A Technical Appendix

A.1 Algorithm Pseudocode

The pseudocode of MASL is summarized in Algorithm 1.

Algorithm 1 Multi-agent Selective Learning (MASL)

Initial parameter θ, ω, θ', ω', a normal replay buffer $\mathcal{D}$.
Initialize parameter of the backtracking model ϕ, a high-value replay buffer $\tilde{\mathcal{B}}$.
for episode e= 1 to E **do**
  Initialize state, $t = 1$
  while $t \leq T$ **do**
    Execute actions $\boldsymbol{a} = (a_1, \cdots, a_N)$, $a_i = \mu_{\theta_i}(o_i')$.
    Receive reward r and next observation $\boldsymbol{o}'$.
    Store $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{o}')$ in replay buffer $\mathcal{D}$.
    Update observations: $\boldsymbol{o} \leftarrow \boldsymbol{o}'$
    if $r \geq \zeta$ **then**
      $h \leftarrow$Algorithm 2$(\boldsymbol{o})$
      Update time step $t = t + h$
    end if
    $t = t + 1$
  end while
  for agent i=1 to N **do**
    Sample mini-batch transition $(\boldsymbol{o}^k, \boldsymbol{a}^k, \boldsymbol{r}^k, \boldsymbol{o}'^k) \sim \mathcal{D}$.
    Pass into the selector to get $(\hat{\boldsymbol{o}}^k, \hat{\boldsymbol{a}}^k, \boldsymbol{r}^k, \hat{\boldsymbol{o}}'^k)$.
    Update critic: $\omega_i \leftarrow \omega_i - \eta \nabla_{\omega_i} \mathcal{L}(\omega_i)$
    Update actor: $\theta_i \leftarrow \theta_i - \eta \nabla_{\theta_i} J(\theta_i)$
    Update Backtracking model:
    $\phi_i \leftarrow \phi_i - \eta \nabla_{\phi_i} \mathcal{L}(\phi_i)$
    Generate recall traces $\tilde{\tau}$ from high-value states sampled from $\tilde{\mathcal{B}}$
    # ***Update actor using recall traces***
    $\theta_i \leftarrow \theta_i - \eta \nabla_{\theta_i} \mathcal{L}(\theta_i)$
  end for
  Update the target networks
end for

The interaction process of selecting high-value trajectories based on retrogression is summarized as Algorithm 2.

Algorithm 2 Retrogression-based Trajectory Selection

Input: Current observation of all agents $\boldsymbol{o}$, a high-value replay buffer $\tilde{\mathcal{B}}$, horizon T;
Output: Time step t
Initialize $t = 1$
repeat
  Each agent i executes an action a_i according to a rule-based strategy $a_i = \pi_r(o_i)$.
  Receive reward r and next observation $\boldsymbol{o}'$.
  Identify high-value trajectories.
  $h = h + 1, \boldsymbol{o} \leftarrow \boldsymbol{o}'$.
until ($r \leq 0$ or $t \geq T$)
Update ζ and store all high-value trajectories in the replay buffer$\tilde{\mathcal{B}}$.

A.2 Training Detials

For all environments, we adapt their opensource implementation from the original multi-agent particle environment in MADDPG. The neural network architecture of the decentralized actor in all the algorithms we implemented is two multi-layer perceptrons (MLP) with 128 hidden units. The critic first extracts features of the agent's own information and the other agents' information separately, and then integrates them to output a Q-value. To emphasize the agent's own information, we process it by two MLP with 128 and 96 hidden units, and the other agents' information is processed by a single-layer perceptron with 96 units. The hidden states are integrated and further processed with two MLP with 64 hidden units. We use critic architecture similar to Message-dropout MADDPG, but without inputting the action into a hidden layer. The dropout technique is applied when processes the other agents' observations and actions. The critics of MADDPG and DDPG use the same architecture of two MLP with 128 hidden units. The backtracking model consists of two parts: a state generator and an action generator. Both generators use two MLP with 128 hidden units and ReLU activation function.

We update the actor and the critic for every 50-time steps. For every 200-time steps, we sample a batch of top 1000 transitions and do a 50 training-steps of the backtracking model. Backtracking model generates trajectories of 3 steps for training. The selectors used in MASL select information from the other K agents. The parameter K is set according to different scenarios, as shown in Table 1. All the algorithms are trained with 6 random seeds for comparison. The code is available in the anonymous Github https://anonymous.4open.science/r/MASL-88A5.

Table 1. Setting of parameter K in different scenarios

#Scenario	RC-5	RC-8	RC-10	RE-6	RE-8	RE-12
#K	2	3	4	4	5	6

A.3 Environment Setting

A.3.1 Resource Collection Task

The resource collection task requires N agents to collect resources in L resource pools. Each agent observes its position and velocity, the positions of the other agents and the resource pools relative to its own. Agents are generated at random locations initially. To collect resources, each agent needs to stay near a resource pool until the episode end. Each agent gets a positive reward $r^+ = c^2$ where c is the number of mining resource pools, and gets a negative reward $r^- = 0.1$ when a collision happens. Agents shares a global reward at each time step, equal to the sum of each agent's individual reward, i.e, the global reward for agents at time step t is $r = \sum_{i=1}^{N} c^2 - 0.1 * d_i$, where d_i is the collision number of agent i.

To study the applicability of MASL, we consider the settings of individual reward and partially shared reward, respectively. For the individual reward setting, each agent i gets a reward $r_i = 5c_i - 0.1 * d_t^i$, where c_i is the number of resource pools being mined by agent i. In general, $c_i \in [0, 1]$, i.e., an agent can mined only one resource pool at a time. For the partially shared reward setting, the situation is equivalent to removing the sum from the global reward setting. Each agent i gets a reward $r_i = c_t^2 - 0.1 * d_t^i$.

A.3.2 Rover Exploration Task

The rover exploration task requires N agents to coordinately explore L target areas in groups. For a coupling requirement of 2, i.e., $p = 2$, we set $N = P * L = 2L$, then the goal of agents is to form a two-person team to explore the areas. Each agent observes its position and velocity, as well as the position of all target areas relative to itself. Agents receive a shared reward $r = \sum_{j=1}^{L} 5 \cdot C_j^2$, where C_j is the exploration index of the target area j. C_j equals to the number of agents c that are exploring target area j and is clipped when c is larger than the coupling requirement, i.e., $C_j = clip(c, (0, p))$.

A.4 Addition Results of Computational Complexity

Since different algorithms cost basically the same running time to collect a sample at each step, the number of training episodes required to achieve specified learning effect (i.e., sample efficiency) is usually used to evaluate the efficiency of algorithms. However, the actual running time varies with the settings of parameters and practical computing equipment used, so it is fairer to further discuss the actual running time while comparing sample efficiency.

Table 2. Measured mean process time (minutes: seconds) required for 10000 timesteps.

	MASL	MADDPG-MD
N=5	1:25	1:24
N=8	3:14	3:00
N=10	5:08	4:40

We discuss the computational complexity of MASL by comparing it with MADDPG-MD algorithm, which proved to be efficient in centralized training. Table .2 contains the process time required for running MASL and MADDPG-MD in resource collection environments. Timings were measured on Intel i7-9700K @ 3.60GHz and Nvidia GeForce RTX 2080 Ti running Python 3.6 and Tensorflow-Gpu 1.13. The average time for running and training on 10000 environment iterations is displayed. It is shown that MASL did not introduce excessive training time compared with MADDPG-MD.