Distributed Reinforcement Learning for Multi-Robot Decentralized Collective Construction

Guillaume Sartoretti[1], Yue Wu[1], William Paivine[1],
T. K. Satish Kumar[2], Sven Koenig[2], and Howie Choset[1]

[1] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15203, USA
gsartore@cs.cmu.edu,
WWW home page: http://www.sartoretti.science
[2] University of Southern California, Los Angeles, CA 90007, USA

Abstract. Inspired by recent advances in single agent reinforcement learning, this paper extends the single-agent asynchronous advantage actor-critic (A3C) algorithm to enable multiple agents to learn a homogeneous, distributed policy, where agents work together toward a common goal without explicitly interacting. Our approach relies on centralized policy and critic learning, but decentralized policy execution, in a fully-observable system. We show that the sum of experience of all agents can be leveraged to quickly train a collaborative policy that naturally scales to smaller and larger swarms. We demonstrate the applicability of our method on a multi-robot construction problem, where agents need to arrange simple block elements to build a user-specified structure. We present simulation results where swarms of various sizes successfully construct different test structures without the need for additional training.

Keywords: multi-agent system, collaboration, distributed learning

1 Introduction

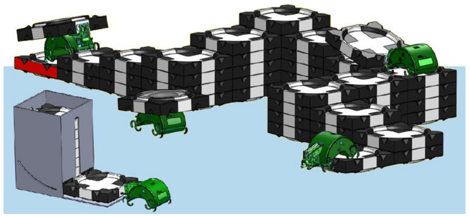
Fig. 1. TERMES robots and testbed (courtesy of Petersen et al. [1]).

Recent years have seen massive leaps forward in single-agent artificial intelligence, in particular in deep-reinforcement learning (deep-RL) [2, 3, 4, 5]. A large community has been focusing on multi-agent reinforcement learning (MARL), interested in extending these single-agent approaches to multi-agent systems. However, natural extensions of single-agent approaches fail when applied to multi-agent problems [6, 7, 8]. The joint MARL problem can rarely be solved [9, 10, 11], mainly due to the high-dimensional state-action space that has to be explored by agents. In this context, the community has been striving for distributed solutions [12, 13, 14, 15]. In this paper, we propose a fully-distributed, scalable learning framework, where multiple agents learn a common, collaborative policy in a shared environment, that can then be deployed on an arbitrary number of agents with little to no extra training.

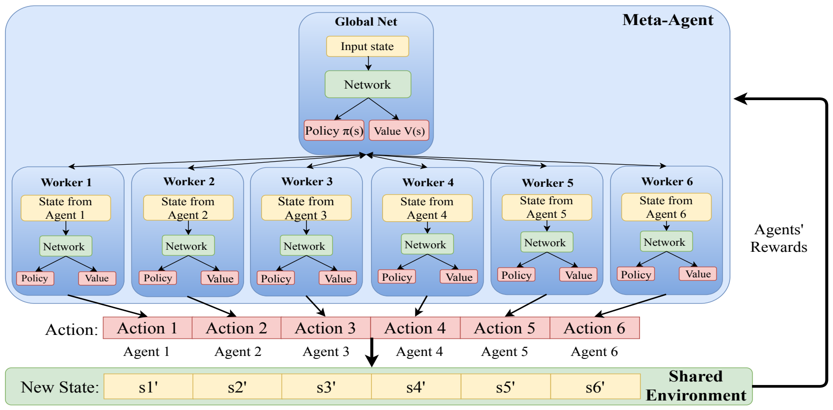

Fig. 2. Structure of the proposed shared learning framework, with 6 agents contained in a meta-agent (similar to the A3C meta-agent) interacting with a shared environment. The global Actor-Critic network is stored in the meta-agent. Each internal agent (worker) draws its current state from the shared environment, and selects an action to be performed next. Individual agents' actions are performed synchronously, in any order, and yield individual rewards. Each worker regularly pushes gradients to the global network, after which it pulls the most recent weights from the updated network.

We focus on a multi-robot construction problem, inspired by Harvard's TERMES project [1, 16], where simple robots must gather, carry and place simple block elements to build a user-specified 3D structure (see Fig. 1). This problem, cast in the reinforcement learning (RL) framework, is a very difficult game with sparse and delayed rewards, as robots need to build scaffolding to reach the higher levels of the structure to complete construction. We use a centralized learning architecture, but fully decentralized execution where each agent runs an identical copy of the policy without explicitly communicating with other agents, and yet a common goal is achieved. The policy is learned centrally, meaning that all agents contribute to it during training (Fig. 2). Once learned, the policy can be implemented on an arbitrary number of agents, and lets each agent view other agents as moving features in the system. To this end, we extend the single-agent asynchronous advantage actor-critic (A3C) algorithm [2] to let multiple agents learn a homogeneous, collaborative policy in a shared environment.

In our recent works, we have studied the applicability of distributed approaches to multiple agents learning a homogeneous policy in a single environment [17]. However, agents could not observe each other as part of each other's state, and learning was performed offline based on recorded experience from a conventional controller. In this work, our main contribution is to extend this framework to online learning of scalable, collaborative policies in a fully-observable system. In our extension, multiple agents are assumed to evolve in a common environment, and to learn from the same episodes, though experienced from different points of view. We rely on experience replay to stabilize the learning and help decorrelate training gradients, and show how a carefully crafted reward structure is necessary to let agents learn collaborative policies in the absence of explicit communications.

This paper is structured as follows: Section 2 provides a short background on multi-agent learning and on the A3C algorithm. Section 3 presents the multi-robot construction problem, and casts it in the RL framework. Section 4 details the online learning process. Section 5 presents and discusses our numerical results. Finally, Section 6 provides concluding remarks.

2 Background

2.1 Single-Agent Asynchronous Distributed Learning

In early 2016, Mnih et al. proposed a novel single-agent learning approach for deep reinforcement learning [2], taking advantage of multi-core architectures to obtain near-linear speed-up via distributed learning. In these now state-of-the-art methods, the learning task is distributed to several agents that asynchronously update a global, shared network, based on their individual experiences in independent learning environments. That is, these methods rely on multiple agents to distributively learn a single-agent policy. A general meta-agent stores a global network, to which the worker agents regularly push their individual gradients during training, after which they pull the most recent global weights that aggregate the lessons learned by all agents.

The increase in performance is mainly due to the fact that multiple agents, experiencing different situations in independent environments, produce highly-decorrelated gradients that are pushed to the global net. Additionally, agents in independent environments can better cover the state-action space concurrently, and faster than a single agent (e.g., a deep Q-learning agent [18, 5, 19]). Among these asynchronous methods, the A3C algorithm, extending the standard Advantage Actor-Critic learner to asynchronous distributed learning, was shown to produce the fastest learning rate in a very stable manner [2]. In A3C, the global network has two outputs: (i) a discrete stochastic policy, and (ii) a value (the critic), estimating the long-term cumulative reward from the current state.

2.2 Multi-Agent Learning

The first and most important problem encountered when transitioning from the single- to multi-agent case is the curse of dimensionality: most joint approaches fail as the dimensions of the state-action spaces explode combinatorially, requiring an absurd amount of training data to converge [6]. In this context, many recent works have focused on decentralized policy learning [12, 13, 14, 15], where agents each learn their own policy, which should encompass a measure of agent cooperation at least during training. One way to do so is to train each agent to predict the other agents' actions [13, 14]. In most cases, some form of centralized learning is involved, where the sum of experience of all agents is used toward training a common aspect of the problem (e.g., network output or value/advantage calculation) [12, 14, 15]. When centrally learning a network output, parameter sharing can be used to let agents share the weights of some of the layers of the neural network and learn faster and in a more stable manner [12]. In actor-critic approaches, for example, the critic output of the network is often trained centrally with parameter sharing, and can be used to train cooperation between agents [12, 14]. Centralized learning can also help when dealing with partially-observable systems, by aggregating all the agents' observations into a single learning process [12, 14, 15]. Many of these approaches rely on explicit communication among agents, to share observations or selected actions during training and sometimes policy execution [13, 14, 15].

3 Multi-Robot Construction

3.1 Problem Statement and Background

The problem outlined in this paper is inspired by the TERMES project, started at Harvard University, which focused on multi-robot construction of complex structures (see Fig. 1). The TERMES testbed [1, 16] consists of small autonomous mobile robots and block sources of approximately the same size as the robots. Robots need to gather blocks from the sources to collaboratively build a user-specified structure.

In the original framework, valid actions for each robot included turning 90 degrees left or right, moving one location forward, and picking up/putting down a block in front of itself. In this work, however, we allow robots to move, as well as pick up/put down blocks in the four main compass directions around them. We leave all other rules defining valid actions unchanged. That is, robots can move to an adjacent location if and only if doing so would move the robot vertically up to one block up or down. Additionally, robots can pick up/put down a block under the condition that it is (or, once placed, becomes) a top block, on the same level as the robot itself. Each robot can carry at most one block.

The original Harvard TERMES robots coordinated using simple local rules based on robots' behaviors and stigmergy, which only allowed them to construct simple structures [1]. Later, in our previous work, we developed a better single-robot planning method [20], which we then generalized to multi-robot planning [21]. However, our conventional planning method is domain-dependent, centralized, and non-optimal. The planner attempts to minimize the total number of block operations (rather than common objectives such as the makespan), and restricts the robots' movements to the edges of the spanning tree of the grid, sometimes resulting in longer paths than necessary. Furthermore, the design of this planner heavily relied on human-based domain knowledge of the problem.

3.2 Multi-Robot Construction as an RL Problem

State Space We consider a discrete 3D representation of the robots' states (and of the world state). That is, for each agent, we directly encode information about the current state of the world in 3D tensors of the same dimensions as the discrete 3D world grid. In order to demonstrate our approach, while keeping training time reasonable, we first focus on a simple $10 \times 10 \times 4$ world (x, y, z). Similar to recent approaches which learned to play games from raw images [3, 5, 4], we decompose the world state into different channels to simplify the learning task for the agents. Specifically, the state s_i of agent i is a 5-tuple of binary tensors:

- Current position: A one-hot tensor (single 1 at the current position of agent i).
- Agents' positions: A binary tensor showing the positions currently occupied by agents in the world (agent i included).
- Blocks' positions: A binary tensor expressing the positions of the blocks.
- Sources' positions: A binary tensor showing the positions of all block sources. Sources are always positioned at the ground level of the world, and no agent/block can ever be found/placed on top of them, thus representing a "tower" of blocks agents can pick from, equivalent to an infinite reservoir of blocks.
- Structure plan: A binary tensor expressing the user-specified structure's plan. This 3D tensor only contains the position of the blocks in the final structure to be built, and in particular does not contain any of the scaffolding blocks that may needed to construct the structure.

We observed that, by separating information into different channels rather than placing them all into a single world tensor, agents seem to learn faster and obtain a more stable policy. We believe that this is due to the fact that, with the proposed state space, agents are able to learn a single type of information from each binary channel of the state tuple. However, note that the resulting learning task is nontrivial, since agents need to merge information from different channels to make decisions. That is, agents still make sequential decisions from raw data, even though these data have been pre-processed into different channels (similar to color channels in an image).

Action Space As commonly done in such discrete grid worlds, we rely on a discrete action set for each agent. The 13 possible actions are four "move" actions (move North/East/South/West), four "pick up" actions (N/E/S/W), four "put down" actions and one "idle" action. Note that, at any time in the simulation, an agent only really has access to at most 9 actions, since agents carrying a block can only put down a block and not pick up an additional block, and vice-versa.

As detailed further in Section 4, agents learn which actions are valid based on their current state; however, due to the fact that agents' actions are executed sequentially in any order, an agent may sometimes pick an action that has become impossible to perform due to another agent's action. In such cases, nothing happens and the agent remains idle for one time step.

Reward Structure In order to minimize the amount of domain knowledge introduced in the learning task, we propose a minimalist, symmetric reward structure. A base action cost r_0 (in practice, $r_0 = -0.02$) incentivizes agents to complete the desired structure as fast as possible. During the building phase (i.e., until the structure is completely built), agents get a positive/negative reward $\pm r_1$

(in practice, $r_1 = 1$) for putting down/picking up a correct block (i.e., a block put down at a position featured on the user-specified structure plan), but no penalty for putting down/picking up incorrect blocks. During the cleanup phase (i.e., when agents should clean up scaffolding blocks), agents additionally get a positive/negative reward for putting down/picking up incorrect blocks (blocks that are not part of the plan).

Moreover, since agents cannot place blocks on higher levels of the world before building a ramp to access these levels, we propose to adapt the positive/negative rewards associated with putting down/picking up higher correct blocks. That is, we observe that to place a block at the n-th level of a simple tower of blocks, agents need to build a ramp (or rather, a staircase) of height $n-1$ to make the n-th level reachable. This ramp will be made out of $\frac{n(n-1)}{2}$ blocks, which may need to be put down in locations which the plan does not denote requiring a block (in the worst-case scenario). Therefore, to incentivize agents to build such a ramp, we propose to multiply the reward associated with 6putting down/picking up the n-th level block by n^2. The resulting reward structure is summarized in the following table:

	Building phase		Cleanup phase	
Action	Correct (n-th level)	Incorrect	Correct (n-th level)	Incorrect
Pick	$r_0 - r_1 \cdot n^2$	r_0	$r_0 - r_1 \cdot n^2$	$r_0 + r_1$
Place	$r_0 + r_1 \cdot n^2$	r_0	$r_0 + r_1 \cdot n^2$	$r_0 - r_1$
Move	r_0			

Actor-Critic Network In this work, we use centralized learning with shared parameters, for both the actor and critic networks, but fully decentralized policy execution. We use the 9-layer convolutional network pictured in Fig. 3. First, 5 3D convolution layers compress the input dimension from 3D to 1D to connect to the next 2 fully-connected (fc) layers. The output of the last fc layer is then fed into a long-short-term memory (LSTM) cell with output size 512. Residual shortcuts [22] connect the input layer to the third convolutional layer, the third layer to the first fc layer, and the first fc layer to the LSTM. The output layer consists of the policy neurons with softmax activation, and a value output predicting the reward. We also added a feature layer connected to the LSTM [3], to train the agent on whether it is carrying a block and whether the structure has been completed.

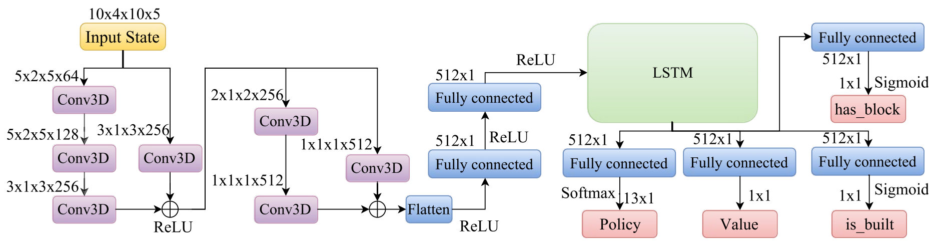

Fig. 3. Nine-layer deep convolutional neural network used for policy estimation.

To prevent invalid outputs, e.g. trying to put down a block when the agent does not have a block at hand, we adapted the way we calculate the reward for the actor. At time step t, let $\pi(a_t|s_t;\theta)$ be the policy estimation for action a (actor) and $V(s_t;\theta_v)$ the value estimation (critic) with parameters θ, where s_t represents the state and a_t the action. We update the policy according to [2]:

$$L_t(\theta') = log\Big(\pi(a_t|s_t;\theta')\, A(s_t,a_t;\theta,\theta_v)\Big), \tag{1}$$

where $A(s_t,a_t;\theta,\theta_v)$ refers to the advantage estimate:

$$\sum_{i=0}^{k-1}\left(\gamma^i r_{t+i}\right) + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v). \tag{2}$$

We also minimize the log-likelihood of the model selecting invalid actions:

$$L_v(\theta) = \sum\nolimits_a log\Big(P\big(Q(s,a;\theta)=1|y_a=0\big)\Big), \tag{3}$$

where $y_a = 0$ if action a is invalid. By minimizing this additional loss function, the model successfully learns to avoid selecting invalid actions.

4 Online Learning

We train multiple agents in a shared environment, randomly initialized in the beginning of each episode. Episodes last 1000 time steps, and training updates occur every 100 time steps. Initial blocks are placed at random in the environment, and the initial agents' positions are also randomized. Additionally, we randomize the goal structure in the beginning of each episode, to train agents to build any structure. The only constant during training is the positions of the sources (one in each corner of the map). The full training code is available online at https://goo.gl/9bZFXn.

During training, the Boltzmann action-selection strategy [23], often used to encourage exploration, did not seem to suffice for this task. Therefore, we push agents to explore the state-action space more intensively, by picking a valid action uniformly at random, every time the state predictor "has_block" (estimating whether the agent is currently carrying a block) is incorrect. This mechanism is mostly active during early training, since this output is quickly learned and

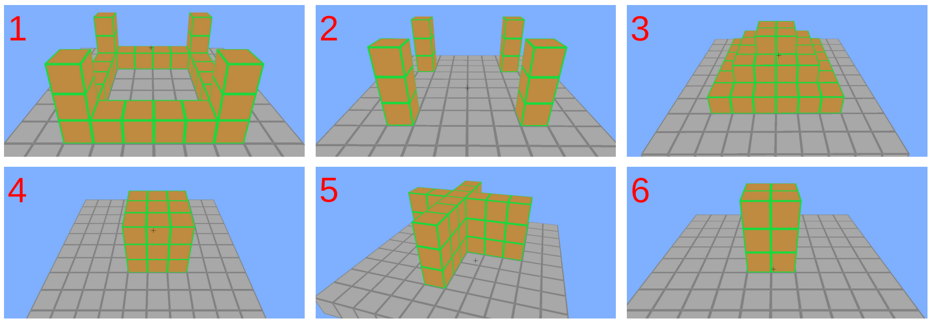

Fig. 4. Goal structures used to estimate the performance of the trained policy. The structures are composed of 28, 12, 56, 27, 33, and 12 blocks respectively.

achieves a near-100% prediction accuracy. Since actions are chosen at random, value learning is disabled when agents pick random actions (since learning would not be on-policy then, as requested in the actor-critic framework). Finally, we concurrently train several groups of $N_a = 4$ agents in $N_e = 4$ different environments, to speed up training.

For the first part of training (first $52,000$ episodes), we trained agents using experience replay [24], to help them further decorrelate their learning gradients. We create an experience pool of the most recent $N_a = 4$ episodes for each agent. During a training step, each agent draws an experience sample from a randomly selected epoch (100 time steps) of its pool and learns from it. After $52,000$ episodes, as rewards started to plateau, we stopped training, and resumed it without experience replay. We also decreased the learning rate (half the initial one, which was $\gamma = 2 \cdot 10^{-5}$), a standard practice that allows more precise convergence. At this point, we also started initializing 50% of the episodes with a world featuring a fully-built randomized structure alongside extra blocks, to more specifically train agents to clean up blocks after completing the construction. This approach did not allow agents to learn to clean up perfectly (as discussed in the next section), but nevertheless seemed to improve their cleanup abilities compared to that mid-training.

5 Simulation Results

5.1 Results

Training is performed on a desktop computer, equipped with an AMD Threadripper 1950X (32 cores) clocked at 3.4GHz, 32Gb RAM and 2 NVIDIA 1080Ti GPUs. Up to two training instances can be run in parallel, each being assigned 16 cores (one per agent) and one GPU. The training time is very long: it took a little less than 9 days to learn the final policy. This is mainly due to the GPU being the resource bottleneck when learning with so many cores, a problem exacerbated by the additional synchronization between agents during training.

To validate the trained policy, we systematically investigated its capacity to construct any user-input structure from an empty or randomly initialized world, as well as to scale to smaller/larger numbers of agents. To these ends, we selected 6 structures that test various aspects of the construction problem (Fig. 4), such as building wide areas, tall structures, tall towers, and walls. For every structure and every swarm size, we run 100 trials on initially empty and 100 trials on randomized, initially nonempty worlds. Trials are capped at $1,000$ time steps.

Worlds and structure plans are randomized uniformly by layer: the first layer consists of uniformly placed blocks with a randomized density (uniformly sampled between 0 and 0.5). Subsequent layers are uniformly sampled in the same way (with the same density), but only blocks that lie on top of existing blocks are kept. For initially empty worlds, only the goal structure is randomized and the world left empty; for nonempty worlds, both world state and structure are randomized. During testing, we measure the number of steps needed to construct the structure (construction time), and the percentage of test cases in which the structure was successfully built (construction accuracy). If the structure was constructed, we further measure the percentage of scaffolding removed after the structure was completed (cleanup accuracy), and report the percentage of trials with full construction and cleanup (completion accuracy).

We first note that all structures are not equal under the construction time and accuracy metrics. Most structures (4 out of 6) are completed with a mean accuracy quickly reaching close to 90% while the construction time decreases, as the number of agents is increased (until a bottleneck point). That is, we notice that the construction time decreases with the number of agents until 8 agents, and then slightly re-increases. This result does not seem surprising, since increasing the number of agents also increases the level of noise in the system (since the policy is suboptimal). That is, the cumulative effects of numerous imperfect actions may result in longer planning time with numerous agents, and seems to actually degrade performances past 8 agents, at which point it seems to offset the advantage of running more agents.

Structures 3 and 4 perform significantly worse than the other structures on these metrics, and we separate their results from that of the four other structures in Fig. 5. When attempting to build structure 3, agents struggle to complete the base layer of the pyramid, as their instinct is to reach for the top levels of the structure as quickly as possible (higher rewards), often leaving holes in the pyramid's base that cannot be filled afterwards. Similarly, the top layer of the

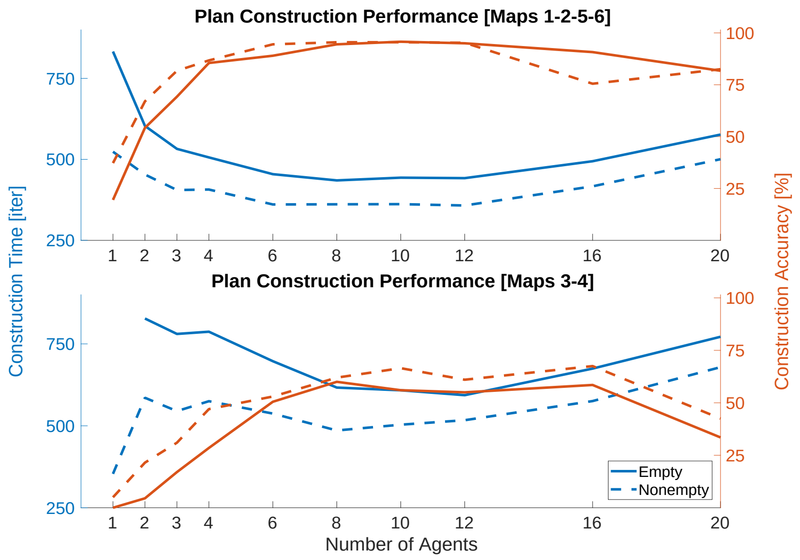

Fig. 5. Construction times (in time steps, capped at 1000 per trial) and plan construction accuracy, based on number of agents, averaged over i) the 4 test structures that yield similar, good performance (top), ii) the 2 test structures that yield similar, poorer performance (bottom). For each structure, averages are shown for 100 trials with random initial agent positions, once from empty worlds (plain line) and once from randomly initialized worlds (dotted).

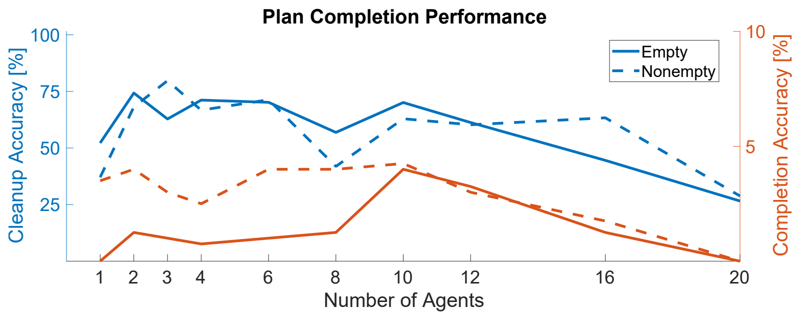

Fig. 6. Cleanup and completion (construction+cleanup) accuracy, wrt. swarm size, averaged over all test structures. Average over 100 trials with random initial agent positions, from empty worlds (plain) as well as from randomly initialized worlds (dotted). The decrease in completion accuracy naturally follows that of the cleanup accuracy.

cube structure 4 is a big challenge for agents, which exhibit difficulties building scaffolding along all faces of the cube to fully complete it, and sometimes seem to take adversarial action with respect to one another.

However, under the cleanup and completion accuracy metrics, all structures seem to yield similar results, which are aggregated in Fig. 6. There, we see that agents are consistently able to cleanup around $60 - 75\%$ of the scaffolding, but have a very low success rate ($2 - 3\%$ only) at completing the plan (construction and full cleanup). We believe, however, that a more specialized training process could be investigated to improve the agents' cleanup performance. Fig. 7 shows several steps during the distributed construction of structure 3 by 8 agents. Example good and bad results, starting from empty or randomly initialized worlds, and with various numbers of agents are available at https://goo.gl/C7r6uf.

5.2 Comparison with Conventional Planning

We further compared the results of our learning-based approach to that of our previous work, which relied on a (conventional) tree-based planner [25]. Our previous approach focused on finding the smallest number of block placements/removals needed by a single agent to complete the user-input structure from an empty world. Note that the minimal number of block placements needed to construct a structure does not depend on the number of agents.

To compare both approaches, we computed the minimum number of block placements/removals needed to complete each of our six test structures, using the single-agent conventional planner described in [25]. In parallel, for each structure, we ran trials of our learned policy from an empty world, with a single agent whose initial position is randomized. The agent follows the trained multi-agent policy (initially learned on four agents), and we measure the number of block placements needed to construct the structure. We average the result over 100 successful trials, i.e., trials where the structure was constructed within 2000 time steps. The results are summarized in the following table:

Structure	Conventional Planner [25]	Learned Policy (mean $\pm$ std)
1	36	114 ± 26
2	36	207 ± 53
3	56	154 ± 21
4	33	215 ± 32
5	48	197 ± 24
6	18	228 ± 44

Our learning-based approach results in far more blocks being placed/removed during the construction of the structure. This makes sense, since the learned policy is not trained to minimize the number of block placements/removals. In the learning-based approach, agents are trained to maximize their rewards, i.e., the placement of correct (higher) blocks, at the cost of sometimes unnecessary or redundant scaffolding. In fact, we believe that, as the number of agents grows, agents may actually benefit from building redundant scaffolding to reach the higher levels of the world. That is, more scaffolding can allow agents to concurrently reach the higher levels of the world, thus reducing the inherent bottleneck associated with the use of a single, “optimal” ramp (i.e., minimal number of placed blocks).

5.3 Discussion

Our distributed learning approach successfully generates reasonable construction plans using little domain knowledge of the task. Additionally, compared to previous approaches which only considered initially empty worlds[16, 25], agents are able to learn how to use blocks initially present in the world to build the structure faster (as displayed in Fig. 5). Most importantly, our approach inherently allows scalability to different numbers of agents, as the different agents are completely independent, and do not require explicit interactions or a centralized planner. As shown in Fig. 5, the trained policy can essentially be “copied” onto an arbitrary number of agents, which will then work together to construct the plan efficiently. Even though training was only ever performed using four agents, the learned policy was able to generalize to other swarm sizes, despite never having experienced any situation with a different swarm size. As shown by the resulting robust, population-independent policy, this approach has great advantages in situations where the number of robots is unknown or where a centralized planner cannot be used. In particular, our approach makes the swarm robust to the failure of an arbitrary number of robots, while maintaining construction accuracy. Note that, depending on the location at which a robot fail, that robot may actually act as an obstacle for other agents. However, provided some robots fail away from the structure, our approach would generally allow the remaining robots to still complete the structure.

However, our approach also has its limitations: more actions are needed for plan completion and more scaffolding is built than with an optimized centralized planner. In addition, fully cleaning up the world remains a challenge for the agents. As mentioned above, a more specialized training process could train

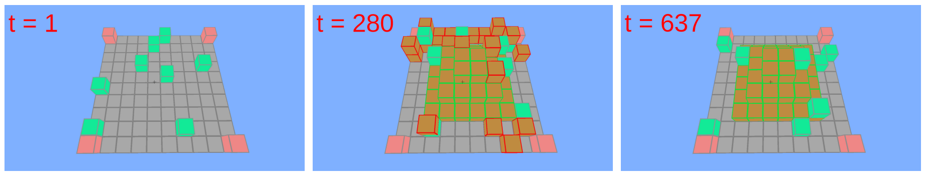

Fig. 7. Example frames of the construction of test structure 3 (Fig. 4), by a team of 8 agents (and respective time steps). Green-/red-edged cubes are correct/incorrect blocks, and solid green/pink cubes are agents/sources.

agents to better clean up scaffolding. Additionally, since the number of remaining scaffolding blocks is usually low, we believe that conventional planners could also be an option to finalize the cleanup process. Finally, and despite the relatively high versatility of our approach in the majority of testing scenarios, we observe that some structures requiring a more "strategic" sequence of actions from an early construction stage cannot be constructed with a high accuracy.

6 Conclusion

In this paper, we presented an extension of the A3C algorithm, which allows multiple agents to distributedly learn a collaborative policy in a shared environment. This policy, once learned, sees other agents as moving features in the environment, around which an agent needs to act in order to complete the collective construction task. In this distributed framework, agents do not cooperate per se (no joint action selection), but learn to work together toward a common goal, in the absence of explicit communication (but in a fully-observable system). We apply this approach to a distributed construction problem where multiple robots must coordinate their actions to gather and place blocks to build a user-specified structure. There, we show that the resulting homogeneous collaborative policy allows agents to construct arbitrary structures, and naturally scales to other swarm sizes while upholding construction accuracy. However, agents seem to struggle with fully cleaning up the scaffolding needed during construction.

In future work, we would like to investigate the re-usability of our neural network to different world dimensions. The current approach requires fully retraining the policy if the world size changes. However, we believe that some of the layers could be reused when the world dimensions are increased, which could drastically reduce the additional training time and make our approach more scalable with the world size. Finally, we believe that our distributed approach can be extended to other multi-agent problems of the same subclass, including collaborative manipulation and multi-agent path planning.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392 and 1817189. Detailed comments from anonymous referees contributed to the presentation and quality of this paper.

Bibliography

[1] Petersen, K., Nagpal, R., Werfel, J.: TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction. In: International Conference on Robotics: Science and Systems (2011)

[2] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the International Conference on Machine Learning, pp. 1928–1937 (2016)

[3] Lample, G., Chaplot, D.S.: Playing FPS Games with Deep Reinforcement Learning. In: AAAI Conference on Artificial Intelligence, pp. 2140–2146 (2017)

[4] Oh, J., Guo, X., Lee, H., Lewis, R.L., Singh, S.: Action-conditional video prediction using deep networks in Atari games. In: Advances in Neural Information Processing Systems, pp. 2863–2871 (2015)

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)

[6] Buoniu, L., Babuška, R., De Schutter, B., Srinivasan, D., Jain, L.C., Buoniu, L., Babuška, R., De Schutter, B.: Multi-agent reinforcement learning: An overview. Studies in Computational Intelligence **310**, 183–221 (2010)

[7] Neto, G.: From single-agent to multi-agent reinforcement learning: Foundational concepts and methods. Learning theory course (2005)

[8] Schwartz, H.M.: Multi-agent machine learning: A reinforcement approach. John Wiley & Sons (2014)

[9] Lau, Q.P., Lee, M.L., Hsu, W.: Coordination guided reinforcement learning. International Conference on Autonomous Agents and Multiagent Systems pp. 215–222 (2012)

[10] Guestrin, C., Lagoudakis, M., Parr, R.: Coordinated reinforcement learning. International Conference on Machine Learning pp. 227–234 (2002)

[11] Le, H.M., Yue, Y., Carr, P.: Coordinated Multi-Agent Imitation Learning. arXiv preprint arXiv:1703.03121 (2017)

[12] Gupta, J.K., Egorov, M., Kochenderfer, M.: Cooperative multi-agent control using deep reinforcement learning. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 66–83. Springer (2017)

[13] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O.P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems, pp. 6382–6393 (2017)

[14] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. arXiv preprint arXiv:1705.08926 (2017)

[15] Foerster, J., Assael, I.A., de Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 2137–2145 (2016)

[16] Koenig, S., Kumar, S.: A Case for Collaborative Construction as Testbed for Cooperative Multi-Agent Planning. In: ICAPS-17 Scheduling and Planning Applications Workshop (2017)
[17] Sartoretti, G., Shi, Y., Paivine, W., Travers, M., Choset, H.: Distributed Learning for the Decentralized Control of Articulated Mobile Robots. In: accepted to ICRA 2018, Brisbane, Australia, May 21-25, 2018 (2018)
[18] Hausknecht, M., Stone, P.: Deep recurrent Q-learning for partially observable mdps. CoRR, abs/1507.06527 **7**(1) (2015)
[19] Van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-Learning. In: AAAI Conference on Artificial Intelligence, pp. 2094–2100 (2016)
[20] Kumar, S., Jung, S., Koenig, S.: A Tree-Based Algorithm for Construction Robots. In: International Conference on Automated Planning and Scheduling (2014)
[21] Cai, T., Zhang, D., Kumar, S., Koenig, S., Ayanian, N.: Local Search on Trees and a Framework for Automated Construction Using Multiple Identical Robots [Short Paper]. In: International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 1301–1302 (2016)
[22] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
[23] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. 1998. A Bradford Book (1998)
[24] Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
[25] Kumar, T.S., Jung, S.J., Koenig, S.: A Tree-Based Algorithm for Construction Robots. In: International Conference on Automated Planning and Scheduling (2014)