POGEMA: Partially Observable Grid Environment for Multiple Agents

Alexey Skrynnik[1], **Anton Andreychuk**[1], **Konstantin Yakovlev**[1], **Aleksandr Panov** [1]

[1]AIRI, Moscow, Russia

Abstract

We introduce POGEMA[1] a sandbox for challenging partially observable multi-agent pathfinding (PO-MAPF) problems . This is a grid-based environment that was specifically designed to be a flexible, tunable and scalable benchmark. It can be tailored to a variety of PO-MAPF, which can serve as an excellent testing ground for planning and learning methods, and their combination, which will allow us to move towards filling the gap between AI planning and learning.

Introduction

Multi-agent pathfinding (MAPF) is a challenging problem with typical applications in video games, logistics, etc. The intrinsic assumption of what is called Classical MAPF (Stern et al. 2019) is that there exists a central controller which possesses all the information about the agents and the environment. It is this controller that plans a set of collision free paths for all of the agents. Thus, such variant of MAPF can be deemed to be *fully observable* and *centralized*. Indeed, plenty of methods exist to solve this variant of MAPF, see (Sharon et al. 2015; Čáp et al. 2015; Surynek 2009; Wagner and Choset 2011) for example.

In many practical applications, though, it is impossible to deploy a reliable infrastructure for a centralized MAPF. Consider, for example, a coverage/surveillance of areas such as nuclear plants, mines, disaster areas, etc. These areas can have poor communications (or none at all). Thus, they require a fundamentally different variant of MAPF, i.e. on the one when the central controller is absent and each agent has limited communication/observation capabilities – *partially observable* MAPF (PO-MAPF), which is intrinsically *decentralized*.

Indeed, PO-MAPF requires different methods compared to fully observable MAPF. In the former case, we seek not for a fixed solution, i.e. the set of conflict-free plans, but rather for a policy that maps agents' observations to actions in such a way that it maximizes the chance of reaching the goal while avoiding the collisions and minimizing the number of actions performed. To foster the development of such methods, which are likely to span across different areas of AI (Search, Planning, Learning etc.) a fast and flexible software environment is needed, which will allow researchers to quickly prototype and evaluate the methods for solving PO-MAPF problems. In this work we present such an environment – POGEMA (Partially Observable Grid Environment for Multiple Agents) (see an example on figure 1).

[1]Code available at https://github.com/AIRI-Institute/pogema

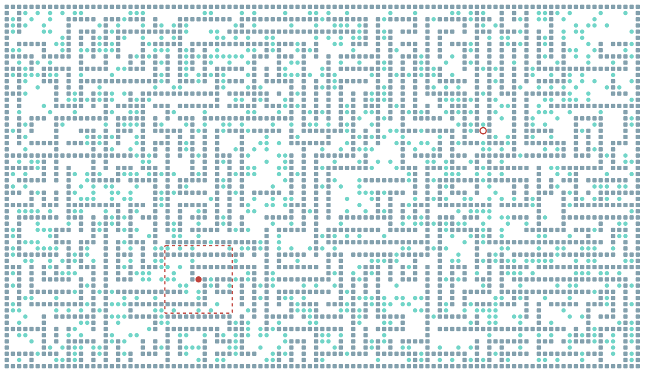
Figure 1: A typical PO-MAPF instance in POGEMA. The agent for whom the path is being constructed is indicated by a red circle, his goal is a white circle with a red border, his area of observation is a square with an intermittent red border. Static obstacles are shown in gray, other agents – in turquoise.

POGEMA relies on the widespread grid representation of the surrounding space and objects. As dictated by PO-MAPF, each agent in this environment has an access only to a local observation, i.e. ego-centric patch of the grid of the user-defined size. Relying on the (history of) observations an agent chooses an action to be performed at the next time step. POGEMA is equipped with several baseline policies for choosing such an action, including the search-based one and the learning-based one. Indeed, POGEMA allows plugging-in user-designed policies both for evaluation and for training (in case of the learnable policies).

POGEMA is specifically tailored to train policies based on Reinforcement Learning (RL) methods. It provides interfaces for well-known RL frameworks and its sample efficiency is up to 10k FPS (frame per second) even for single-agent cases (for multi-agent cases it is much faster). The latter allows fast training of the complex value and state approximators based on neural network models.

Some preliminary results show that some heuristic search-based planners are sufficient to solve some PO-MAPF tasks, and learnable approaches are better able to cope with other types of tasks. This shows that PO-MAPF setting can serve as an excellent testing ground for the development of hybrid methods of simultaneous planning and learning, which will allow us to move towards filling the gap between AI Planning and Reinforcement Learning. POGEMA is an excellent tool for testing the efficiency of new methods in this area and presents a fast and effective tool and a set of baselines with comprehensible metrics and a simple interface for creating your own new algorithms.

Related Works

There are a number of environments in the RL community that are used to train agents in a multi-agent setting (MARL). Such environments include Flatland (Mohanty et al. 2020) and Petting Zoo MAgent (Zheng et al. 2017). Flatland is designed to solve the specific problem of fast conflict-free train scheduling on a fixed railway map. This environment is quite slow and focused on full observability. MAgent is a fast environment for modelling the group and swarm behaviour of agents with a set of actions that, in addition to moving, includes actions for interacting with other agents. This environment has a limited set of scenarios and does not have an interface for testing planning solutions. A number of other visual-based environments known in MARL (Dota 2, Starcraft) seem to be more heavy and complex in terms of encoding observations and actions, which makes it impossible to give a quick and convenient interface for comparing trajectory planning methods. Table 1 provides a brief comparison of the main features of POGEMA and similar environments.

Table 1: Comparison of POGEMA with other multi-agent grid-based environments. FPS was benchmarked with 80 agents for each environment. For MAgent we used the *Battlefield* task.

Environment	FPS	Procedural generation	Requires generalization	Partial observability
POGEMA	83.000	✓	✓	✓
MAgent	184.000			✓
Flatland	156	✓	✓	

POGEMA Environment

Basics

Consider n agents which populate to the 4-connected grid composed of the free and blocked cells. This grid can be either procedurally generated by POGEMA or provided by the user. Each agent is assigned to the goal cell which it has to reach. At every time step an agent receives a local ego-centric observation, whose size R is defined by the user, and picks an action, which can be either move to one of the adjacent cells or waits in the current cell. The action picking algorithm, i.e. the *policy*, is defined by the user (several baselines are also provided). After each agent picks an action POGEMA applies all the actions that are feasible, i.e. do not lead to the collisions (either between the agent and the obstacle or between several agents). Agents that picked the infeasible action remain where they were. If an agent enters its goal cell it is removed from the environment (the so-called "disappear-at-target" behavior). The *episode* ends either if all the agents reach their goals or if the user-specified time step limit, K, is reached.

Observation space

In POGEMA, the agent occupying the cell with the coordinates (i, j) is able to observe the status of the cells $i \pm R, j \pm R$, where R is the user-defined observation radius. Thus, the observation is a patch of a grid of size $[2 \cdot R+1] \times [2 \cdot R+1]$ centred at the currently occupied cell. Any information regarding the other agents, except their current locations (e.g., their goals, paths (or path segments) to the goals, etc.), *is not included* in the observation. This is purposefully done to simulate the most challenging PO-MAPF setup, when the agents can not communicate with each other and share any information. We are planning to add other, less restrictive, observation designs in future.

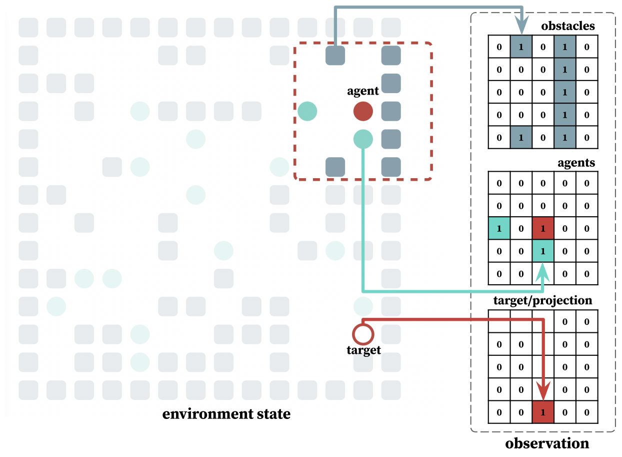

Figure 2: POGEMAobservation space for RL algorithms. Observation space consists of three matrices corresponding to obstacles, other agents and target or its projection.

Technically, the observation is encoded as three matrices. The first one bears the information about the static obstacles within the field of view. The second one contains the information about the other agents. The third one includes the goal projection (see Figure 2). The agent's target can be encoded using direction or relative coordinates, but we present it as a matrix to simplify input encoding of the neural network (it is easier to encode data of one modality).

State Space

Majority of MARL algorithms utilise environment state during training (centralized training, decentralized execution scheme). Pogema has a method to provide state of the environment. For PO-MAPF the full state consists of the global map and the positions/goals of all agents. In contrast to observation space, the state space can change depending on the number of agents and size of the grid. Thus, algorithms with centralized training are restricted to be trained in the exact PO-MAPF domain.

Reward Function and Metrics

The agent receives a reward of 1.0 when it reaches the goal and 0.0 in all other cases. We have chosen this function, since it is universal for the wide range of PO-MAPF configurations. Moreover, this reward is easy to interpret, as it corresponds to the individual success rate (ISR) metric for each agent. To shape the reward function one could use the wrappers mechanism of OpenAI Gym. The second metric is a cooperative success rate (CSR), which is equal to 1 only if all of the agents have reached the target.

Interfaces for RL Frameworks

OpenAI Gym (Brockman et al. 2016) is de facto standard interface for the agent to interact with the environment in RL. Unfortunately, Gym is designed for classic single-agent interaction, which restricts its application for PO-MAPF problems. *PettingZoo* (Terry et al. 2020) framework provides a unified interaction interface for multi-agent RL (MARL).

Pogema provides PettingZoo and single-agent Gym integration out of the box (see example Figure 3). Besides PettingZoo, MARL community has several other interfaces: vectorized Gym-like interaction, interaction used in PyMARL (Samvelyan et al. 2019) and Rllib (Liang et al. 2017) interface. In addition to these frameworks we provide integration with high-quality Asynchronous-PPO algorithm, implemented in *SampleFactory* (Petrenko et al. 2020).

```
import gym
import pogema

# Create Pogema environment
# with PettingZoo interface
env = gym.make("Pogema-8x8-hard-v0",
integration="PettingZoo")
...

# Create single-agent Pogema environment
# with OpenAI Gym interface
env = gym.make("Pogema-8x8-easy-v0",
integration="gym")
...
```

Figure 3: Sample code for creating Pogema environment with a parallel PettingZoo interface and single-agent Gym interface. Please note, *Pogema-8x8-easy-v0* corresponds to a grid with size 8×8 with only one agent.

Builtin Benchmarks

We provide builtin configurations of the maps for easier comparison of different algorithms. Pogema benchmark consists of three scales of grid configurations with four levels of difficulty (see example in Figure 4). We picked sizes: 8×8, 16×16, 32×32, 64×64. The obstacles are placed

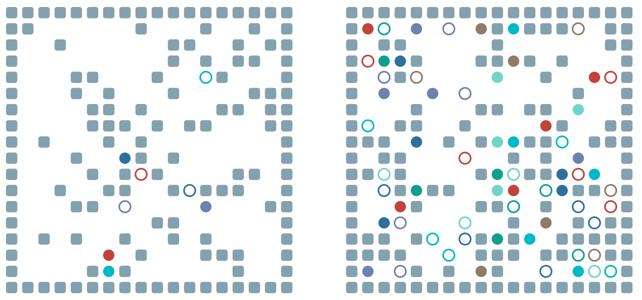

Figure 4: Easy and extra-hard Pogema configuration for 16×16 map.

randomly with density 30%. Maps with obstacle density between 30% and 40% present the greatest difficulty, since maps with less obstacles are trivial and maps with more obstacles are decomposed to several components. It's guaranteed that any agent can solve the task, if other agents do not interfere him. We provide four difficulty levels: *easy*, *normal*, *hard* and *extra-hard*, which corresponds to different density of agents on the map (see Table 2). For all these environments agent's the observation radius is the same, and equals to 5, which corresponds to $11x11$ field of view.

Custom Maps

POGEMA provides a wide range of customization options. First, we describe how to adjust random maps. The main settings are defined in *GridConfig*: size of the environment, obstacle density, number of agents, radius of agent's field of view and maximum length of the episode. In addition *GridConfig* has seeding option, which is *None* by default (thus, positions of obstacles, agents and targets are new after each reset).

```
# Define four rooms map as a string
grid = """
.....#.....
.....#.....
...........
.....#.....
.....#.....
#.####.....
.....###.##
.....#.....
.....#.....
...........
.....#.....
"""

# Define new configuration
grid_config = GridConfig(map=grid)

# Create Pogema environment using Gym
env = gym.make('Pogema-v0', config=grid_config)
```

Figure 5: A sample code for creating PO-MAPF instance with custom map in Pogema.

To create an environment with a custom map, one is supposed to specify *map* field in *GridConfig* (see Figure 5). Positions of agents and targets also can be defined in *GridConfig* if that needed. For the large configurations we suggest the option to store it in *YAML* format, which will be validated on POGEMA side.

Experiments

Planning Baseline

PO-MAPF problems can be solved with varied success by classical path-planning approaches such as A*(Hart, Nilsson, and Raphael 1968). Each agent plans its path independently, replanning the path on each iteration, receiving new observation. Other agents are considered as static obstacles, as their trajectories are unknown and they cannot communicate.

While pure A* can easily find a path for a single agent, in case of presence of large amount of agents on the map, they block each others paths. To increase the chances to find a solution, the suggested planning baseline uses some ad-hoc enhancements. First, in case of failure to find a trajectory for an agent, it moves to the adjacent cell that is the closest to the goal. Second, in case of indicating an oscillating behavior, when agent repeatedly moves between the same cells, a wait action with 50% probability is added.

The results of the planning baseline with different combinations of enhancements are presented in Table 3.

PyMARL Baselines

Centralized methods are the standard techniques in MARL tasks. In this experiment we show POGEMAcapabilities for benchmarking such kinds of algorithms. We use PyMARL implementations of such algorithms as: QMIX (Rashid et al. 2018), VDN (Sunehag et al. 2017) (see Figure 6). Also, we provide results for decentralized approach IQL (Tan 1993). For the experiment we used *extra-hard* version of 8×8 environment and trained each algorithm for 2 million steps. The results emphasize the requirement of cooperative behaviour of the agents.

APPO Baseline

Besides small PO-MAPF tasks, POGEMAis suitable for large-scale RL experiments. Figure 7 presents training results for all difficulties of $32x32$ configurations, trained with 100 millions steps. For this experiment, we use *SampleFactory* implementation of APPO algorithm. Fast environment and distributed RL framework allows us to train RL agent in a few hours on a single GPU, even with Impala (Espeholt et al. 2018) encoder and recurrent heads. Moreover, in $32x32$ *extra-hard* configuration were simultaneously trained 128 agents in each environment worker. Increasing density of the agents and map size, significantly decreases the performance of the algorithms. Which shows the importance of both pathfinding and conflict resolution components.

Conclusion

In this work we presented POGEMA fast and easy to use environment for creating a variety of PO-MAPF tasks. We've

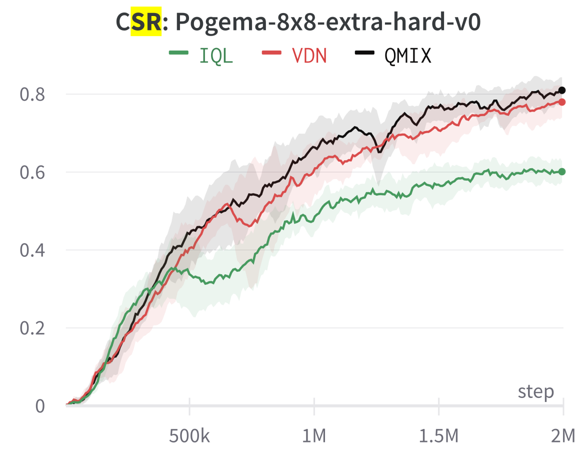

Figure 6: Learning curves of PyMARL algorithms. Centralized methods (QMIX, VDN) show better results, since *Extra-hard* version of POGEMA 8×8 requires cooperative interaction. The results are averaged over three runs.

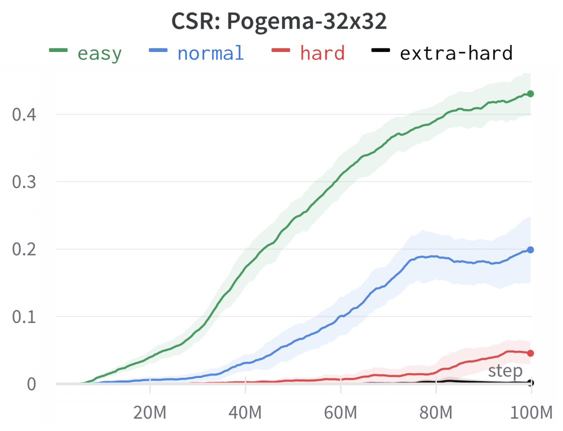

Figure 7: CSR metrics for APPO trained on 32×32 builtin POGEMA configurations.

designed a number of builtin configurations to help the community benchmark both learning and planning approaches. POGEMA environments are procedurally generated, which ensures agent's ability to generalization. To simplify further experimentation we open source our code[1] with RL and planning algorithms.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Čáp, M.; Novák, P.; Kleiner, A.; and Selecký, M. 2015. Prioritized planning algorithms for trajectory coordination of

[1]Code available at github.com/Tviskaron/pogema-baselines

multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3): 835–849.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 1407–1416. PMLR.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Gonzalez, J.; Goldberg, K.; and Stoica, I. 2017. Ray RLLib: A Composable and Scalable Reinforcement Learning Library. *CoRR*, abs/1712.09381.

Mohanty, S.; Nygren, E.; Laurent, F.; Schneider, M.; Scheller, C.; Bhattacharya, N.; Watson, J.; Egli, A.; Eichenberger, C.; Baumberger, C.; et al. 2020. Flatland-rl: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893*.

Petrenko, A.; Huang, Z.; Kumar, T.; Sukhatme, G.; and Koltun, V. 2020. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, 7652–7662. PMLR.

Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304. PMLR.

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. 2019. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant., N. R. 2015. Conflict-based search for optimal multiagent path finding. *Artificial Intelligence Journal*, 218: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019)*, 151–158.

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

Surynek, P. 2009. A novel approach to path planning for multiple robots in bi-connected graphs. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, 3613–3619. IEEE.

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

Terry, J. K.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sulivan, R.; Santos, L.; Perez, R.; Horsch, C.; Dieffendahl, C.; Williams, N. L.; Lokesh, Y.; Sullivan, R.; and Ravi, P. 2020. PettingZoo: Gym for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2009.14471*.

Wagner, G.; and Choset, H. 2011. M*: A complete multi-robot path planning algorithm with performance bounds. In *Proceedings of The 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, 3260–3267.

Zheng, L.; Yang, J.; Cai, H.; Zhang, W.; Wang, J.; and Yu, Y. 2017. MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. *CoRR*, abs/1712.00600.

Appendix

Table 2: Builtin POGEMA configurations.

Environment	agent density	num Agents	Episode length
Pogema-8x8-easy-v0	2.2%	1	64
Pogema-8x8-normal-v0	4.5%	2	64
Pogema-8x8-hard-v0	8.9%	4	64
Pogema-8x8-extra-hard-v0	17.8%	8	64
Pogema-16x16-easy-v0	2.2%	4	128
Pogema-16x16-normal-v0	4.5%	8	128
Pogema-16x16-hard-v0	8.9%	16	128
Pogema-16x16-extra-hard-v0	17.8%	32	128
Pogema-32x32-easy-v0	2.2%	16	256
Pogema-32x32-normal-v0	4.5%	32	256
Pogema-32x32-hard-v0	8.9%	64	256
Pogema-32x32-extra-hard-v0	17.8%	128	256
Pogema-64x64-easy-v0	2.2%	64	512
Pogema-64x64-normal-v0	4.5%	128	512
Pogema-64x64-hard-v0	8.9%	256	512
Pogema-64x64-extra-hard-v0	17.8%	512	512

Table 3: Percentage of successfully solved instances by planning baseline. GA - greedy actions. FL - fix loops.

	A*	A*+GA	A*+FL	A*+GA+FL
Pogema-8x8-easy-v0	100%	100%	100%	100%
Pogema-8x8-normal-v0	100%	100%	100%	100%
Pogema-8x8-hard-v0	82%	84%	90%	100%
Pogema-8x8-extra-hard-v0	60%	64%	66%	92%
Pogema-16x16-easy-v0	96%	96%	98%	100%
Pogema-16x16-normal-v0	68%	78%	96%	100%
Pogema-16x16-hard-v0	46%	50%	86%	100%
Pogema-16x16-extra-hard-v0	10%	14%	38%	84%
Pogema-32x32-easy-v0	38%	38%	98%	98%
Pogema-32x32-normal-v0	12%	16%	94%	96%
Pogema-32x32-hard-v0	0%	0%	62%	80%
Pogema-32x32-extra-hard-v0	2%	2%	6%	22%

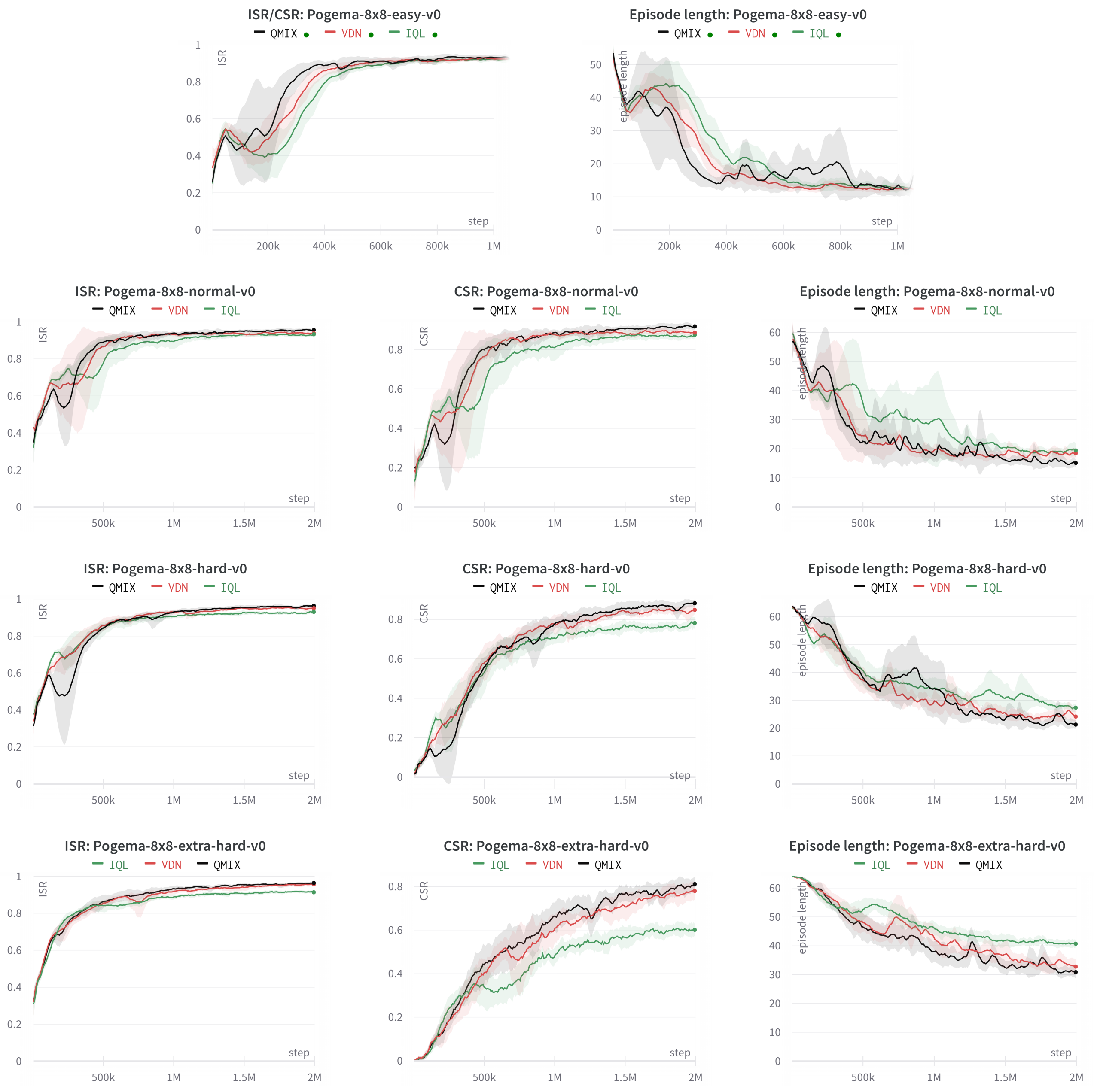

Figure 8: Results for QMIX, VDN and IQL for the all difficulties of 8×8 benchmark. There is only one agent in *easy* configuration, thus we report combined plot for ISR/CSR metrics.

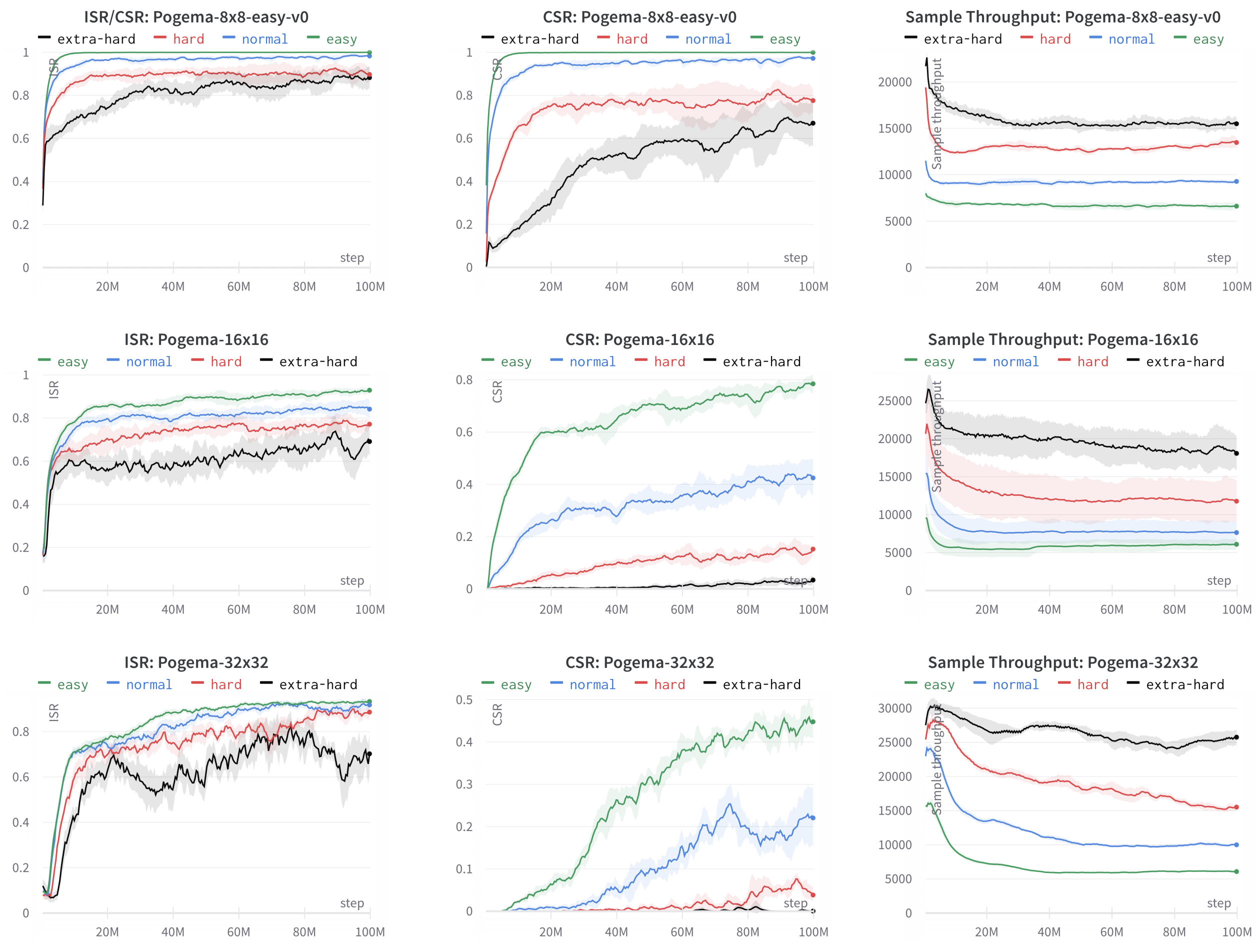

Figure 9: Results for APPO for the all difficulties of 8×8, 16×16, 32×32 benchmarks. Rights plots reports sample throughput for APPO.