ACE 补充材料

非稳态问题

在多智能体环境中，若全局状态为 $\mathbf{s}$ ，单个智能体 i 的局部观察为 o_i。由于智能体之间相互影响，因此智能体 i 在观察 o_i 下执行动作 u_i 后得的 r_i 与 o_i' 是由所有智能体的行为造成的，即 $r_i = R_i(\mathbf{s}, \mathbf{u})$ ， $o_i' = T_i(\mathbf{s}, \mathbf{u})$。所以对于单个智能体 i 而言，即使在观察 o_i 下一直都执行动作 u_i ，但是由于 $\mathbf{s}$ 未知且其他智能体的策略在不断变化，此时智能体 i 得到的 r_i 与 o_i' 可能是不同的（比如之前的 (o_i, u_i) 得到的奖励为 1，但是下一次由于队友的动作发生了变化，奖励又变成了 -1）。这就是 MARL 中的不稳定性，即 reward 与 transition 存在不稳定性，破坏了强化学习算法遵循的马尔可夫假设，从而导致智能体 i 的值函数 $Q_i(o_i, u_i)$ 的更新十分不稳定。

ACE （Cooperative Multi-agent Q-learning with Bidirectional Action-Dependency） 提出了双向动作依赖 Q 学习算法来解决上述问题，其核心思路是：不再要求多个智能体同时产生动作，而是把多智能体协作问题转化为一种稳定又高效的 **sequentially expanded MDP （SE-MDP）**，建模智能体之间的双向动作依赖，抽象出最精简的协作表征，让每个智能体一个接一个地产生决策行为，最终将非平稳的多智能体决策问题转化为特殊的平稳单智能体决策问题。

ACE 方法介绍

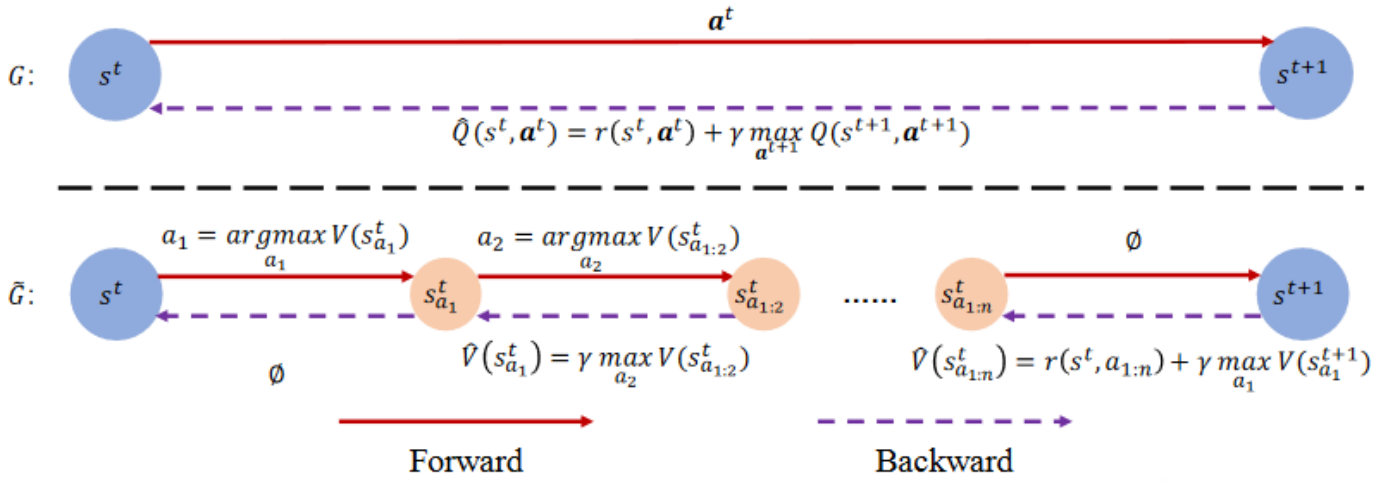
双向动作依赖

如图 1 所示，ACE 通过引入中间状态，将原始的多智能体 MDP $\mathcal{G}$ 转化成一个单智能体 MDP $\widetilde{\mathcal{G}}$ 。

- intermediate state $s_{a_{1:i}}^t$ 表示在状态 s_t 下第 1 到第 i 个 agent 已经做出了 action a_i 到达的状态；
- 智能体 i 接收 $s_{a_{1:i-1}}^t$ 做出动作 a_i ，下一刻的中间状态变为 $s_{a_{1:i}}^t$ ，得到奖励 0；
- 最后一个智能体采取动作后，中间状态变为 $s_{a_{1:n}}^t$ ，由一个伪智能体产生一个空 action，状态转移到 s^{t+1} ，收到环境奖励为 $r\left(s^t, \boldsymbol{a}^t\right)$ 。

所以原本的多智能体 MDP $G\left(s^t, \boldsymbol{a}^t, r\left(s^t, \boldsymbol{a}^t\right), s^{t+1}\right)$ ，被扩展为序列决策下的单智能体 MDP，即 sequentially expanded MDP （SE-MDP）

$\widetilde{\mathcal{G}}(\left(s^t, a_1^t, 0, s_{a_1}^t\right), \left(s_{a_1}^t, a_2^t, 0, s_{a_{1:2}}^t\right), ..., (s_{a_{1:n-1}}^t, a_n, r(s^t, \boldsymbol{a}^t), s^{t+1})$



（图 1：原始 MMDP（黑色虚线上方）和转换后的 SE-MDP（黑色虚线下方）之间的比较。MMDP 中的单个序列被扩展到 SE-MDP 中的 n 个顺序扩展状态。）

双向动作依赖的 Q-Learning （ACtion-dEpendent Q-learning （ACE））

ACE 可以将现有的单智能体的 actor-critic 或者 value-based 的方法迁移到多智能体场景，在下文中我们介绍 ACE 如何结合 DQN，如果不加特殊说明，ACE 即 ACE-DQN。

Decision with Rollout

之前基于值的方法通常学习 $Q(s,a)$ ，选择 max Q。在 SE-MDP 中，如果智能体 $i+1$ 在状态 $s^t_{a_{1:i}}$ 执行动作 a_{i+1} ，状态会直接转移到 $s^t_{a_{1:i+1}}$ ，因此我们可以通过计算所有可能的下一步 SE-state 的 value，选择

$$a^t_{i+1} = \arg\max_{a_{i+1}} V\left(s^t_{a_{1:i}, a_{i+1}}\right)$$

Update with Rollout

我们通过单智能体的标准贝尔曼方程更新 value function V 。在 SE 状态为 $s^t_{a_{1:i}}$ 时 rollout 所有可能的 a_{i+1} 得到所有可能的 $V\left(s^t_{a_{1:i+1}}\right)$ ，并选择最大值作为 target value；对于决策序列的最后一个 SE 状态 $s^t_{a_{1:n}}$ ，则需要 rollout 到下一个决策序列的第一个 SE 状态 $s^{t+1}_{a_1}$ ， $V\left(s^t_{a_{1:i}}\right)$ 的贝尔曼目标 $\hat{V}\left(s^t_{a_{1:i}}\right)$ 如下：

$$\hat{V}\left(s^t_{a_{1:i}}\right) = \begin{cases} \max_{a_{i+1}} \ \gamma V\left(s^t_{a_{1:i}, a_{i+1}}\right), & \text{if } i < n \\ \max_{a_1} \ r\left(s^t, a_{1:n}\right) + \gamma V\left(s^{t+1}_{a_1}\right), & \text{if } i = n \end{cases}$$

网络表示

由于状态如何表征对算法的效率和性能影响较大，所以本节我们重点讨论 $s^t_{a_{1:i}}$ 在 DNN 中的网络表示。

- **Decomposed State Embedding**

MMDP 的一个 transition $\left(s^t, \boldsymbol{a}, r(s^t, \boldsymbol{a}), s^{t+1}\right)$ 对应序列展开后的 n 个中间 transition，并且每个中间 transition 都需要评估 $|A_i|$ 个下一状态，也就是说总共有 $\sum_{i=1}^{n} |A_i|$ 需要评估。直接计算每一个

embedding $e_s(s^t_{a_{1:i}})$ 计算量会非常大，所以 ACE 将 state embeddings $e_s(s^t_{a_{1:i}})$ 分解成两部分 shared embedding $e_s\left(s^t\right)$ 和共享的 action embedding $e_a\left(a_1\right), ..., e_a\left(a_n\right)$。action embedding 共享一个 encoder。通过组合初始状态的 embedding $e_s\left(s^t\right)$ 和相应的 action embedding $e_a\left(a_1\right), ..., e_a\left(a_i\right)$ 得到最终的 state embedding $e_s(s^t_{a_{1:i}})$。这样状态只需要进行一次编码。

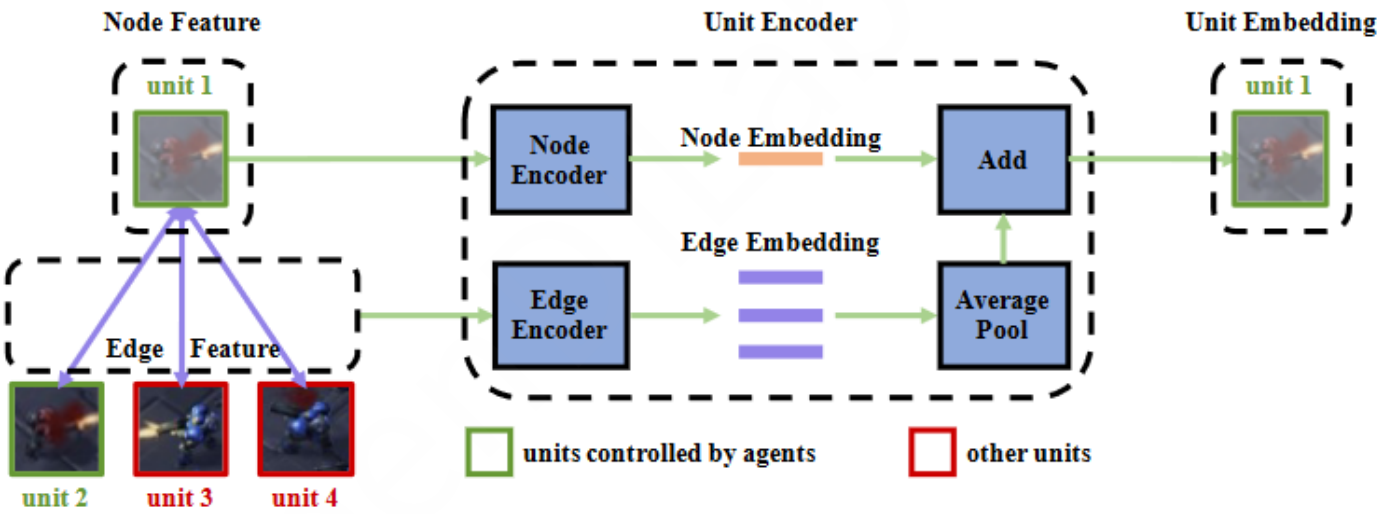
为了更好地建模不同智能体之间的交互，包括在 SMAC 里面运输船的治疗操作，枪兵和”光头“的攻击行为，ACE 引入了以下两种 embedding。

- **Unit-wise State Embedding**

ACE 用 unit encoder 来生成每个 unit 对应的 embedding $e_s\left(s^t\right) = [e_u\left(u_1\right), ..., e_u\left(u_m\right)]$，其中 n 个 unit 由智能体控制， m-n 个由环境控制。每个 unit encoder e_u 简单地使用全连接层来实现，输入包括 node feature 和 edge feature。

  - node feature 是每个单元的特征，比如 SMAC 中枪兵的血量和护盾，GRF 中球员的速度；
  - edge feature 编码单元之间的关系，比如不同单元之间的距离。

将得到的 edge embedding 进行平均池化，然后和 node embedding 相加，以获得最终的 unit embedding。整个过程如图 2 所示。



（图 2：unit encoder 的示意图。node embedding 由 node encoder 生成，edge embedding（针对单元及其交互单元）由 edge encoder 获得。将 edge embedding 平均池化后加上 node embedding 以得到 unit embedding。）

- **Interaction-aware Action Embedding**

向 $s^t_{a_{1:i-1}}$ 中加入动作 a_i，就变为 $s^t_{a_{1:i}}$，因此 $s^t_{a_{1:i-1}}$ 需要能够感知单元之间的交互，ACE 引入两种 interaction-aware action embedding：

  - active embedding $e^a_a(a_i)$： 编码动作 a_i 对于智能体 u_i 的作用
  - passive embedding $e^p_a(a_i)$：编码动作 a_i 对于目标智能体 u_j 的作用

即 $e_a(a_i) = [e^a_a(a_i), e^p_a(a_i)]$。

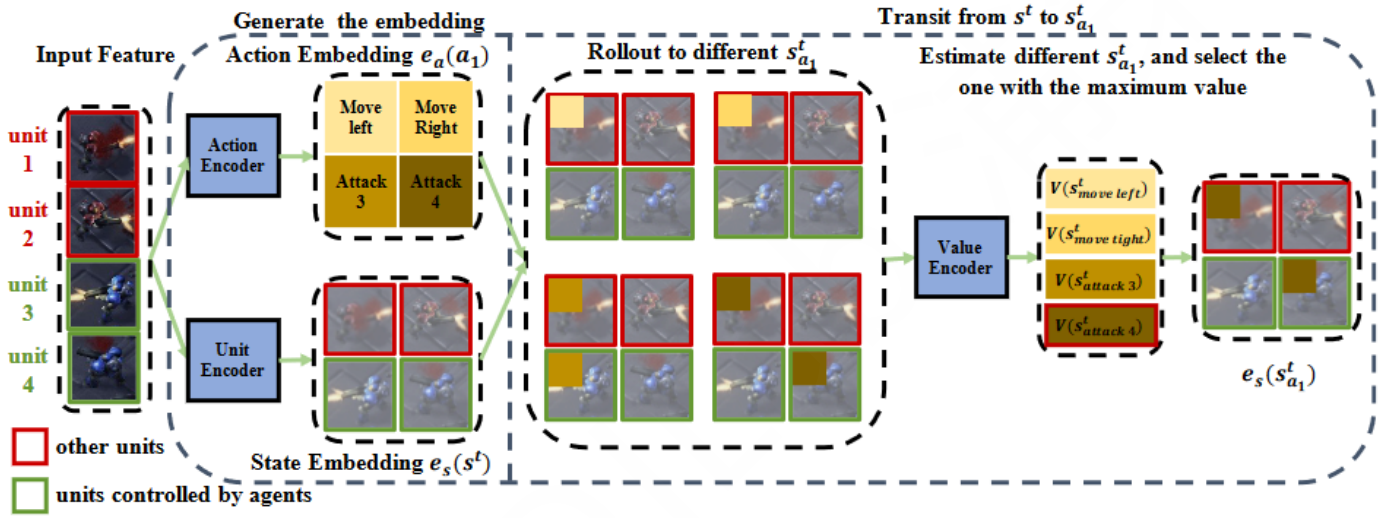
- 组合 embedding

state embedding 最终表示为 $e_s(s^t_{a_{1:i}}) = [e_u(u_{1,a_{1:i}}), ..., e_u(u_{m,a_{1:i}})]$

其中 $e_u(u_{j,a_{1:i}})$ 表示 unit j 的 embedding $e_u(u_j)$ 和相关的动作 embedding 组合。

$$e_u(u_{j,a_{1:i}}) = \begin{cases} e_u(u_j) + \sum_{e^p_a(a_k)\in P(a_{1:i})_j} e^p_a(a_k), & \text{if } i < j \\ e_u(u_j) + e^a_a(a_j) + \sum_{e^p_a(a_k)\in P(a_{1:i})_j} e^p_a(a_k), & \text{if } i >= j \end{cases}$$

当 $i < j$ 时，说明智能体 j 还没有做过动作，所以仅对 passive embedding $e^p_a(a_k)$ 求和。而当 $i \geq j$ 时，说明智能体 j 已经做过动作，需要加上 active embedding $e^a_a(a_j)$。



（图 3：以 SMAC 为例，ACE 的流程示意图。地图上有四个 unit。unit 1 和 2 由我方智能体控制，unit 3 和 4 是由环境控制的敌人。首先，生成初始 state embedding，包括从 unit encoder 获得的所有 unit 的初始 embedding，以及从 action encoder 获得的全部动作的 action embedding（图中仅显示了 unit 1 的 action embedding，其中动作攻击 3 和攻击 4 分别表示 unit 1 攻击 unit 3 和 4）。然后，智能体 unit 1 是第一个做出决定的，因此其 action embedding 被合并到初始 unit embedding 中，以 rollout 到不同的新 SE-state $e(s^t_{a_1})$ 的 embedding（图中 4 个 rollout 得到的 SE-state）。然后，所有这些新的 SE-state 都由 Value Encoder 进行评估。最后，具有最大值的 SE-state 被保留，并由接下来智能体 2 的 rollout 使用。）

实验结果

实验设置

- 任务
  - Spiders-and-Fly 更困难版本
  - StarCraft Multi-Agent Challenge
  - Google Research Football
- 评价标准
  - Spiders-and-Fly： 预先设定 oracle 策略，评判标准如下
    - 在十个 step 实现 100%成功率所需的样本量
    - RL 策略和 oracle 策略捕获苍蝇所需的平均步骤之间的差距

  ACE 使用多组随机 seed ，每个子环境分别执行 32 个 episode ，测试以下两个环境：
  - SMAC 环境：中位数胜率
  - GRF 环境：平均胜率

表现

Spiders-and-Fly

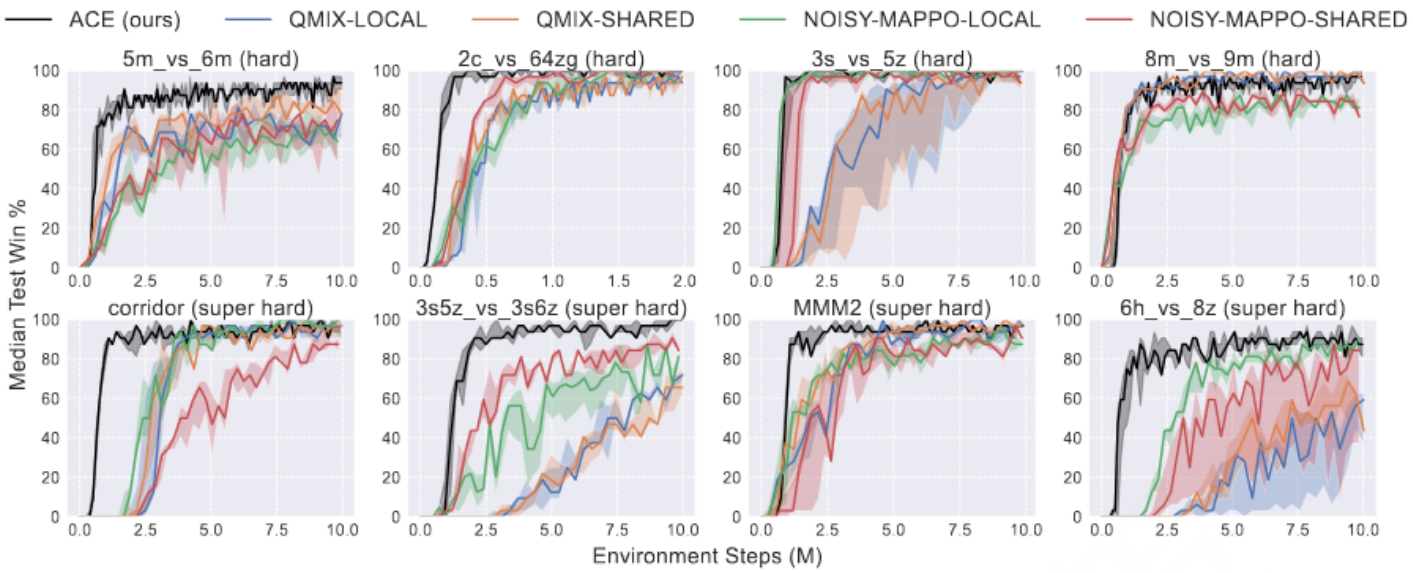
将 ACE 与三种值分解方法 QTRAN、QMIX 和 VDN 进行了比较，结果如表 1 所示：ACE 是唯一能够近似 oracle 策略性能的方法，而基线虽然在某些情况下也能找到最佳行为，但无法始终收敛到最佳策略。此外，ACE 只需减少 50%的样本，即可在十个步骤中实现 100%的成功率。

Metric	Map	VDN	QMIX	QTRAN	ACE
Steps	5×5	0.78±0.10	0.77±0.10	0.60±0.09	**0.04±0.03**
	7×7	0.90±0.12	0.87±0.11	1.02±0.09	**0.07±0.02**
Samples (M)	5×5	0.19±0.02	0.19±0.02	0.17±0.02	**0.09±0.01**
	7×7	1.97±0.10	1.81±0.09	1.68±0.09	**1.01±0.06**

（表 1：ACE 与 Spiders-and-Fly 的基线进行比较。Steps 表示方法的平均 step 与 oracle 策略之间的差距。Samples 代表在 10 个 step 内实现 100%成功率所需的的样本数量。）

SMAC

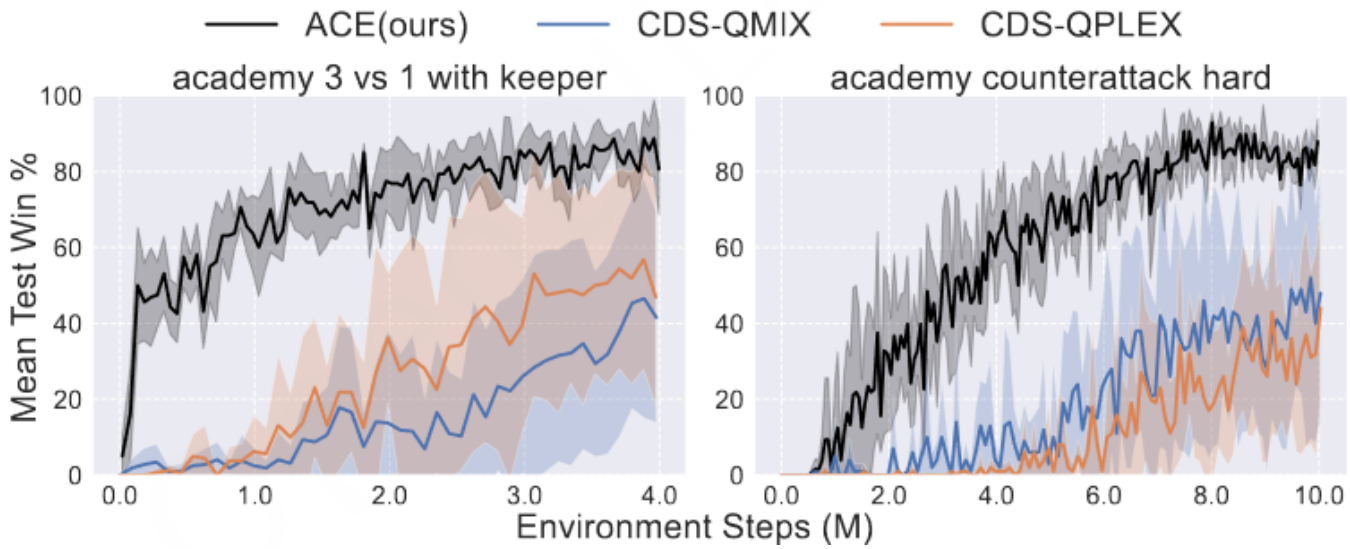
从图 4 可以看出，ACE 在最终的胜率和样本利用率上都显著超越目前的 SOTA 方法（包括 fine-tuned QMIX 和 NOISY-MAPPO）。其次，ACE 在绝大多数所有地图上达到了 100%的胜率，尤其在 5m vs 6m 和 3s5z vs 3s6z 这两张之前方法明显表现较弱的地图上，ACE 依然展现出了强大的效果。



（图 4：在四个 hard 难度和 super hard 难度的 SMAC 地图中 ACE 和基线的对比）

GRF

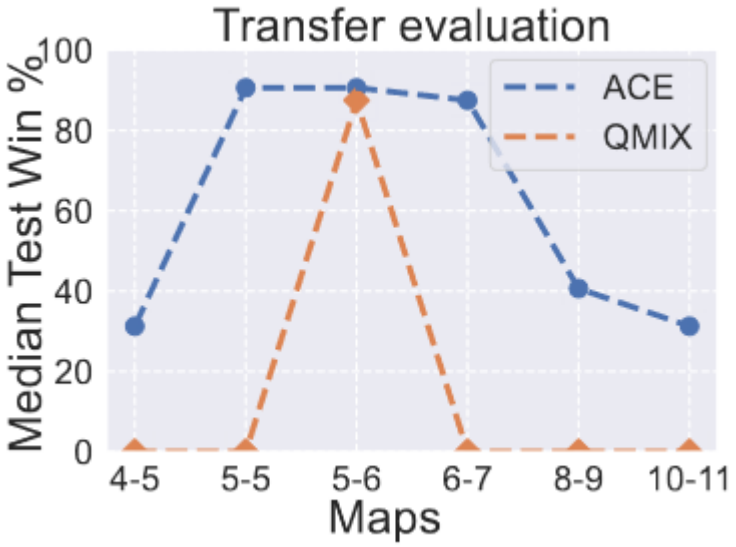
如图 5 所示，在 GRF 环境中，ACE 优于 SOTA 方法 CDS-QMIX 和 CDS-QPLEX。值得注意的是，ACE 和当前 SOTA 方法之间的差距甚至比 SMAC 更大，这主要是由于足球游戏需要更复杂多样的合作技能。



（图 5：在 GRF 环境中 ACE 和基线的比较）

拓展实验：ACE 的泛化能力

ACE 具有良好的泛化能力。与之前研究所采用的单个智能体单独做出决定的方法相比，ACE 显式地建模了代理之间的合作关系。所以当前序的智能体由于任务的变化而做出次优动作决策时，后序的智能体会根据学习到的合作能力来进行弥补。论文中进行了泛化实验，在 5m_vs_6m 上训练 ACE 和 fine-tuned QMIX-SHARED [1]，并在 4m_vs_5m、5m_vs_5m，6m_vs_7m、8m_vs_9m 和 10m_vs_11m 上测试它们。如图 6 所示，尽管在测试地图上没有任何微调， ACE 仍然取得了较高的胜率，这表明它对智能体数量的变化具有很好的泛化能力。



（图 6：从 5m_vs_6m 迁移到不同智能体数量的地图。’ x-y’ 表示 xm_vs_ym。无论代理数是增加还是减少，ACE 都可以在 zero-shot 设置中实现卓越的性能。）

参考文献

[1] Hu J, Jiang S, Harding S A, et al. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning[J]. arXiv preprint arXiv:2102.03479, 2021.

[2] Li C, Liu J, Zhang Y, et al. ACE: Cooperative Multi-agent Q-learning with Bidirectional Action-Dependency[J]. arXiv preprint arXiv:2211.16068, 2022.