Relational Reinforcement Learning for Lifelong Multi-Agent Path Planning

Evan Czyzycki*, Julian de Gortari Briseno†, Pehuen Moure‡, Amirali Omidfar§ and Ankit Ranjan¶
* University of California, Los Angeles
Email: eczy@cs.ucla.edu
†University of California, Los Angeles
Email: julian700@g.ucla.edu
‡University of California, Los Angeles
Email: pehuen@g.ucla.edu
§ University of California, Los Angeles
Email: omidfar@ucla.edu
¶ University of California, Los Angeles
Email: ankitranjan@g.ucla.edu

***Abstract*—Multi-agent path finding (MAPF) can be used to streamline pickup and delivery tasks in warehouses through optimal planning of agents. The concept of MAPF in a two dimensional grid-world has been tackled for agents working to collectively reach desired positions as fast as possible and has had promising research. Fully decentralized planners trained using reinforcement learning have seen considerable success in recent years through two implementations, PRIMAL and MAPPER. One unaddressed issue present in these algorithms is difficulty in scalability and generalization. This work presents the implementation of relational reinforcement learning on top of previously successful algorithms to add ability to generalize learning to similar grid-like environments.**

I. Introduction

Amazon has developed a robot-driven package sorting center where a set of holonomic robots navigate a warehouse environment in order to move package from arbitrary locations to other arbitrary locations [1]. The problem of programming agents to move objects from certain positions to others in a continuous online fashion is often called multi-agent pickup and delivery (MAPD). The computational aspect of this problem wherein these robots must determine the optimal path from their positions to other arbitrary goal positions while obeying environment constraints is called multi-agent pathfinding (MAPF). While much active research has been done in this space, many approaches to this problem rely upon exact pathfinding methods such A^* which quickly becomes prohibitively expensive as the state space and number of agents increases. By using more powerful reinforcement learning paradigms such as relational reinforcement learning [11], it may be possible to achieve useful run-time performance while avoiding the expensive computation of exact methods.

Our code can be found at the following GitHub repository: https://github.com/eczy/warehouse-mapf.

[1]https://www.wired.com/story/amazon-warehouse-robots

II. Motivation

The increased demand of MAPF systems in warehouse robotics has shifted the focus to more recent methods that deal with fully decentralized policies as a way to reduce the complexity of the system and increase the scalability. However a remaining concern is the necessity of high correlation between training and test environments. Methods such as PRIMAL [1] have clearly shown that agents trained in 70 x 70 grid environments perform quite poorly when tested in grids of greater size. In this study we explore several approaches which may address this limitation. We hope to emphasize a new aspect of learning through which agents can gain a more general understanding of their environments and effectively apply it to unknown environments that they have never visited before. Specfically, we investigate the addition of relational learning to these MAPF approaches with the expectation that the distributed agents will learn higher order representations of their environments and therefore be more capable of generalizing to new environments. Adding this module to previously successful MAPF algorithms could show an increase in performance via generalization, and consequently scalability.

III. Related Work

In this section we discuss previous work in the MAPF field to present an understanding of the current state of the field and areas for improvement. First we discuss centralized multi-agent path finders, followed by decoupled multi-agent path finders and lastly relational reinforcement learning.

A. Centralized Multi-agent Path Planners

In the standard MAPF formulation the solvers is tasked with finding a feasible set of solutions for a given set of agents with associated goals once. Solving the MAPF problem optimally is NP-Hard [10]. Optimal one shot MAPF solvers search over the joint space for all agents, and thus scale exponentially with the number of agents of the system [6] [5]. Centralized sub-optimal (bounded) MAPF solvers are able to dynamically increase the search space to provide reasonably feasible solution, but also exponentially in run-time with the number of agents [3]. In the lifelong multi-agent path finding (LMAPF) formulation the problem becomes prohibitive as the solver needs to be rerun frequently as different goals have different completion times. The combination of high computational costs with frequent need for re-planning has lead to exploration for decentralized path planners.

B. PRIMAL

Researchers from the University of Carnegie Melon, Commonwealth Scientific and Industrial Research Organisation (CSIRO), and University of Southern California (USC) have developed a decentralized, partially observable algorithm that uses imitation learning with reinforcement learning to solve the MAPF problem. The agents imitate an expert implementing a centralized conflict based search (CBS) [9] or centralized ODrM* (operator decoupled recursive M*) [2] and alternate with reinforcement learning that uses an asynchronous advantage actor-critic (A3C) algorithm to learn single-agent decentralized policy. A deep neural network is used to approximate the policies, which is referred to as Pathfinding via Reinforcement and Imitation Multi-Agent Learning (PRIMAL).

PRIMAL displayed success in showing that the decentralized policy learned by each agent entails some degree of cooperation with other agents, especially when adding to the mix a series of penalties for uncooperative behaviour. The addition of imitation learning also greatly speeds up the process of training and builds on previously successful planners like the above mentioned ODrM* and CBS. The learned policy is capable of being applied to any number of agents and therefore scalability is greatly improved upon as well. As a result, the agents learn to take movements to favor the whole team and not just themselves (Figure 2).

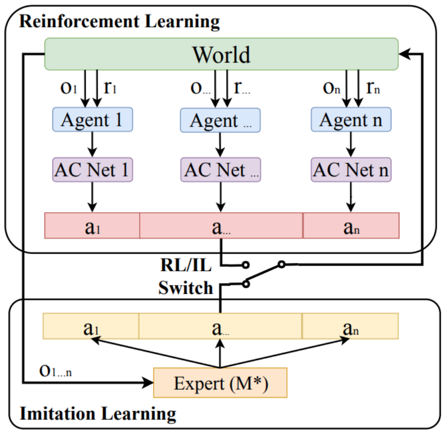

Fig. 1. Structure of PRIMAL's hybrid RL/IL [7]

An extension to PRIMAL was then developed in the form of PRIMAL2 [1], which expands the original formulation to take into account lifelong scenarios and dense structured environments. The dense structure resembles a maze and wherein for all agents to reach their targets, they must develop a highly cooperative policy. To achieve this, the observation state is altered to include information about likely paths to be taken by nearby agents (by using A*), and obstruction caused by agents and obstacles.

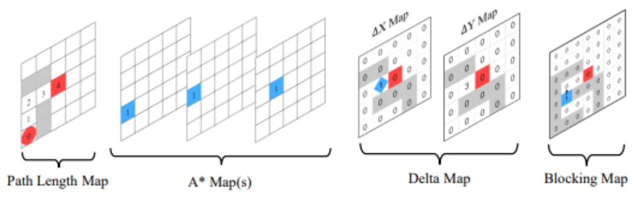

Fig. 2. Additional observation channels in PRIMAL2 [1]

C. MAPPER

Researchers from the University of Carnegie Melon and Tsinghua have developed a decentralized Multi-Agent Path Planning algorithm with Evolutionary Reinforcement learning (MAPPER) [4]. This work introduces the ideas of dynamic obstacles (cooperative and non-cooperative), the utilization of a global path planner to divide a long range planning problem into short range segments and the use of an evolutionary algorithm that eliminates bad policies and maintains good ones. The division of long range to short range segments increases the performance of reinforcement learning as the rewards are not as sparse. This algorithm utilizes a Convolutional Neural Network (CNN) with max-pooling that receives observation data in the form of multiple channels. This is then fed into the advantage actor-critic (A2C) network along with a goal position. Finally, the evolutionary training method greatly increases scalability and stability during training, especially where the number of agents grows, as the environment becomes more complicated and variance of the gradient grows exponentially (Figure 3).

Comparing this to other similar work like PRIMAL [1] shows that the training time is greatly reduced because of decentralized training. MAPPER uses decentralized training to learn decentralized policies, whereas PRIMAL uses centralized training to learn decentralized policies.

D. Relational Reinforcement Learning

Relational Reinforcement Learning (RRL) improves the efficiency, generalization capacity, and interpretability of conventional approaches through structured perception [11]. It utilizes self-attention (similar to transformer networks) to learn the higher-order relationships between entities in the environment and guide a model-free policy. It enhances baselines in terms of sample complexity, the ability to generalize to more complex scenes than experienced during training, and overall performance. A part of RRL's contribution comes from *non-local computations* using a shared function and *iterative computation*. As indicated in [11], an agent computing pairwise interactions between entities, independent of their spatial proximity, using a shared function, will be a better

Algorithm 1 Multi-Agent Evolutionary Training Approach

```
Require: Agents number $N$; discount factor $\gamma$; evolution interval $K$; evolution rate $\eta$;
1: Initialize agents' model weights $\Theta = \{\Theta_1, ..., \Theta_N\}$
2: repeat
3:   Set accumulated reward $R_1^{(k)}, ..., R_N^{(k)} = 0$
4:   // update model parameters via A2C algorithm
5:   for $k = 1, ..., K$ do
6:     for each agent $i$ do
7:       Executing the current policy $\pi_{\Theta_i}$ for $T$ timesteps, collecting action, observation and reward $\{a_i^t, o_i^t, r_i^t\}$, where $t \in [0, T]$
8:       Compute return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$
9:       Estimate advantage $\hat{A}_i = R - V^{\pi_{\Theta_i}}(o_i)$
10:      Compute gradients $\nabla_{\Theta_i} J = \mathbb{E}[\nabla_{\Theta_i} \log \pi_{\Theta_i} \hat{A}_i]$
11:      Update $\Theta_i$ based on gradients $\nabla_{\Theta_i} J$
12:    end for
13:    $R_i^{(k)} = R_i^{(k)} + R_i$
14:  end for
15:  Normalize accumulated reward to get $\bar{R}_1^{(k)}, ..., \bar{R}_N^{(k)}$
16:  Find maximum reward $\bar{R}_j^{(k)}$ with agent index $j$
17:  // Evolutionary selection
18:  for each agent $i$ do
19:    Sample $m$ from uniform distribution between $[0, 1]$
20:    Compute evolution probability $p_i = 1 - \frac{\exp(\eta \bar{R}_i^{(k)})}{\exp(\eta \bar{R}_j^{(k)})}$
21:    if $m < p_i$ then
22:      $\Theta_i \leftarrow \Theta_j$
23:    end if
24:  end for
25: until converged
```

Fig. 3. MAPPER Algorithm (Evolutionary Elimination)

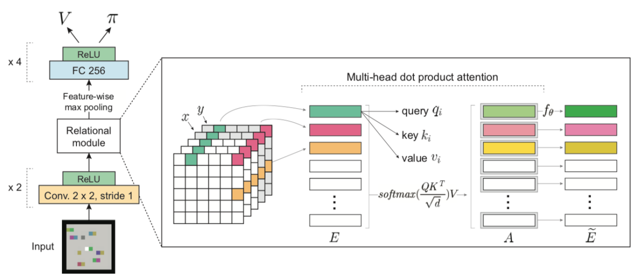

Fig. 4. An MHDPA step used in Relational Deep Reinforcement Learning

option for learning important relations than an agent that only computes local interactions. To deploy and compute non-local interactions, in [11] they chose a computationally efficient attention mechanism, assuming there exists a set of entities for which interactions must be computed, and Multi-Head Dot-Product Attention (MHDPA), or self-attention was utilized as the operation that computes interactions between these entities (Figure 4).

IV. Formulation

A. State Space

The environment can be represented as a matrix, since the robots travel in a equally distributed grid pattern on the floor. Let G represent the matrix in which the robots move. Each point in G can have a corresponding "type" for every point on the floor: $\{p, d, t, o\}$ corresponding to pickup point, drop off point, obstacle, and travel point correspondingly. G will be represented as a matrix. An example of $G_{(3,3)}$, with a pickup on the top left corner and a drop off on the bottom right corner can be seen below:

$$G = \begin{bmatrix} p & t & t \\ t & o & t \\ t & t & d \end{bmatrix}$$

Using this environment setup we can define the state space to be the representation of all the drives on the floor and their current location. Let s_t be the state of the floor at any given time t; it will also be represented as a matrix of size $2 * \mathbb{R}^{(|G|,|G|)}$. For s_t, let the first set of inputs be denoted as S_1. All S_1 can be defined with an integer $z \in Z$ being used if there is a robot z at a given (i, j), a -1 if there is an obstacle, and a 0 being used otherwise. Thus the full S_1 can be defined as:

$$S_1 = \left\{ \begin{bmatrix} z & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & z & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & z \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \cdots \right\}$$

Let the second entry of the state matrix be denoted as S_2. This sub-state represents each robot's current goal locations, thus it will refer to an index i, j for all $|G > 0|$ and a goal destination. An example S_2 for a single drive who's goal is to move to bottom right can be seen below:

$$S_2 = \left\{ \cdots, \begin{bmatrix} [3,3] & None & None \\ None & None & None \\ None & None & None \end{bmatrix}, \cdots, \right\}$$

Then the full state can be represented as:

$$s_t = \{[S_{it}, S_{jt}] \forall S_{it} \in S_1, S_{jt} \in S_2\}$$

B. Observation Space

The observation space consists of the information each agent receives at each time step. Since this is a fully observable grid world, each agent observes the entire state including all agents, obstacles, and goals. The representation shown here allows for compatibility with existing work in the MAPF field [7] [1] [4]. The observation space consists of multiple channels from the point of view of the agent z, that describe the position of agents or position of goals. Up to 5 channels may be used in total at time t for each agent z:

$$O_{zt} = [O_{zt1}, O_{zt2}, O_{zt3}, O_{zt4}, O_{zt5}]$$

O_{zt1} = Agent z's channel: NxN matrix showing the coordinate of agent z at time step t.

O_{zt2} = Other agents channel from agent z's point of view: NxN matrix showing the coordinate of other agents from agent z's point of view at time step t.

O_{zt3} = Goal channel: NxN matrix showing the coordinate of the goal of agent z at time step t.

O_{zt3} = Other agents goal channel: NxN matrix showing the coordinates of the goals of the other agents at time step t.

O_{zt5} = Obstacle map channel: NxN matrix showing the coordinates of static obstacles at time t. This is mainly used for compatibility reasons.

These channels are the input to a CNN layer which then feeds into the rest of the network architecture.

C. Action Space

At every given time step each of the agents can choose one of five actions, move up, down, left, right, or no action, thus the set of all possible actions can be defined as:

$$A = \{Up, Left, Down, Right, Wait\}$$

In the implementation, the action space is enumerated with, Right = 0, Up = 1, Left = 2, Down = 3, and Wait = 4. In the case that the chosen action cannot be taken due to interference from obstacles or other agents, the wait action is taken instead.

D. Transition Probabilities

In our formulation the transition probabilities for an action that would not cause a collision and are in-bounds are deterministic. For sake of brevity let us define a two helper function: 1.) $Unoccupied$, which has as input the state at $t-1$ and a_i and returns if the new location is not occupied; and 2.) $Compass$ which yields of output state s_t when applying a_i to s_{t-1}:

$$P = \{p\left(s_t|s_{t-1}, a_i\right) = \begin{cases} 1 & \text{if:} \\ & \quad \text{Unoccupied}(s_t) = \text{True}, \\ & \quad \text{Compass}(s_{t-1}, a_i) = s_t \\ 0 & \text{otherwise} \end{cases}$$

E. Rewards

Let the rewards be defined as follows, inspired by [7] and [4]:

Reward	Value
Move	-0.3
No movement off goal	-0.5
No movement on goal	0.0
Oscillation penalty	-2.0
Goal	20.0

By having a small negative reward for movement, the agent is incentivized to reach the goal quickly. This also means a greater negative reward should be associated with no movement off goal, to avoid the agent staying in its place, and an even greater penalty is used to avoid an oscillation motion whereas the agent moves one step forward and one backward, in a sequential manner. This is mild reward shaping and it is done with the intent that the agent learns faster from constant feedback, reducing training time. However, reward shaping can heavily negatively influence the discovery of novel policies. To combat this, a large reward is achieved for reaching the goal state.

V. Methods

A. Environment Setup

To see the effect of RRL on some of the existing MAPF algorithms discussed above, we first created a 2D grid simulation of the warehouse environment. To simplify the training and deployment we considered deterministic transition probabilities with a fully observable state space. We constraint the agents to non-diagonal actions. Additionally our environment has static obstacles.

Because of computational constraints, four agents were utilized in a reduced grid of 11x11 and minimal obstacles. There were a few modifications we added to each algorithm to make it compatible with our environment. For example, in the case of the PRIMAL network, we also needed to compute a unit vector pointing towards the goal of each agent based on their current state, as well as the Euclidean distance towards the same goal, so as to use this information as input to the corresponding network.

B. Relational Reinforcement Learning Architecture Design

To determine the effect of introducing relational architecture to MAPF algorithms, we compare the performance of PRIMAL both with and without relational modules on our warehouse environment. The non-relational PRIMAL architecture is identical to the architecture defined in [8]. The relational PRIMAL architecture is similar. We add a single relational module using multi-headed dot-product attention as defined in [11] before the first max-pooling layer of the original architecture. All other layers are unchanged.

The structure of this relational module as it is defined in [11] can be seen in Figure 4. The relational module receives an input from some set of feature extraction layers which is considered to be the set of entities E on which relations can be learned. This set of entities is then passed through a multi-head dot product attention (MHDPA) layer which learns the relations between these entities. Finally, a 2-layer dense network is applied in parallel to the output of this layer in order to produce updated entities $\tilde{E}$.

C. Training

Each algorithm was trained on an NVIDIA GeForce RTX 2080 SUPER with an Intel i7 processor with the parameters listed below.

1) Centralized Relational Agent: We train our single centralized RRL agent using advantage actor critic (A2C) for $2e7$ iterations and an initial learning rate of $7e-4$. We use a discount factor of 0.99 and an episode length of 200.

2) Relational and Non-relational PRIMAL: We train our distributed PRIMAL agents using advantage actor critic (A2C) for $2e7$ iterations and an initial learning rate of $2e-5$. We use a discount factor of 0.95 and an episode length of 256. We set the IL/RL training split to $(20/80)$.

VI. Results

For evaluation purposes we compare the standard PRIMAL implementation to the performance with the relational module. As a baseline we compare our results to the standard bounded closed form solver in ODrM* and a single centralized RRL agent.

Due to a server crash, we were only able to retrieve results for a few different training scenarios on PRIMAL (referred to here as baseline model) and PRIMAL with RRL. The results are shown below: The outcomes of our training are presented in two categories:

1) The accumulated reward per episode
2) The number of targets (goals) reached per episode

PRIMAL Baseline Results

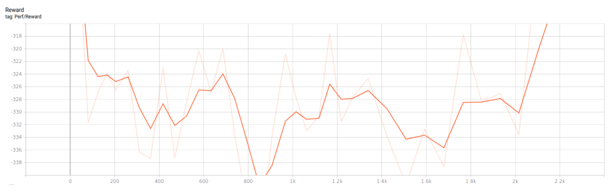

Fig. 5. Baseline PRIMAL reward per episode

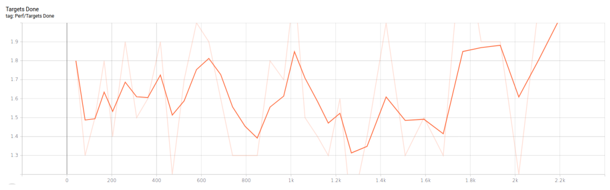

Fig. 6. Baseline PRIMAL targets reached per episode

PRIMAL with RRL Results

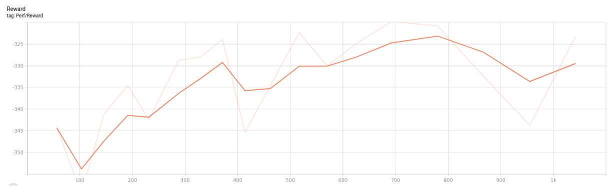

Fig. 7. PRIMAL with RRL reward per episode

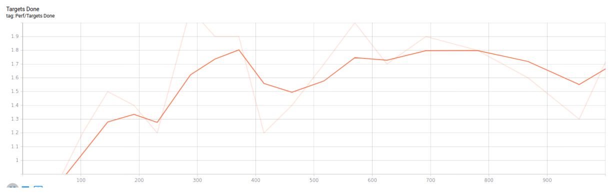

Fig. 8. PRIMAL with RRL targets reached per episode

As shown in the plots above, PRIMAL with RRL tends to follow a smoother curve in improving its performance and optimizing the policies taken. However observation is only made over very limited data, and as it is, we cannot make any conclusions about the effect of our changes to the PRIMAL algorithm. We do observe a difference between the two curves which may indicate that longer running experiments might yield interesting results.

VII. Discussion

Had our results been successfully collected, we would have first examined the training curves and cumulative reward between the relational and non-relational PRIMAL models. Then, we would have examined the zero-shot generalization of both approaches by examining the performance of both models on larger/more complex environments.

Whether we find that the relational module improved or did not improve the performance of the PRIMAL algorithm, we would have examined the relational module of the relational PRIMAL agent in order to analyze the relationships learned in the environment, similar to the analysis done in [11]. We would have hoped to find relations learned between each agent and other agents as well as between each agent and its goal state. If our results implied that the relational module did not improve performance, then this analysis could potentially show why the agent was acting non-optimally.

We would then examine the performance of each algorithm on worst-case environments. Specifically, these environments include those in which agents are most likely to collide or obstruct each other. We would have hoped to observe an improved worst case performance in the relational PRIMAL method. As above, we would examine the relational modules to determine the relationships learned in the relational agent.

Finally, we would report training and inference runtime statistics between relational and non-relational methods.

VIII. Future Work

Due to computational constraints and time constraints, our study was limited to smaller state spaces and agent counts. In the future, we hope to evaluate our approach on more powerful hardware that would allow us to experiment with larger environment sizes and larger environment counts. This might provide us a better baseline for the capability of the algorithm and the effects of the relational module at scale. One shot learning could be used to assess generalization ability, similar to the analysis found in [11]. Furthermore, we are interested in analyzing the performance of this approach in more difficult environments such as those with dynamic obstacles, non-deterministic state transitions, noisy sensors, etc. Similarly, another avenue for future work is to consider a continuous space warehouse and implementing our approach with this fashion. This would reduce the dependency of our approach on a discretized environment, which might be too difficult for many practical approaches. Finally, we are interested in evaluating the benefits of relational reinforcement learning in fully distributed systems. PRIMAL and MAPPER are only semi-distributed since they both rely on a central planner. Would relational reinforcement learning improve the performance of the distributed portions of these systems enough to omit the central planner?

References

[1] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal2: Pathfinding via reinforcement and imitation multi-agent learning – lifelong, 2020.

[2] Cornelia Ferner, G. Wagner, and H. Choset. Odrm* optimal multirobot path planning in low dimensional search spaces. *2013 IEEE International Conference on Robotics and Automation*, pages 3854–3859, 2013.

[3] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses, 2020.

[4] Zuxin Liu, Baiming Chen, Hongyi Zhou, Guru Koushik, Martial Hebert, and Ding Zhao. Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments, 2020.

[5] Hang Ma, Craig Tovey, Guni Sharon, T Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. 02 2016.

[6] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments, 2017.

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

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

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

[10] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, page 1443–1449. AAAI Press, 2013.

[11] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Relational deep reinforcement learning, 2018.