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

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

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 $\boldsymbol{a}_t$:

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

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

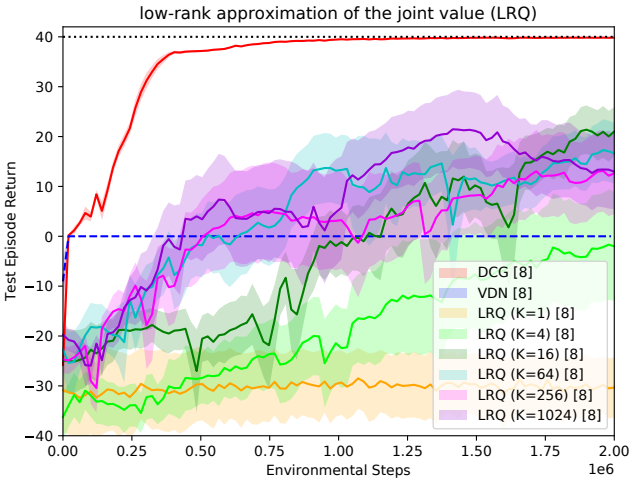

Figure 6. Low-rank approximation of the joint value function (LRQ) in the relative overgeneralization task of Section 5.1 for varying numbers of factors $K \in \{1, 4, 16, 64, 256, 1024\}$.

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, $\forall i \in \{1, \dots, n\}$,

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

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:

$$\boldsymbol{h}^i_0 \;:=\; \boldsymbol{0}\,, \qquad \boldsymbol{h}^i_t \;:=\; h_\psi(\cdot|\boldsymbol{h}^i_{t-1}, o^i_t, a^i_{t-1}) \tag{14}$$

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

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 relative 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 pseudo-code 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

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

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 $K = 64$.

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 17, where both DCG variants use a rank-1 payoff approximation. Note that 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.

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	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).

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*
    $\boldsymbol{f}^{\mathrm{E}}_e \leftarrow \frac{1}{2} f^e_\phi(\cdot, \cdot | \boldsymbol{h}^i_t, \boldsymbol{h}^j_t) + \frac{1}{2} f^e_\phi(\cdot, \cdot | \boldsymbol{h}^j_t, \boldsymbol{h}^i_t)^\top \quad \in \quad \mathbb{R}^{A \times A}$ // *permutation invariant payoffs*
  else // *if low-rank approximation*
    $[\hat{\mathbf{F}}, \bar{\mathbf{F}}] := f^e_\phi(\cdot, \cdot, \cdot | \boldsymbol{h}^i_t, \boldsymbol{h}^j_t) \quad \in \quad \mathbb{R}^{2 \times A \times K}$ // *compute payoffs "forwards" ...*
    $[\hat{\mathbf{F}}', \bar{\mathbf{F}}'] := f^e_\phi(\cdot, \cdot, \cdot | \boldsymbol{h}^j_t, \boldsymbol{h}^i_t) \quad \in \quad \mathbb{R}^{2 \times A \times K}$ // *... and "backwards" ...*
    $\boldsymbol{f}^{\mathrm{E}}_e \leftarrow \frac{1}{2} \hat{\mathbf{F}} \bar{\mathbf{F}}^\top + \frac{1}{2} \bar{\mathbf{F}}' \hat{\mathbf{F}}'^\top \quad \in \quad \mathbb{R}^{A \times A}$ // *... for permutation invariance*
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 $\boldsymbol{f}^V$ and payoff tensor $\boldsymbol{f}^E$*

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*
  $q' \leftarrow$ 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*

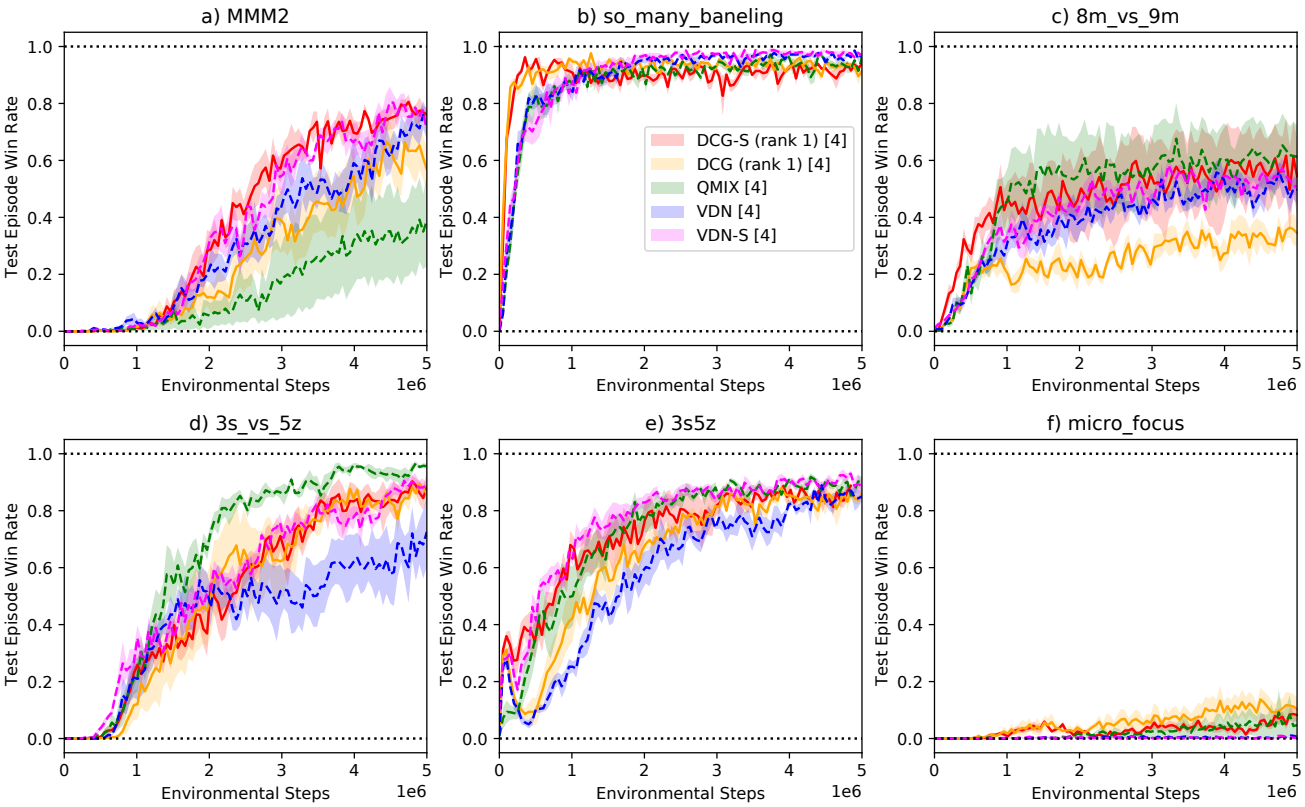

Figure 8. Cumulative reward for test episodes on SMAC maps (mean and shaded standard error, [number of seeds]) for `QMIX`, `VDN`, `VDN-S` and fully connected DCG with rank $K = 1$ payoff approximation (`DCG (rank 1)`) and additional state-dependent bias function (`DCG-S (rank 1)`).