Learning to Cooperate: Application of Deep Reinforcement Learning for Online AGV Path Finding

Extended Abstract

Yi Zhang, Yu Qian, Yichen Yao, Haoyuan Hu, Yinghui Xu
Cainiao Network
{shuding.zy,qianyu.qy,eason.yyc,haoyuan.huhy}@cainiao.com,renji.xyh@taobao.com

ABSTRACT

Multi-agent path finding (MAPF), naturally exists in applications like picking-up and dropping-off parcels by automated guided vehicles (AGVs) in the warehouse. Existing algorithms, like conflict-based search (CBS), windowed hierarchical cooperative A* (WHCA), and other A* variants, are widely used to find the shortest paths in different manners. However, in real-world environments, MAPF cases are dynamically generated and need to be solved in real time. In this work, a decentralized multi-agent reinforcement learning (MARL) framework with multi-step ahead tree search (MATS) strategy is proposed to make efficient decisions. Through performing experiments on a 30×30 grid world and a real-world warehouse case, our proposed MARL policy is proved to be capable of: 1) scaling to a large number of agents in real-world environment with online response time within acceptable levels; 2) outperforming existing algorithms with shorter path length and solution time, as the number of agents increases.

KEYWORDS

Multi-agent path finding; Multi-agent reinforcement learning

ACM Reference Format:

Yi Zhang, Yu Qian, Yichen Yao, Haoyuan Hu, Yinghui Xu. 2020. Learning to Cooperate: Application of Deep Reinforcement Learning for Online AGV Path Finding. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 3 pages.

1 INTRODUCTION

In the modern warehouses and factories, automated guided vehicles (AGVs) are widely utilized to perform end-to-end transportation tasks [17]. Variants of multi-agent path finding (MAPF) problems are raised for optimizing the efficiency of AGVs [7, 8]. In the literature, two categories of algorithms are popular for solving MAPF:

A*-based algorithms: rely on complete observations and utilize A* to calculate full paths for agents, which could work in both centralized (e.g. conflict-based search (CBS) [5, 12], windowed hierarchical cooperative A* (WHCA) [12, 13], ORCA [15]) and decentralized manners.

Learning-based algorithms: take local observations as input to decide one-step or limited length of paths for decentralized agents (e.g. a learning method called PRIMAL [10]).

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

The former methods allow AGVs to continuously control speeds and guarantees no conflict if the environment is determined, while the latter approaches are more robust to the changing world and more practical for real-time decision making scenario.

In recent years, there has been some seminal work on using deep architectures to automatically learn heuristics for combinatorial problems [1, 3, 4, 6, 9, 16]. These advances motivated us to propose a multi-agent reinforcement learning framework to parameterize the policy to obtain a stronger heuristic algorithm for path finding problem. Different from PRIMAL [10], we allow agents closely following the others, which is promising to fulfill more jobs in AGV path finding cases. However, to the best of our knowledge, there is no evidence to prove that a purely-decentralized MARL policy could perfectly avoid conflicts between agents.

2 LEARNING TO COOPERATE

2.1 MDP Definition

From a decentralized point of view, MDP is defined as:

State $s_t^k \in \mathcal{S}$: The state of AGV k at time step t considers information in the 5×5 neighboring grids of the AGV k, each grid consists of the positions of obstacles and other AGVs, the goal information of current AGV and other observable AGVs.

Action $a_t^k \in \mathcal{A}$: Action space of each AGV is divided into two parts: move in four cardinal directions or stay still.

Reward Function $r_t^k \in \mathcal{R} \leftarrow \mathcal{S} \times \mathcal{A}$: The one step reward is designed as -0.2 and -0.4 when an AGV moves towards or away from its target, -0.5 for staying still, -20 for colliding with obstacles or the other AGVs, +40 for arriving at the target grid.

2.2 MARL framework

Value Network. The state-value function is learned by minimizing the following loss function derived from Bellman equation:

$$L_{\theta_v} = \left(V_{\theta_v}\left(s_t^k\right) - V_{target}\left(s_{t+1}^k;\pi\right)\right)^2 \tag{1}$$

$$V_{target}\left(s_{t+1}^k;\pi\right) = r_t^k + V_{\theta'_v}\left(s_{t+1}^k\right) \tag{2}$$

Policy Network. In this paper, we use the same objective in the actor-critic algorithm from Proximal Policy Optimization [11]:

$$L_\theta = \mathbb{E}_t\left[\min\left(r_t\left(\theta\right), clip\left(r_t\left(\theta\right), 1-\epsilon, 1+\epsilon\right)\right)A_t\right] \tag{3}$$

$$A_t = -V_{\theta_v}\left(s_t\right) + \sum_{\tau=t}^{T-1} r_\tau\, \gamma^{\tau-t} + \gamma^{T-t} V_{\theta_v}\left(s_T\right) \tag{4}$$

where θ, θ_v denotes the parameters of policy network and value network.

Algorithm 1: Multi-agent Training with Searching

1: **for** $m = 1$ to $N_{episode}$ **do**
2: Reset environment and get initial state
3: **Stage 1: Sampling**
4: **while** $t < T$ **do**
5: **for** k=1 to K **do**
6: Calculate probability weights $\pi(a_t^k|s_t^k)$
7: Create root node v_0 with $Q_0 = 0, N_0 = 0$
8: **while** $n < N$ **do**
9: **if** any node $v_l \in V$ is not fully-expanded **then**
10: Expand $v_0 \longrightarrow v_l$ by choosing an action sequence $\{a_t, a_{t+1}, \dots a_{t+\tau}\}$ according to policy π
11: Estimate $s_{t+\tau}$ and $r_{t+\tau}$ by simulating actions
12: Add new child v_l to V with
13: **else**
14: Sample a node v_l with the maximum UCB
15: Backward propagation of parent and ancestor nodes by $Q_l = Q_l + q_l$, $N_l = N_l + 1$
16: **end if**
17: **end while**
18: Choose v_0 with the largest Q_l/N_l and its action a_t^k
19: Execute $\mathbf{a}_t^k$ and observe reward $\mathbf{r}_t^k$, next state $\mathbf{s}_{t+1}^k$
20: **end for**
21: Store the transitions $(s_t^k, a_t^k, r_t^k, s_{t+1}^k)$ into M
22: **end while**
23: **Stage 2: Learning**
24: Sample a batch of experience: s_t^k, $V_{target}(s_{t+1}^k; \pi)$
25: Update by minimizing the value loss Eq.(1) over the batch
26: Update as $\theta \longleftarrow \theta + \nabla_\theta L_\theta$ according to Eq.(3)
27: **end for**

2.2.1 Multi-step ahead tree search (MATS) strategy. As shown in Line 7-18 in Algorithm 1, a tree in depth τ is created to enumerate possible states of AGV k by randomly sampling actions in the next τ timesteps from the root node v_0. The node depth (time level) τ, action from the father node $a_{t+\tau}$, state $s_{t+\tau}$, cumulated reward from the root node $q_l = r_{t+\tau}$, cumulated score after rounds of samples Q_l, sample times N_l will be stored at each node. N rounds of Monte-Carlo samples will be performed on all the paths. In each round, the terminal node with the highest upper confidence bound (UCB) will be chosen, which is calculated by Eq.(5). Here, C_p is a positive constant denoting exploration rate.

$$UCB_l = \frac{Q_l}{N_l} + C_p \frac{p(s_t, a_t)}{1 + N_l} \tag{5}$$

2.2.2 Postprocessing method. To avoid all possible conflicts, we postprocess actions by the following steps: 1) Return if the given action is conflict-free; 2) Sort the other four actions in a decreasing order according to the probability weights from the policy; 3) Choose the first action and go to step 1.

3 RESULTS AND COMPARISON ANALYSIS

3.1 Square grid world

In this 30×30 example, there are 7 distributing sources (pick-up places) and 80 equally spaced sinks (drop-off places).

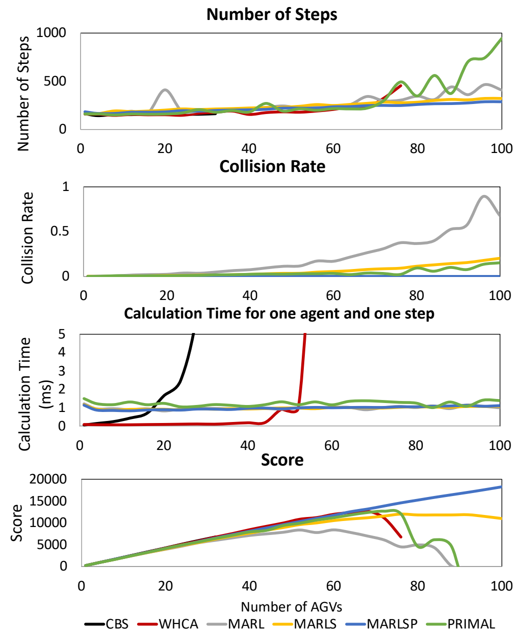

Figure 1: Comparison results for AGV path finding in the 30×30 grid world by different algorithms.

In this experiment, the policies we used is learned from the environment with only 10 agents. As shown in the results in Figure 1, the scores of the simplest MARL policy (trained without MATS strategy) and PRIMAL drop dramatically to negative values when more agents are involved. By forcing all the agents not to follow the others, many collisions are naturally avoided in PRIMAL. But the side effect of this assumption for PRIMAL in the AGV path finding case is its longer paths and lower efficiency than our MARL policy. With the help of MATS strategy, MARLS overcomes the drawback of MARL and decreases the collision rates to 10%-20% for 100 agents. Benefit from postprocessing, MARLSP avoids all conflicts and is proved to be the best trained policy. Even though WHCA and CBS provides shorter paths, higher scores than MARLSP in parser worlds, the calculation time for one-step decision exceeds 20ms (strict limitation for online use) as the number of agents increases.

3.2 Real-world warehouse

In the real-world warehouse, the graph size is 63×115. There are 299 sinks with 8 sources on both west and east sides. Since MARLSP outperforms many other RL policies in this work, we only compare MARLSP with WHCA and PRIMAL in this section. Similar with the results in the previous example, MARLSP beats WHCA only when the number of agents is larger than 150, and totally wins PRIMAL in nearly all the conditions.

4 CONCLUSION

In conclusion, MARL policy is efficient and effective for solving AGV path finding problems in denser worlds. The multi-step ahead tree search (MATS) strategy and postprocessing methods in this work significantly improve the scalability and robustness of policies.

REFERENCES

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*.

[2] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games* 4, 1 (2012), 1–43.

[3] Yujie Chen, Yu Qian, Yichen Yao, Zili Wu, Rongqi Li, Yinzhi Zhou, Haoyuan Hu, and Yinghui Xu. 2019. Can Sophisticated Dispatching Strategy Acquired by Reinforcement Learning?. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 1395–1403.

[4] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. 2019. A Multi-task Selected Learning Approach for Solving 3D Flexible Bin Packing Problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 1386–1394.

[5] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, TK Satish Kumar, and Sven Koenig. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

[6] Wouter Kool and Max Welling. 2018. Attention Solves Your TSP. *CoRR* abs/1803.08475 (2018). arXiv:1803.08475 http://arxiv.org/abs/1803.08475

[7] Hang Ma and Sven Koenig. 2016. Optimal target assignment and path finding for teams of agents. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1144–1152.

[8] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, TK Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon. 2017. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515* (2017).

[9] MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. 2018. Deep Reinforcement Learning for Solving the Vehicle Routing Problem. *CoRR* abs/1802.04240 (2018). arXiv:1802.04240 http://arxiv.org/abs/1802.04240

[10] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. 2019. PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters* 4, 3 (2019), 2378–2385.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). http://arxiv.org/abs/1707.06347

[12] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[13] David Silver. 2005. Cooperative Pathfinding. *AIIDE* 1 (2005), 117–122.

[14] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[15] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-body collision avoidance. In *Robotics research*. Springer, 3–19.

[16] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 2692–2700.

[17] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine* 29, 1 (2008), 9–9.