled as hyper-edges in the DCG (Figure 1c), but this hurts the sample efficiency as discussed above.

We evaluate the advantage of a richer policy class with a variation of the above predator-prey task. To disentangle the effects of relative overgeneralization, in this task prey can be caught by only one agent (without punishment). Unbeknownst to the agent, however, a fair coin toss decides at each time step whether catching the prey is rewarding ($r=1$) or punishing ($r=-1$). The current coin flip can be observe through an additional feature, which is placed in a random corner at the beginning of each episode. Due to the short visibility range of the agents, the feature is only visible in one of the 9 positions closest to its corner.

Figure 4a shows the performance of `QTRAN`, `QMIX`, `IQL` and `VDN`, all of which have decentralized policies, in comparison to fully connected `DCG` and `CG`. The baseline algorithms have to learn a policy that first identifies the location of the indicating feature and then herds prey into that corner, where the agent is finally able to catch it without risk. By contrast, DCG and CG can learn a policy where one agent finds the indicator, allowing all agents that share an edge to condition their payoffs on that agent's current observation. As a result, this policy can catch prey much more reliably, as seen in the high performance of `DCG` compared to all baseline algorithms. Interestingly, as `CG` conditions on all agents' histories simultaneously, the baseline shows an advantage in the beginning but then learns more slowly and reaches a significantly lower performance. The joint value of `QTRAN` conditions on all observations, but the algorithm's constraints enforce the greedy policy to be consistent with a VDN factorized value function, which appears to prevent a good performance. `LRQ`'s factorization architecture appears too unstable to learn anything here. We also investigate the influence of the DCG topologies in Table 1, shown in Figure 4b. While other topologies do not reach the same performance as fully connected DCG, they still learn a policy that significantly outperforms all baseline algorithms, around the same performance as fully connected `CG`.

5.3. Low-Rank Approximation

While the above experiments already show a significant advantage of DCG with independent payoff outputs for each action pair, we observe performance issues on StarCraft II maps with this architecture. The most likely cause is the difference in the number of actions per agent: predator-prey agents choose between $|\mathcal{A}^i| = 6$ actions, whereas SMAC agents on comparable maps with 8 enemies have $|\mathcal{A}^i| = 13$ actions. While payoff matrices with 36 outputs in predator-prey appear reasonable to learn, 169 outputs in StarCraft II would require significantly more samples to estimate the payoff of each joint-action properly.

Figures 3c and 4c show the influence of *low-rank payoff approximation* (Equation 5, $K \in \{1, \dots, 4\}$) on the predator-prey tasks from previous subsections. Figure 3c shows that any low-rank approximation (`DCG (rank` K`)`) significantly improves the sample efficiency over the default architecture with independent payoffs for each action pair (`DCG (full)`). The improvement in Figure 4c is less impressive, but shows even rank $K = 1$ approximations (`DCG (rank 1)`) perform slightly better than `DCG (full)`.

5.4. Scaling Up to StarCraft II

The default architecture of DCG with independent payoffs for each action pair performs poorly in StarCraft II. We

therefore test a $K=1$ low-rank payoff approximation DCG with (DCG-S) and without (DCG) privileged information bias function v_φ, defined in (7), on six StarCraft II maps (from SMAC, Samvelyan et al., 2019). We report all learning curves in Figure 8 of the appendix and show as an example the *super hard* map MMM2 in Figure 5.

DCG is expected to yield an advantage on maps that struggle with relative overgeneralization, which should prevent VDN from learning. We observe on almost all maps that DCG and VDN perform similar. Also, adding privileged information improves performance for DCG-S and the corresponding VDN-S (Rashid et al., 2018) in many cases.

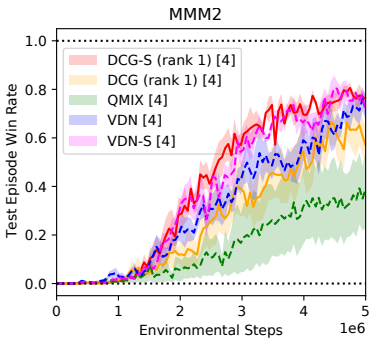

Figure 5. Win rate of test episodes on the SMAC map MMM2. Both DCG and DCG-S use rank 1 approximation.

We conclude from the results presented in the appendix that, in all likelihood, the SMAC benchmark does not suffer from relative overgeneralization. However, the fact that DCG-S matches QMIX, the state-of-the-art on SMAC, demonstrates that the algorithm scales to complex domains like StarCraft II. Furthermore, DCG and DCG-S perform comparable to their VDN counterparts. This demonstrates that the added payoffs and message passing, which allowed to overcome relative overgeneralization in Section 5.1, do not affect the algorithm's sample efficiency. This is a clear advantage over prior CG methods (CG in Figure 2, Castellini et al., 2019).

6. Conclusions & Future Work

This paper introduces the *deep coordination graph* (DCG), an architecture for value factorization that is specified by a *coordination graph* (CG) and can be maximized by message passing. We evaluate deep Q-learning with DCG and show that the architecture enables learning of tasks where *relative overgeneralization* causes all decentralized baselines to fail, whereas centralized critics are much less sample efficient than DCG. We also demonstrate that artificial decentralization can lead to suboptimal behavior in all compared methods except DCG. Our method significantly improves over existing CG methods and allows for the first time to use CG in tasks with large state and action spaces. Fully connected DCG performed best in all experiments and should be preferred in the absence of prior knowledge about the task. The computational complexity of this topology scales quadratically, which is a vast improvement over the exponential scaling of joint value estimates. Additionally, we introduce a low-rank payoff approximation for large action spaces and a privileged bias function (DCG-S). Evaluated on StarCraft II micromanagement tasks, DCG-S performs competitive with the state-of-the-art QMIX. DCG can also be defined with hyper-edges that connect more than two agents. Similar to our LRQ baseline, low-rank approximation can be used to approximate the payoff of high-order hyper-edges, and coordinate ascend can maximize them locally. Furhtermore, due to its permutation invariance, DCG has the potential to transfer/generalize to different graphs/topologies. This would in principle allow the training of DCG on dynamically generated graphs (e.g. using an attention mechanism, Liu et al., 2019). By including hyper-edges with varying degrees, one could allow the agents to flexibly decide in each state with whom they want to coordinate. We plan to investigate this in future work.

Acknowledgments

The authors would like to thank Tabish Rashid for his QTRAN implementation. This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement number 637713), the National Institutes of Health (grant agreement number R01GM114311), EPSRC/MURI grant EP/N019474/1, the JP Morgan Chase Faculty Research Award and a generous equipment grant from NVIDIA. Vitaly Kurin has also been supported by the Samsung R&D Institute UK studentship.

References

Alonso-Mora, J., Baker, S., and Rus, D. Multi-robot formation control and object transport in dynamic environments via constrained optimization. *International Journal of Robotics Research*, 36(9):1000–1021, 2017a.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017b.

Bacoyannis, V., Glukhov, V., Jin, T., Kochems, J., and Song, D. R. Idiosyncrasies and challenges of data driven learning in electronic trading. In *NeurIPS Workshop on Challenges and Opportunities for AI in Financial Services*, 2018. URL https://arxiv.org/abs/1811.09549.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li,

Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks, 2018.

Behbahani, F., Shiarlis, K., Chen, X., Kurin, V., Kasewa, S., Stirbu, C., Gomes, J., Paul, S., Oliehoek, F. A., Messias, J., and Whiteson, S. Learning from demonstration in the wild. In *IEEE International Conference on Robotics and Automation*, pp. 775–781, 2019.

Böhmer, W., Rashid, T., and Whiteson, S. Exploration with unreliable intrinsic reward in multi-agent reinforcement learning. *CoRR*, abs/1906.02138, 2019. URL http://arxiv.org/abs/1906.02138. Presented at the ICML *Exploration in Reinforcement Learning* workshop.

Castellini, J., Oliehoek, F. A., Savani, R., and Whiteson, S. The representational capacity of action-value networks for multi-agent reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pp. 1862–1864, 2019. URL http://www.ifaamas.org/Proceedings/aamas2019/pdfs/p1862.pdf.

Chen, Y., Zhou, M., Wen, Y., Yang, Y., Su, Y., Zhang, W., Zhang, D., Wang, J., and Liu, H. Factorized q-learning for large-scale multi-agent systems. *CoRR*, abs/1809.03738, 2018. URL http://arxiv.org/abs/1809.03738.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning*, 2014. URL http://arxiv.org/abs/1412.3555.

Correa-Posada, C. M. and Sánchez-Martin, P. Integrated power and natural gas model for energy adequacy in short-term operation. *IEEE Transactions on Power Systems*, 30 (6):3347–3355, 2015.

Crick, C. and Pfeffer, A. Loopy belief propagation as a basis for communication in sensor networks. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pp. 159–166. Morgan Kaufmann Publishers Inc., 2002.

Dotoli, M., Fay, A., Miśkowicz, M., and Seatzu, C. Advanced control in factory automation: a survey. *International Journal of Production Research*, 55(5):1243–1259, 2017.

Foerster, J., Assael, Y., de Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS 2016: Proceedings of the Thirtieth Annual Conference on Neural Information Processing Systems*, 2016. URL http://www.cs.ox.ac.uk/people/shimon.whiteson/pubs/foersternips16.pdf.

Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., and Whiteson, S. Stabilising experience replay for deep multi-agent reinforcement learning. In *ICML 2017: Proceedings of the Thirty-Fourth International Conference on Machine Learning*, 2017. URL http://www.cs.ox.ac.uk/people/shimon.whiteson/pubs/foerstericml17.pdf.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 2974–2982. AAAI Press, 2018. URL https://arxiv.org/abs/1705.08926.

Guestrin, C., Lagoudakis, M., and Parr, R. Coordinated reinforcement learning. In *ICML*, volume 2, pp. 227–234, 2002a.

Guestrin, C., Venkataraman, S., and Koller, D. Context-specific multiagent coordination and planning with factored mdps. In *Eighteenth National Conference on Artificial Intelligence*, pp. 253–259, 2002b.

Hausknecht, M. J. and Stone, P. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposia*, pp. 29–37, 2015. URL http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673.

Jiang, J., Dun, C., Huang, T., and Lu, Z. Graph convolutional reinforcement learning. In *Internation Conference on Learning Representation*, 2020. URL https://arxiv.org/abs/1810.09202.

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

Koller, D. and Parr, R. Computing factored value functions for policies in structured mdps. In *Proceedings of IJCAI*, pp. 1332–1339, 1999.

Kraemer, L. and Banerjee, B. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, 1992.

Liu, Y., Wang, W., Hu, Y., Hao, J., Chen, X., and Gao, Y. Multi-agent game abstraction via graph attention neural network, 2019. URL https://arxiv.org/abs/1911.10715.

Lowe, R., WU, Y., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for

mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 30*, pp. 6379–6390. 2017. URL https://arxiv.org/pdf/1706.02275.pdf.

Luo, T., Subagdja, B., Wang, D., and Tan, A. Multi-agent collaborative exploration through graph-based deep reinforcement learning. In *2019 IEEE International Conference on Agents (ICA)*, pp. 2–7, 2019. URL http://ica2019.crowdscience.org/wp-content/uploads/2019/10/Full-27-PID6151059.pdf.

Malysheva, A., Kudenko, D., and Shpilman, A. Magnet: Multi-agent graph network for deep multi-agent reinforcement learning. In *Adaptive and Learning Agents Workshop at AAMAS*, 2019. URL https://ala2019.vub.ac.be/papers/ALA2019_paper_17.pdf.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533, February 2015.

Murphy, K. P., Weiss, Y., and Jordan, M. I. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 467–475. Morgan Kaufmann Publishers Inc., 1999.

Oliehoek, F. A. and Amato, C. *A concise introduction to decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276, 9783319289274.

Oroojlooy jadid, A. and Hajinezhad, D. A review of cooperative multi-agent deep reinforcement learning. *CoRR*, abs/1908.03963, 2019. URL http://arxiv.org/abs/1908.03963.

Panait, L., Luke, S., and Wiegand, R. P. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.

Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 4292–4301, 2018.

Rogers, A. C., Farinelli, A., Stranders, R., and Jennings, N. R. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011. ISSN 0004-3702. URL http://www.sciencedirect.com/science/article/pii/S0004370210001803.

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C.-M., Torr, P. H. S., Foerster, J., and Whiteson, S. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

Schröder de Witt, C. A., Förster, J. N., Farquhar, G., Torr, P. H., Böhmer, W., and Whiteson, S. Multi-agent common knowledge reinforcement learning. In *Advances in Neural Information Processing Systems 32*, pp. 9927–9939, 2019. URL https://arxiv.org/abs/1810.11702.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5887–5896, 2019. URL http://proceedings.mlr.press/v97/son19a.html.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 2085–2087, 2018.

Tacchetti, A., Song, H. F., Mediano, P. A. M., Zambaldi, V., Kramr, J., Rabinowitz, N. C., Graepel, T., Botvinick, M., and Battaglia, P. W. Relational forward models for multi-agent learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJlEojAqFm.

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

Van der Pol, E. and Oliehoek, F. A. Coordinated deep reinforcement learners for traffic light control. In *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*, December 2016. URL https://sites.google.com/site/malicnips2016/papers.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016. URL https://arxiv.org/pdf/1509.06461.pdf.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019.

Wainwright, M., Jaakkola, T., and Willsky, A. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166, 2004.

Watkins, C. and Dayan, P. Q-learning. *Machine Learning*, 8:279–292, 1992.

Wei, E., Wicke, D., Freelan, D., and Luke, S. Multiagent soft q-learning. In *AAAI Spring Symposium Series*, 2018. URL https://www.aaai.org/ocs/index.php/SSS/SSS18/paper/view/17508/15482.

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. Mean field multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 5571–5580, 2018. URL http://proceedings.mlr.press/v80/yang18d.html.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

Yedidsion, H., Zivan, R., and Farinelli, A. Applying max-sum to teams of mobile sensing agents. *Engineering Applications of Artificial Intelligence*, 71:87–99, 2018. ISSN 0952-1976. URL http://www.sciencedirect.com/science/article/pii/S0952197618300381.

A. Appendix

A.1. Baseline algorithms

All discussed algoorithms are implemented in the PyMARL framework (Samvelyan et al., 2019) and can be found at https://github.com/wendelinboehmer/dcg.

IQL *Independent Q-learning* (Tan, 1993) is a straightforward approach of value decentralization that allows efficient maximization by modeling each agent as an independent DQN $q_\theta^i(a^i|\tau_t^i)$. The value functions can be trained without any knowledge of other agents, which are assumed to be part of the environment. This violates the stationarity assumption of P and can become therefore instable (see e.g. Foerster et al., 2017). IQL is nonetheless widely used in practice, as parameter sharing between agents can make it very sample efficient.

Note that parameter sharing requires access to privileged information during training, called *centralized training and decentralized execution* (Foerster et al., 2016). This is particularly useful for actor-critic methods like MADDPG (Lowe et al., 2017), Multi-agent soft Q-learning (Wei et al., 2018), COMA (Foerster et al., 2018) and MACKRL (Schröder de Witt et al., 2019), where the centralized critic can condition on the underlying state s_t and the joint action $\boldsymbol{a}_t \in \mathcal{A}$.

VDN Another way to exploit centralized training is *value function factorization*. For example, value decomposition networks (VDN, Sunehag et al., 2018) perform centralized deep Q-learning on a joint Q-value function that factors as the sum of independent *utility functions* f^i, for each agent i:

$$q_\theta^{\text{VDN}}(\boldsymbol{\tau}_t, \boldsymbol{a}) \quad := \quad \sum_{i=1}^{n} f_\theta^i(a^i|\tau_t^i)\,. \tag{8}$$

This value function q^{VDN} can be maximized by maximizing each agent's utility f_θ^i independently.

QMIX (Rashid et al., 2018) improves upon this concept by factoring the value function as

$$q_{\theta\phi}^{\text{QMIX}}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}) \;:=\; \varphi_\phi\big(s_t, f_\theta^1(a^1|\tau_t^1), \ldots, f_\theta^n(a^n|\tau_t^n)\big)\,.$$

Here φ_ϕ is a *monotonic mixing hypernetwork* with non-negative weights that retains monotonicity in the inputs f_θ^i. Maximizing each utility f_θ^i therefore also maximizes the joint value q^{QMIX}, as in VDN. The mixing parameters are generated by a neural network, parameterized by ϕ, that condition on the state s_t, allowing different mixing of utilities in different states. QMIX improves performance over VDN, in particular in StarCraft II micromanagement tasks (SMAC, Samvelyan et al., 2019).

QTRAN Recently Son et al. (2019) introduced QTRAN, which learns the centralized critic of a greedy policy w.r.t. a VDN factorized function, which in turn is distilled from the critic by regression under constraints. The algorithm defines three value functions q^{VDN}, q and v, where $q(\boldsymbol{\tau}_t, \boldsymbol{a})$ is the centralized Q-value function, as in Section 2.1, and

$$v(\boldsymbol{\tau}_t) \quad := \quad \max q(\boldsymbol{\tau}_t, \cdot) - \max q^{\text{VDN}}(\boldsymbol{\tau}_t, \cdot)\,. \tag{9}$$

They prove that the greedy policies w.r.t. q and q^{VDN} are identical under the constraints:

$$q^{\text{VDN}}(\boldsymbol{\tau}_t, \boldsymbol{a}) - q(\boldsymbol{\tau}_t, \boldsymbol{a}) + v(\boldsymbol{\tau}_t) \quad \geq \quad 0\,, \tag{10}$$

$\forall \boldsymbol{a} \in \mathcal{A}, \forall \boldsymbol{\tau}_t \in \{(\mathcal{O}^i \times \mathcal{A}^i)^t \times \mathcal{O}^i\}_{i=1}^n$, with strict equality if and only if $\boldsymbol{a} = \arg\max q^{\text{VDN}}(\boldsymbol{\tau}_t, \cdot)$. QTRAN minimizes the parameters ϕ of the centralized asymmetric value $q_\phi^i(a^i|\boldsymbol{\tau}_t, \boldsymbol{a}^{-i})$, $\boldsymbol{a}^{-i} := (a^1, \dots, a^{i-1}, a^{i+1}, \dots, a^n)$, for each agent (which is similar to Foerster et al., 2018) with the combined loss $\mathcal{L}_{\text{TD}}$:

$$\mathcal{L}_{\text{TD}} := \mathbb{E}\Big[\tfrac{1}{nT}\sum_{t=0}^{T-1}\sum_{i=1}^{n}\Big(r_t + \gamma\, \bar{y}_{t+1}^i - q_\phi^i(a_t^i|\boldsymbol{\tau}_t, \boldsymbol{a}_t^{-i})\Big)^2\Big],$$

where $\bar{y}_t^i := q_{\bar{\phi}}^i(\bar{a}_t^i|\boldsymbol{\tau}_t, \bar{\boldsymbol{a}}_t^{-i})$ denotes the centralized asymmetric value and $\bar{\boldsymbol{a}}_{t+1} := \arg\max q_\theta^{\text{VDN}}(\boldsymbol{\tau}_{t+1}, \cdot), \forall t$, denotes what a greedy decentralized agent would have chosen. The decentralized value q_θ^{VDN} and the greedy difference v_ψ, with parameters θ and ψ respectively, are distilled by regression of the each q_ϕ^i in the constraints. First the equality constraint:

$$\mathcal{L}_{\text{OPT}} := \mathbb{E}\Big[\tfrac{1}{n(T+1)}\sum_{t=0}^{T}\sum_{i=1}^{n}\Big(q_\theta^{\text{VDN}}(\boldsymbol{\tau}_t, \bar{\boldsymbol{a}}_t) - \perp\bar{y}_t^i + v_\psi(\boldsymbol{\tau}_t)\Big)^2\Big],$$

where the 'detach' operator $\perp$ stops the gradient flow through q_ϕ^i. The inequality constraints are more complicated. In principle one would have to compute a loss for every action which has a negative left hand side in (10). Son et al. (2019) suggest to only constraint executed actions $\boldsymbol{a}_t$:

$$\mathcal{L}_{\text{NOPT}} := \mathbb{E}\Big[\tfrac{1}{n(T+1)}\sum_{t=0}^{T}\sum_{i=1}^{n}\Big(\min\Big\{0, \; q_\theta^{\text{VDN}}(\boldsymbol{\tau}_t, \boldsymbol{a}_t) - \perp q_\phi^i(a_t^i|\boldsymbol{\tau}_t, \boldsymbol{a}_t^{-i}) + v_\psi(\boldsymbol{\tau}_t)\Big\}\Big)^2\Big]. \tag{11}$$

We use this loss, called `QTRAN-base`, which performed better in our experiments than `QTRAN-alt` (see Son et al., 2019). The losses are combined to $\mathcal{L}_{\text{QTRAN}} := \mathcal{L}_{\text{TD}} + \lambda_{\text{OPT}}\mathcal{L}_{\text{OPT}} + \lambda_{\text{NOPT}}\mathcal{L}_{\text{NOPT}}$, with $\lambda_{\text{OPT}}, \lambda_{\text{NOPT}} > 0$.

CG To compare the effect of parameter sharing and restriction to local information in DCG, we evaluate a variation of Castellini et al. (2019) that can solve sequential tasks. In this baseline all agents share a RNN encoder of their belief over the current global state $\boldsymbol{h}_t := h_\psi(\cdot|\boldsymbol{h}_{t-1}, \boldsymbol{o}_t, \boldsymbol{a}_{t-1})$ with $\boldsymbol{h}_0 := h_\psi(\cdot|\boldsymbol{0}, \boldsymbol{o}_0, \boldsymbol{0})$, as introduced in Section 2.1. However, the parameters of the utility or payoff functions are not shared, that is, $\theta := \{\theta_i\}_{i=1}^n$ and $\phi := \{\phi_{ij}|\{i,j\} \in \mathcal{E}\}$. Each set of parameters θ_i and ϕ_{ij} represents one linear layer from $\boldsymbol{h}_t$ to $\mathcal{A}^i$ and $\mathcal{A}^i \times \mathcal{A}^j$ outputs, respectively. Other wise the baseline uses the same code as DCG, that is, Algorithms 1, 2 and 3.

LRQ As DCG uses low-rank approximation of the payoff outputs, it is a fair question how a low-rank approximation of the full joint value function (LRQ) would perform (akin to one hyper-edge shown in Figure 1c). This approach is similar to FQL (Chen et al., 2018), but we drop here the homogeneity assumptions between agents. Instead, we define the joint value function as a sum of K factors, which each are the product of n factor functions $\bar{f}^{ik}$, one for each agent:

$$q^{\text{LRQ1}}(\boldsymbol{\tau}_t, \boldsymbol{a}) := \sum_{k=1}^{K}\prod_{i=1}^{n}\bar{f}_\theta^{ik}(a^i|\boldsymbol{\tau}_t). \tag{12}$$

The joint histories of all agents $\boldsymbol{\tau}_t$ are encoded with a common RNN with 512 hidden neurons. In difference to DCG, LQR cannot be maximized by message passing. Instead we perform coordinate ascend by choosing a random joint action $\bar{\boldsymbol{a}}_0$ and iterating, $\forall i \in \{1, \dots, n\}$,

$$\bar{a}_{l+1}^i := \arg\max_{a' \in \mathcal{A}^i}\sum_{k=1}^{K}\bar{f}^{ik}(a'|\boldsymbol{\tau}_t)\prod_{j\neq i}\bar{f}^{jk}(\bar{a}_l^j|\boldsymbol{\tau}_t). \tag{13}$$

The iteration finishes if the value $q^{\text{LRQ1}}(\boldsymbol{\tau}_t, \bar{\boldsymbol{a}}_l)$ no longer increases of after a maximum of $l = 8$ iterations.

Experiments on the predator-prey tasks with $K \in \{1, 4, 16, 64, 256, 1024\}$ revealed that, due to the large input space of $\boldsymbol{\tau}_t$, the above approximation did not learn anything. To allow a better comparison, we use the same input restrictions and parameter sharing tricks as DCG, that is, we restrict the input of each factor function to the history of the corresponding agent and share all agents' parameters:

$$\boldsymbol{h}_0^i := \boldsymbol{0}, \qquad \boldsymbol{h}_t^i := h_\psi(\cdot|\boldsymbol{h}_{t-1}^i, o_t^i, a_{t-1}^i) \tag{14}$$

$$q^{\text{LRQ2}}(\boldsymbol{\tau}_t, \boldsymbol{a}) := \sum_{k=1}^{K}\prod_{i=1}^{n}\bar{f}_\theta^{k}(a^i|\boldsymbol{h}_t^i). \tag{15}$$

Figures 3 and 4 show that this architecture learns the task with $K = 64$, albeit slowly. Figure 6 demonstrates the effect of the number of factors K on the solution of the rela-

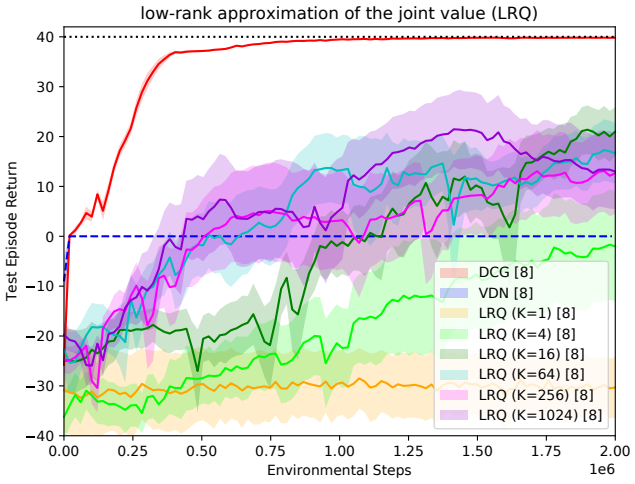

Figure 6. Low-rank approximation of the joint value function (LRQ) in the relative overgeneralization task of Section 5.1 for varying numbers of factors $K \in \{1, 4, 16, 64, 256, 1024\}$.

tive overgeneralization task of Section 5.1. Given enough factors, LRQ learns the task, albeit slowly and with a lot of variance between seeds, probably due to imperfect maximization by coordinate ascend.

A.2. DCG Algorithms

All algorithms defined in this paper are given in pseudocode on Page 16: Algorithm 1 computes the utility and payoff tensors, which are used by Algorithm 2 to compute the joint Q-value and by Algorithm 3 to return the joint actions that greedily maximize the joint Q-value. An open-source python implementation within the PyMARL framework (Samvelyan et al., 2019) can be found online at `https://github.com/wendelinboehmer/dcg`.

A.3. Hyper-parameters

All algorithms are implemented in the PYMARL framework (Samvelyan et al., 2019). We aimed to keep the hyper-parameters close to those given in the framework and consistent for all algorithms.

All tasks used discount factor $\gamma = 0.99$ and ϵ-greedy exploration, which was linearly decayed from $\epsilon = 1$ to $\epsilon = 0.05$ within the first $50,000$ time steps. Every 2000 time steps we evaluated 20 greedy test trajectories with $\epsilon = 0$. Results are plotted by first applying histogram-smoothing (100 bins) to each seed, and then computing the mean and standard error between seeds.

All methods are based on agents' histories, which were individually summarized with h_ψ by conditioning a linear layer of 64 neurons on the current observation and previous action, followed by a ReLU activation and a GRU (Chung et al., 2014) of the same dimensionality. Both layers' parameters are shared amongst agents, which can be identified by a one-hot encoded ID in the input. For the `CG` baseline, the linear layer and the GRU had $64n = 512$ neurons. This allows a fair comparison with DCG and also had the best final performance amongst tested dimensionalities $\{64, 256, 512, 1024\}$ in the task of Figure 4. Independent value functions q^i_θ (for `IQL`), utility functions f^v_θ (for `VDN/QMIX/QTRAN/DCG`) and payoff functions f^e_ϕ (for `DCG`) are linear layers from the GRU output to the corresponding number of actions. The hyper-network φ_ϕ of `QMIX` produces a mixing network with two layers connected with an ELU activation function, where the weights of each mixing-layer are generated by a linear hyper-layer with 32 neurons conditioned on the global state, that is, the full grid-world. For `QTRAN`, the critic q^i_ϕ computes the Q-value for an agent i by taking all agents' GRU outputs, all other agents' one-hot encoded actions, and the one-hot encoded agent ID i as input. The critic contains four successive linear layers with 64 neurons each and ReLU activations between them. The greedy difference v_ψ also conditions on all agents' GRU outputs and uses three successive linear layers with 64 neurons each and ReLU activations between them. After some coarse hyper-parameter exploration for `QTRAN` with $\lambda_{\text{OPT}}, \lambda_{\text{NOPT}} \in \{0.1, 1, 10\}$, we chose the loss parameters $\lambda_{\text{OPT}} = 1, \lambda_{\text{NOPT}} = 10$. The `LRQ` results in the main text used the state-encoding from `CG` and $K = 64$.

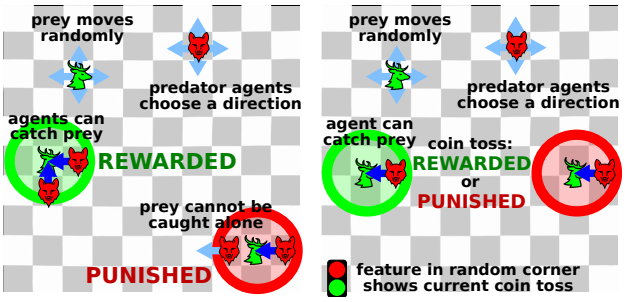

Figure 7. Illustrations of the *relative overgeneralization task* (left, Sec. 5.1) and the *artificial decentralization task* (right, Sec. 5.2).

All algorithms were trained with one RMSprop gradient step after each observed episode based on a batch of 32 episodes, which always contains the newest, from a replay buffer holding the last 500 episodes. The optimizer uses learning rate 0.0005, $\alpha = 0.99$ and $\epsilon = 0.00001$. Gradients with a norm ≥ 10 were clipped. The target network parameters were replaced by a copy of the current parameters every 200 episodes.

A.4. StarCraft II details

We kept all hyper-parameters the same and evaluated the six maps in Table 2. All maps are from SMAC (Samvelyan et al., 2019), except `micro_focus`, which was provided to us by the SMAC authors. The results for `DCG-S`, `DCG`, `QMIX` and `VDN` are given in Figure 8 on Page 15, where both DCG variants use a rank-1 payoff approximation. Note that

Name	Agents	Enemies	Diff.
`so_many_baneling`	7 Zealots	32 Banelings	easy
`8m_vs_9m`	8 Marines	9 Marines	hard
`3s_vs_5z`	3 Stalker 5 Zealots	5 Zealots	hard
`3s5z`	3 Stalker 5 Zealots	3 Stalker 5 Zealots	hard
`MMM2`	1 Medivac 2 Marauders 7 Marines	1 Medivac 3 Marauders 8 Marines	super hard
`micro_focus`	6 Hydralisks	8 Stalker	super hard

Table 2. Types of agents, enemies and difficulty of all tested StarCraft II maps for SMAC (Samvelyan et al., 2019).

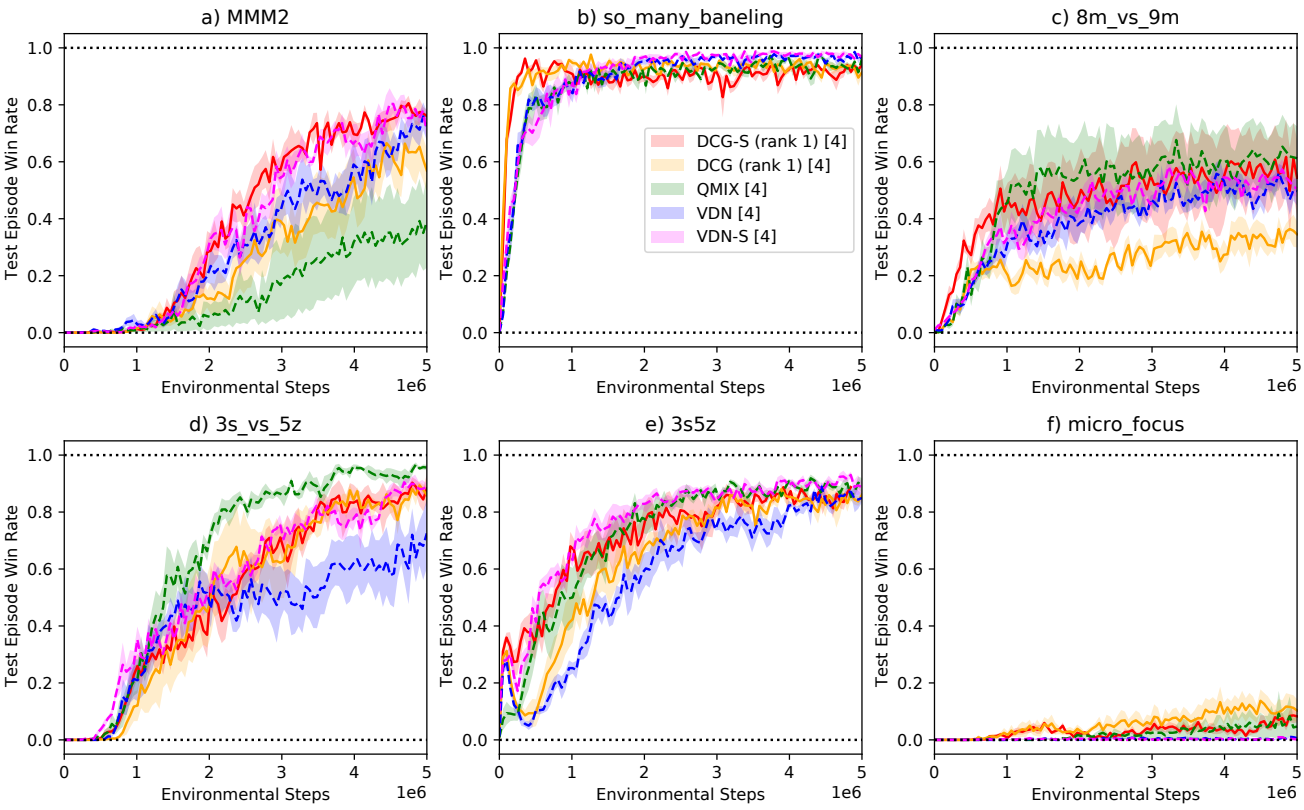

Figure 8. Cumulative reward for test episodes on SMAC maps (mean and shaded standard error, [number of seeds]) for `QMIX`, `VDN`, `VDN-S` and fully connected DCG with rank $K = 1$ payoff approximation (`DCG (rank 1)`) and additional state-dependent bias function (`DCG-S (rank 1)`).

our results differ from those in Samvelyan et al. (2019), due to slightly different parameters and an update after every episode. The latter differs from the original publication because we use the the `episode_runner` instead of the `parallel_runner` of PYMARL. These choices ended up improving the performance of QMIX significantly.

As expected, a direct comparison with the state-of-the-art method `QMIX` depends strongly on the StarCraft II map. On the one hand, `DCG-S` clearly outperforms `QMIX` on `MMM2` (Figure 8a), which is classified as *super hard* by SMAC. We also learn much faster on the *easy* map `so_many_baneling` (Figure 8b). On the other hand, `QMIX` performs better on the *hard* map `3s_vs_5z` (Figure 8d), which might be due to the low number of 3 agents. For that amount of agents, the added representational capacity of DCG may not improve the task as much as the non-linear state-dependent mixing of QMIX. It is hard to pin-point why state dependent mixing is an advantage here, though. However, given that `DCG-S` and `VDN-S` perform equally well on all maps except `so_many_baneling` indicates that the SMAC benchmark probably does not suffer much from the relative overgeneralization pathology.

Algorithm 1 Annotates a CG by computing the utility and payoff tensors (rank K approximation).

function ANNOTATE($\{\boldsymbol{h}^i_{t-1}, a^i_{t-1}, o^i_t\}_{i=1}^n, \mathcal{E}, \{\mathcal{A}^i\}_{i=1}^n, K \in \mathbb{N}$) ▷ $A := |\cup_i \mathcal{A}^i|$

- $\boldsymbol{f}^{\mathrm{V}} := \mathbf{0} \quad \in \quad \mathbb{R}^{n \times A}$ ▷ initialize utility tensor
- $\boldsymbol{f}^{\mathrm{E}} := \mathbf{0} \quad \in \quad \mathbb{R}^{|\mathcal{E}| \times A \times A}$ ▷ initialize payoff tensor
- **for** $i \in \{1, \dots, n\}$ **do** ▷ compute batch with all agents
  - $\boldsymbol{h}^i_t := h_\psi(\boldsymbol{h}^i_{t-1}, o^i_t, a^i_{t-1})$ ▷ new hidden state
  - $\boldsymbol{f}^{\mathrm{V}}_i \leftarrow f^v_\theta(\boldsymbol{h}^i_t) \quad \in \quad \mathbb{R}^A$ ▷ compute utility
  - **for** $a \in \{1, \dots, A\} \setminus \mathcal{A}^i$ **do** ▷ set unavailable actions ...
    - $\boldsymbol{f}^{\mathrm{V}}_{ia} \leftarrow -\infty$ ▷ ... to $-\infty$
- **for** $e = (i, j) \in \mathcal{E}$ **do** ▷ compute batch with all edges
  - **if** $K = 0$ **then** ▷ if no low-rank approximation
    - $\boldsymbol{f}^{\mathrm{E}}_e \leftarrow \frac{1}{2} f^e_\phi(\cdot, \cdot | \boldsymbol{h}^i_t, \boldsymbol{h}^j_t) + \frac{1}{2} f^e_\phi(\cdot, \cdot | \boldsymbol{h}^j_t, \boldsymbol{h}^i_t)^\top \quad \in \quad \mathbb{R}^{A \times A}$ ▷ symmetric payoffs
  - **else** ▷ if low-rank approximation
    - $[\hat{\mathbf{F}}, \bar{\mathbf{F}}] := f^e_\phi(\cdot, \cdot, \cdot | \boldsymbol{h}^i_t, \boldsymbol{h}^j_t) \quad \in \quad \mathbb{R}^{2 \times A \times K}$
    - $[\hat{\mathbf{F}}', \bar{\mathbf{F}}'] := f^e_\phi(\cdot, \cdot, \cdot | \boldsymbol{h}^j_t, \boldsymbol{h}^i_t) \quad \in \quad \mathbb{R}^{2 \times A \times K}$
    - $\boldsymbol{f}^{\mathrm{E}}_e \leftarrow \frac{1}{2} \hat{\mathbf{F}} \bar{\mathbf{F}}^\top + \frac{1}{2} \bar{\mathbf{F}}' \hat{\mathbf{F}}'^\top \quad \in \quad \mathbb{R}^{A \times A}$ ▷ symmetric payoffs
- **return** $\{\boldsymbol{h}^i_t\}_{i=1}^n, \boldsymbol{f}^{\mathrm{V}}, \boldsymbol{f}^{\mathrm{E}}$ ▷ return hidden states $\boldsymbol{h}^i_t$, utility tensor $\boldsymbol{f}^{\mathrm{V}}$ and payoff tensor $\boldsymbol{f}^{\mathrm{E}}$

Algorithm 2 Q-value computed from utility and payoff tensors (and potentially global state s_t).

function QVALUE($\boldsymbol{f}^{\mathrm{V}} \in \mathbb{R}^{|\mathcal{V}| \times A}, \boldsymbol{f}^{\mathrm{E}} \in \mathbb{R}^{|\mathcal{E}| \times A \times A}, \boldsymbol{a} \in \mathcal{A}, s_t \in \mathcal{S} \cup \{\varnothing\}$) ▷ $v_\varphi(\varnothing) = 0$

- **return** $\frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \boldsymbol{f}^{\mathrm{V}}_{ia^i} + \frac{1}{|\mathcal{E}|} \sum_{e=(i,j) \in \mathcal{E}} \boldsymbol{f}^{\mathrm{E}}_{ea^ia^j} + v_\varphi(s_t)$ ▷ return the Q-value of the given actions $\boldsymbol{a}$

Algorithm 3 Greedy action selection with k message passes in a coordination graph.

function GREEDY($\boldsymbol{f}^{\mathrm{V}} \in \mathbb{R}^{|\mathcal{V}| \times A}, \boldsymbol{f}^{\mathrm{E}} \in \mathbb{R}^{|\mathcal{E}| \times A \times A}, \mathcal{V}, \mathcal{E}, \{\mathcal{A}^i\}_{i=1}^{|\mathcal{V}|}, k$) ▷ $A := |\cup_i \mathcal{A}^i|$

- $\boldsymbol{\mu}^0, \bar{\boldsymbol{\mu}}^0 := \mathbf{0} \ \in \ \mathbb{R}^{|\mathcal{E}| \times A}$ ▷ messages forward ($\boldsymbol{\mu}$) and backward ($\bar{\boldsymbol{\mu}}$)
- $\boldsymbol{q}^0 := \frac{1}{|\mathcal{V}|} \boldsymbol{f}^{\mathrm{V}}$ ▷ initialize "Q-value" without messages
- $q_{\max} := -\infty; \quad \boldsymbol{a}_{\max} := \left[\arg\max_{a \in \mathcal{A}^i} q^0_{ai} \,\middle|\, i \in \mathcal{V} \right]$ ▷ initialize best found solution
- **for** $t \in \{1, \dots, k\}$ **do** ▷ loop with k message passes
  - **for** $e = (i, j) \in \mathcal{E}$ **do** ▷ update forward and backward messages
    - $\boldsymbol{\mu}^t_e := \max_{a \in \mathcal{A}^i} \left\{ (q^{t-1}_{ia} - \bar{\mu}^{t-1}_{ea}) + \frac{1}{|\mathcal{E}|} \boldsymbol{f}^{\mathrm{E}}_{ea} \right\}$ ▷ forward: maximize sender
    - $\bar{\boldsymbol{\mu}}^t_e := \max_{a \in \mathcal{A}^j} \left\{ (q^{t-1}_{ja} - \mu^{t-1}_{ea}) + \frac{1}{|\mathcal{E}|} (\boldsymbol{f}^{\mathrm{E}}_e)^\top_a \right\}$ ▷ backward: maximizes receiver
    - **if** `message_normalization` **then** ▷ to ensure converging messages
      - $\boldsymbol{\mu}^t_e \leftarrow \boldsymbol{\mu}^t_e - \frac{1}{|\mathcal{A}^j|} \sum_{a \in \mathcal{A}^j} \mu^t_{ea}$ ▷ normalize forward message
      - $\bar{\boldsymbol{\mu}}^t_e \leftarrow \bar{\boldsymbol{\mu}}^t_e - \frac{1}{|\mathcal{A}^i|} \sum_{a \in \mathcal{A}^i} \bar{\mu}^t_{ea}$ ▷ normalize backward message
  - **for** $i \in \mathcal{V}$ **do** ▷ update "Q-value" with messages
    - $\boldsymbol{q}^t_i := \frac{1}{|\mathcal{V}|} \boldsymbol{f}^{\mathrm{V}}_i + \sum_{e=(\cdot,i) \in \mathcal{E}} \boldsymbol{\mu}^t_e + \sum_{e=(i,\cdot) \in \mathcal{E}} \bar{\boldsymbol{\mu}}^t_e$ ▷ utility plus incoming messages
    - $a^t_i := \arg\max_{a \in \mathcal{A}^i} \{q^t_{ia}\}$ ▷ select greedy action of agent i
  - $q' \leftarrow \text{QVALUE}(\boldsymbol{f}^{\mathrm{V}}, \boldsymbol{f}^{\mathrm{E}}, \boldsymbol{a}^t, \varnothing)$ ▷ get true Q-value of greedy actions
  - **if** $q' > q_{\max}$ **then** $\{\boldsymbol{a}_{\max} \leftarrow \boldsymbol{a}^t; \ \ q_{\max} \leftarrow q'\}$ ▷ remember only the best actions
- **return** $\boldsymbol{a}_{\max} \in \mathcal{A}^1 \times \ldots \times \mathcal{A}^{|\mathcal{V}|}$ ▷ return actions that maximize the joint Q-value