# MARL (CURVE) @ IDM Lab

Student: Dongming Shen
Supervisor: Taoan Huang
PI: Sven Koenig, (Bistra Dilkina)

# GitHub

GitHub Code: https://github.com/AlvinShen99/marl_SK

Original Package: https://github.com/oxwhirl/pymarl

Possible Extensions:

https://github.com/Replicable-MARL/MARLlib

https://github.com/uoe-agents/epymarl

https://github.com/semitable/robotic-warehouse

https://github.com/koulanurag/ma-gym

# Problem setup

Classical MAPF settings

But with a focus on: small number of agents on small maps

1. Can replace solution database heuristic in some (lifelong) MAPF solver
2. Can replace repair heuristic in LNS

# MARLEnv Config - Maps & Agents

**Maps Config:**

https://movingai.com/benchmarks/mapf/index.html - .map files

- Map config is initialized only once when Env created and kept fixed.

**Agent Config (start locations, goals):**

https://movingai.com/benchmarks/mapf/index.html - .scen files

- Every time Env.init() / Env.reset() called, randomly sample N agents' config from a random .scen config file.

# MARLEnv Config - Step/Transition Methodology

**Collision Handling:**

- Env Collision: not allowed (by limiting available actions)
- Agent Node Collision: allowed with punishment & observation
- Agent Edge Collision: allowed with punishment & observation
- NOTE: available actions is ONLY constraint by the environment, not agents

**Implementations Ideas:**

1. In .step(), keep a separate map M’ storing temporary agent info in transition.
2. Post-process M’ to count Node collisions = # agents at same positions.
3. Post-process M’ & M to count Edge collisions = # agents swapped positions.
4. Apply punishments on collisions.

# MARLEnv Config - Observation (Full) - Depreciated

**Observations: (fully-observable). Problem: cannot generalize**

- 1. Full map, each location:
  - 0 (empty), -1 (obstacle), >0 (number of agent)
- 2. All agents' features:
  - location (x,y), start, goal, timestep, unit vector direction to goal, distance to goal
- 3. Collision agents' index (2 type of collisions)
- Finally, flatten everything into a single 1D array.

Note: since fully-observable, all agents have same observations

Reference: https://arxiv.org/pdf/1809.03531.pdf (PRIMAL)

# MARLEnv Config - Observation (Partial)

**Observations for each agent: (partially-observable)**

- **1. Partially observable window (WxW), for each location:**
  - 0 (empty), -1 (obstacle or out-of-map), >0 (number of agent at this location)
- **2. K(including itself)-Nearest-Neighbor agents' features, for each agent:**
  - Locations: (2) current location, (2) start location, (2) goal location
  - Goal Vector: (2) unit vector direction, (1) L1/L2/path-distance to goal location
  - Collisions: (1) if node collision, (1) if edge collision
  - Other: (1) L1/L2/path-distance from the observing agent, (1) timestep until start
- **Finally, flatten everything into a single 1D array as observation.**

Reference: https://arxiv.org/pdf/1809.03531.pdf (PRIMAL)

# MARLEnv Config - Reward Structure

**Rewards: m(0.01), nc(1), ec(1), s(0.02), s’ (0), f(10) all positive, s>m**

- Move (L, R, U, D): -m
- Node Collision, Edge Collision: -nc, -ec
- Stay (at goal): -s’ => should consider target conflict as an extension
- Stay (not at goal): -s
- Finish Episode: f (a single reward when all agents finish tasks)

The environment generates have a **single reward** by summing agents’ rewards.

Reference: https://arxiv.org/pdf/1809.03531.pdf (PRIMAL)

# MARLEnv Config - Hyper Parameters

**Hyper Parameters in the Env:**

- Rewards: m, s, s’, nc, sc, f
- Partially observable window size: WxW
- KNN observable agents: K
- Each observable agent’s features: …

# Problems - fixed in Partial Observation

- Each network only works for a FIXED number of agents (observation space's size depends on the number of agents), week generalization
- Network Architecture (MAC => each agent rnn & vdn sum)
- Replay Buffer size too large (8*8 map 5 agents takes ~ 20G. Buffersize=5000, episodelen=200) for RAM (32G)
- How to run/test the saved models

# Extension Ideas

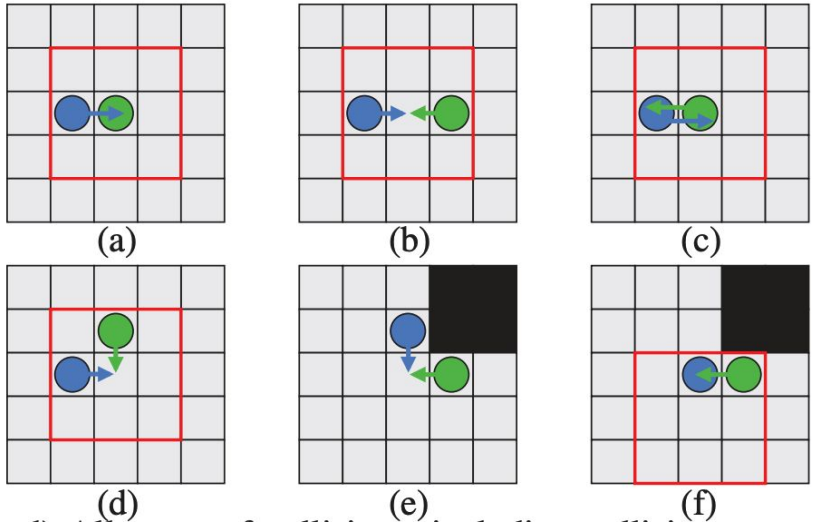
- CNN to pre-process windows observation (change network structures)
- GCN = Graph Convolutional Network
- DQN idea: only use most recent N frames
- Use FastMap coordinate to represent locations
- Target Conflict in reward
- How to define “Done” for each agent

# Application 1

Replace solution database heuristics in MAPF solvers (https://ieeexplore.ieee.org/ielaam/7083369/8932682/8962218-aam.pdf)

Previous work precompute all optimal solutions in all possible config on ***empty*** 3x3 and 3x2 map

Made assumption on the map that make sure they could always find empty 3x3 and 3x2 areas




Toy example for RL: 3x3 empty areas

# Application 2

Repairing solutions in LNS for **a small set of agents** (in the neighborhood)

1. Deconflicting the paths in LNS
2. If already conflict free, find an improved set of paths

Need to deal with moving obstacles (other agents' fixed paths)

# Related work: PRIMAL

Train a single-agent policy with RL and IL

They use a centralized planning algorithm as the expert policy in IL

Not explicitly trained to achieve the same goal (i.e. maximizing the reward)

Main differences:

1. We focus on relatively small agent sizes
2. Use VDN, an MARL algorithm, to maximize the sum of reward

# Related Work

https://arxiv.org/pdf/2211.02127.pdf

# Reference


- VDN: https://arxiv.org/abs/1706.05296
- QMIX: https://arxiv.org/abs/1803.11485
- PRIMAL: https://github.com/gsartoretti/PRIMAL
- PyMARL: https://github.com/oxwhirl/pymarl

# Dec - Jan TODO:

- Success rate? Simulation?
- Test cases (8x8 empty)
- https://github.com/Jiaoyang-Li/CBSH2
- https://github.com/Jiaoyang-Li/CBSH2-RTC
- https://github.com/Jiaoyang-Li/MAPF-LNS

# Version 1.1:

- Rewards: move (-.01), stay (-.02), stay_goal (0), collide (-1), complete (30).
- Result:
  - Test_return_mean: -200 to around -12 after 9000000 iteration
  - Ep_len_mean: always 100 (never really complete)
- Problem assumption: reward sparsity – not enough incentive for moving toward the goal. Note that episode completion requires all agents staying at goals at the same time, which is not likely to happen by chance.
- Possible Solution: add extra reward as incentives for moving to the goal.

# Version 1.2:

- Rewards: move (-.01), stay (-.02), stay_goal (0), collide (-1), complete (30).
- Extra Rewards: extra [-.01, .01] reward when moving closer/away to goal (determined by shortest path distance difference between steps)
- Result:
  - Test_return_mean: -200 to around -15 after 2000000 iteration
  - Ep_len_mean: always 100 (never really complete)
- Problem assumption: still no incentive to STAY at goal until early completion criterion met.
- Possible solution: increase stay_goal and complete reward, decrease collide reward to avoid the agent pick “not moving” policy.

# Version 1.3:

- Rewards: move (-.01), stay (-.02), stay_goal (1), collide (-.05), complete (300), [-.01, .01] reward when moving closer/away to goal.
- Result:
  - Test_return_mean: -50 to around 820 after 3200000 iterations
  - Ep_len_mean: 100 to around 70 (low rew) then back to around 97 (high rew)
- Problem assumption: most agent stay at goal to get more 1 reward until near 100 all agent reach goal to get final 300.
- Possible solution: modify goal completion reward from a fixed number to considering early completion extra incentive.

# Version 1.N Results:

- Big stay & complete reward, Small collision reward:
  - Agents can reach goals and finish the episode fast, but will collide.
- Normal collision reward:
  - Agents will eventually converge to the “not moving” policy
- Increase complete reward if finish early
  - No change
- Other small modifications
  - No change

# Version 1.N Reflection:

- We tried several reward settings and see some general patterns of results, but still did not find a result setting that can let the agents learn what we want it to learn.
- After consideration we think this is due to reward sparsity for the actual final reward that is important mixing with too many dense rewards that actually became a distraction for learning. One possible solution is to only include completion reward & collision reward, and the reward that aims to help the agent achieve the goal.

# Version 2 Important Modifications:

1. Start with small collision rewards (small punishment to let the agent learn how to reach goal), then gradually enlarge collision rewards (to let the agent learn how to avoid colliding with each other).
2. Completion rewards based on the time of completion (avoid some agents keep waiting at goal to get the continuing rewards without completing).
3. Remove move/stay reward altogether, aiming to remove the distraction from the dense rewards and let the policy focus on the sparse final rewards.

# Version 2 Results (temporary):

- At beginning of training with small collision punishment, the agents learn how to reach goals fast (average episode length of 10) but has lots of collisions. Note: now the incentive to finish goal earlier is to get a larger complete reward, instead of getting small & dense punishment for moving/staying at each single time step. The modification of replacing dense with final sparse helped the learning process.
- As we gradually increase the collision punishment, there are significantly less collisions (but not yet collision-free), however the episode length also increases (average episode length of around 10).

# Version 2 TODO:

1. Create a method to avoid collision at test time even when the policy outputs results that lead to collision.
2. Implement a visualizer to visualize the plan at test time.
3. During testing, compute and log some extra indicators (total cost, make span, total number of collisions, etc.)

# Update 5:

- Rewards: move (-.1), stay (-.2), stay_goal (1), collide (-20), complete (0). => aim to avoid collision, extra [-.1, .1] reward when moving
- Converged to “not move” policy with reward -200 (much smaller than collision) => collide punish still too large

# Update 6:

- Rewards: move (-.1), stay (-.2), stay_goal (1), collide (-5), complete (0). => aim to avoid collision, extra [-.1, .1] reward when moving
- Converged to “not move” policy with reward -200 (much smaller than collision) => collide punish still too large

# Update 7:

- Rewards: move (-.1), stay (-.4), stay_goal (1), collide (-2), complete (0). => aim to avoid collision, extra [-.1, .1] reward when moving
- Converged to “not move” policy with reward -200 (much smaller than collision) => collide punish still too large

# Note:

- 适当减少Stay_goal的reward
- Stay_goal
- Stay_goal不持续给，给一个大的，离开再扣除
- Complete reward调整
- Collision 改大一点

# Note Feb 23:

- Reduce dense rewards, only include completion reward & collision reward