Time-Constrained Multi-Agent Path Finding in Non-Lattice Graphs with Deep Reinforcement Learning

Marijn van Knippenberg M.S.V.KNIPPENBERG@TUE.NL
Mike Holenderski M.HOLENDERSKI@TUE.NL
Vlado Menkovski V.MENKOVSKI@TUE.NL
Eindhoven University of Technology

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Multi-Agent Path Finding (MAPF) is a routing problem in which multiple agents need to each find a lowest-cost collection of routes in a graph that avoids collisions between agents. This problem occurs frequently in the domain of logistics, for example in the routing of trains in shunting yards, airplanes at airports, and picking robots in automated warehouses. A solution is presented for the MAPF problem in which agents operate on an arbitrary directed graph, rather than the commonly assumed grid world, which extends support to use cases where the environment cannot be easily modeled in a grid shape. Furthermore, constraints are introduced on the start and end times of the routing tasks, which is vital in MAPF problems that are part of larger logistics systems. A Reinforcement Learning-based (RL) approach is proposed to learn a local routing policy for an agent in a manner that relieves the need for manually designing heuristics. It relies on a Graph Convolutional Network to handle arbitrary graphs. Both single-agent and multi-agent RL approaches are presented, showing how a multi-agent setup can reduce training time by exploiting the similarities in agent properties and local graph topologies.

Keywords: Deep Reinforcement Learning; Multi-Agent Reinforcement Learning; Graph Neural Networks; Routing; Multi-Agent Path Finding; Optimization

1. Introduction

Multi-Agent Path Finding (MAPF) is a classic routing problem in which the goal is to find shortest paths for a set of agents such that none of the agents collide, see Stern et al. (2019); Zhang et al. (2019). The problem has many real-life applications in logistics, especially in situations where movement space is very restricted. Examples include warehouse robots, airport towing, robotics, and train routing. In each of these scenarios, the objective is for agents to reach their intended destination most cost-effectively while avoiding collisions. Increasing the size of the graph-based environment in which the agents move makes solving the problem more computationally expensive, and as the number of agents increases, so does the difficulty of finding a valid solution, due to the increased chance of conflicts occurring. Existing approaches focus on grid-based environments, and as the problem complexity grows, they either become very slow or drop dramatically in solution quality. In this work, one of the goals is to find a middle ground between these two situations. The contribution of this work is two-fold: (i) the formulation of a MAPF problem is extended with time constraints, and (ii) a Deep Reinforcement Learning-based approach is proposed for solving the problem in graph-based environments that does not rely on a grid-world structure. In many practical cases, MAPF is a component of a larger logistics problem. To work in step with the in- and out-flow of the other parts of this system, it is important that the MAPF component can match the other components' constraints, such as different operating times. Introducing start times and deadlines for each agent can, for example, account for the flow of trains through a part of a larger railway network. Taking a Machine Learning-based approach, we invest more resources into the offline component of the solving process, training a model that can then be re-used on any problem instance. Since this model generates a single solution for each problem, such an approach will generally be faster than iterative methods, especially as the problem complexity increases. This is beneficial, as in practical cases, it is the online setting that is more time-constrained: there is enough time to prepare a solving method, but the actual (re-)scheduling often has to be performed under tight time constraints at regular intervals. Instead of requiring human involvement in the design of a heuristic, in the proposed approach, agents use synthetic data to learn how to navigate the environment, and avoid collision with other agents, coming up with their own heuristic. The data is generated based on practical use cases. A single policy is learned that is shared across all agents. Agents take into account the locations and goals of other agents in their vicinity. To handle arbitrarily large graph environments, agents operate on the knowledge of a locally observed sub-graph, reducing the amount of data that is used by each agent. We see that this approach matches state-of-the-art methods for the classic MAPF problem and that it scales well for the time-constrained MAPF problem. The data-driven nature of this approach also means that models can be specialized to particular industry use cases, simply by constraining the training data.

2. Background

2.1. Multi-Agent Path Finding

Multi-Agent Path Finding (MAPF) is a traditional routing problem that can be viewed as multiple instances of the shortest path problem being placed in the same environment. Assuming that at most one agent can occupy a spot in that environment at a time, agents may collide as they move around. The problem is known to be NP-hard, as described in LaValle (2006). Solutions to the problem can be either centralized, where a single planning entity controls all agents, or decentralized, where agents act independently and optionally communicate to avoid collisions, for example in van den Berg et al. (2009); Čáp et al. (2013); Cui et al. (2012); Leroy et al. (1999). Some solvers build a solution iteratively, routing all agents one time step at a time, while others first route all agents to their respective goals in an optimistic manner, and then look back to correct the planned routes to eliminate collisions, as demonstrated in Wagner and Choset (2015). More recently, hybrid approaches have been proposed that combine the centralized and decentralized points of view, such as Conflict-Based Search (CBS) in Barer et al. (2014); Sharon et al. (2012). Another recent trend is the application of Deep Learning to some parts of the solving process, such as performing algorithm selection in Kaduri et al. (2020).

While searching for a solution, the objective is usually either the makespan of the schedule (i.e. the duration until all agents have reached their goal) or the sum of costs of the schedule (i.e. sum of all action costs of all agents). If the shortest paths of the agents do not collide at any point in time, the shortest paths make up the optimal schedule. If the shortest paths do collide, one or more agents will have to diverge from the shortest path to accommodate other agents. Diverging can mean either selecting a different path towards the goal or waiting at a particular point in the path to allow another agent to pass first.

Previous work on applying time constraints to the MAPF problem exists, but limits itself to only defining a deadline time for each agent, in Ma et al. (2018). This view of the MAPF problem is expanded here by not only constraining each agent to a finishing time but also by defining a starting time before which an agent is not able to take any action.

2.2. Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is the application of Reinforcement Learning principles in situations where multiple, autonomous agents interact with the same environment, and with each other, see Buşoniu et al. (2010). By splitting the problem among multiple agents, the complexity of the problem can be reduced. Care has to be taken though, as taking a multi-agent approach can quickly cause the state-action space to explode combinatorially. A much-studied approach to this dilemma is to have each agent learn its own decentralized policy. This policy should then encompass some form of co-operational behavior with other agents. This can be done by employing centralized learning, such as predicting agents' actions, but this scales poorly as the number of agents increases, Lowe et al. (2017). What scales better is to have the agents train on a combined reward/value function, such as in Foerster et al. (2016); Gupta et al. (2017). Scaling can be further improved if the agents are homogeneous. In this case, parameters can be shared among agents Gupta et al. (2017). A centralized learning component can also help performance in partially observable systems, which in that case allows agents to share information about their local environment, as demonstrated in Foerster et al. (2016); Gupta et al. (2017).

Some studies include an explicit communication system between agents, where local observations and or agent output is shared among some or all agents, for example in Foerster et al. (2016); Lowe et al. (2017). Although this may improve final performance as more information is available to each agent, not only of the environment but also of the intentions of other agents, it comes with significant computational overhead. Here we only consider a multi-agent RL system with implicit communication between agents: agents may observe each other's location and properties, but this is provided as part of the environment and is not explicitly communicated.

MARL has been applied to MAPF problems in a limited sense, namely to the classic MAPF problem in grid worlds in Sartoretti et al. (2019). While this approach improves on non-ML methods, moving to non-grid worlds and including time constraints poses new challenges. First, if the world has a grid-like structure, Convolutional Neural Network (CNN) layers can easily be applied to exploit the locality information in the grid. Without the regular structure of grid worlds, environments become much more difficult to parse, and extracting local observations from the environment becomes less straightforward. Second, there are no existing expert systems that can be leveraged to kick-start learning through the application of Imitation Learning, as was done in Sartoretti et al. (2019).

2.3. Graph Convolutional Networks

Graph Convolutional Networks (GCN) as described in Kipf and Welling (2017) are a recent innovation in Deep Learning that allows Neural Networks to process graph-based data. By using GCNs, a graph can be completely embedded into a latent space, including any node and edge attributes. This is important, as the graphs that will be considered in this work have both node and edge attributes. This embedding can then be further processed by other neural components. The application of this type of neural architecture has been applied to MAPF problems before but has been limited to lattice graphs in Jiang et al. (2020), which reduces the environments to grid worlds.

3. Problem Formulation

The problem addressed in this paper is formulated as a Multi-Agent Path Finding problem with time constraints.

The environment consists of a directed graph $G = (V, E)$ containing n nodes (see also Figure 1). For the sake of ensuring that agents can always reach their goal location regardless of their starting location, environment graphs are assumed to be strongly connected, i.e. each node is reachable from all other nodes. Agents will travel along the edges of the graph and be stationed at its nodes. Edges and nodes can only be used by a single agent at any one time step. An edge between nodes v and v' contains a non-negative cost value $c(v, v')$, which indicates the cost for an agent to travel along that edge. Each node may contain attributes that limit the actions that agents can take when located at that node. Notably, each node contains a $canWait$ attribute, which determines whether agents are allowed to wait at that particular node, or if they have to keep moving.

There are k agents, each of which has a job associated with it. A job of agent i is defined by four values: start time t_i^s, deadline time t_i^d, starting node v_i^s, and goal node v_i^g. The goal of each agent is to move from the start node to the goal node within the time limits set by the start time and the deadline time, at the lowest possible cost. Time is considered to be discrete, with all agents executing an action simultaneously at each time step.

At each time step, agents choose to perform either a “wait” or a “move” action. A move action relocates an agent from its current node to one of the node’s neighbors. A “wait” action keeps the agent at its current node. Each of these actions has a cost $c(a) = c_{v,v'}$ associated with it. This cost is determined by the cost of the edge that is moved along in case of a movement action (moving from v to v', where $v \neq v'$), or by the cost associated with waiting in the case of a wait action ($v = v'$).

A sequence of actions by a single agent i is called a plan $\pi_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,m})$. A plan is considered valid if after executing the plan the agent’s location is its goal node, and if it reaches the goal node before its deadline expires (in case of a hard deadline constraint). A schedule S consists of a plan for each agent in the instance. A schedule is valid if all plans that it contains are valid, and if its execution does not cause any conflicts. A conflict can be either a node collision or an edge collision. In a node collision, an agent moves to a node that is already being occupied by another agent that chooses to wait, or two agents move to the same node in the same time step. In an edge collision, two agents move along the same bi-directional edge at the same time step, but in opposite direction. Since neither nodes nor edges are considered shareable resources, the presence of any of these collisions renders a schedule invalid. Given a problem instance, a solution is comprised of a valid schedule.

The overall objective is to minimize the combined action costs of all agents in the instance. If deadline times are considered a strict constraint, the objective function is the same as the classical MAPF objective

$$G = \min_{S} \sum_{\pi \in S} \sum_{a \in \pi} c(a) \tag{1}$$

noting again that a plan is only valid if it can be executed within the time constraints set by the job. In case the deadline times are instead considered a soft constraint, missing a deadline can be considered to degrade the overall quality of a schedule

$$G = \min_{S} \sum_{\pi \in S} \sum_{a \in \pi} c(a) + w \sum_{\pi \in S} max(0, t_{\pi,actual} - t_{\pi,deadline}) \tag{2}$$

where w is a positive weight that determines the impact of missing deadlines. The makespan, the shortest time for all agents to complete their jobs, is not considered here, as jobs have different starting times.

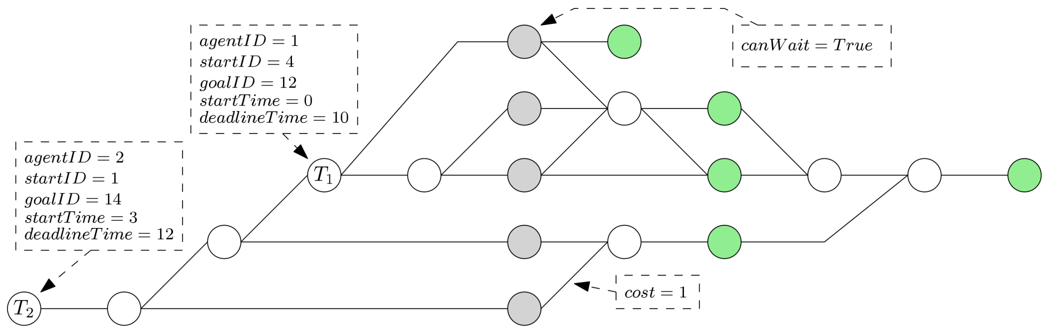

Figure 1: Small example of a problem instance. Gray nodes indicate nodes where agents can wait. Green nodes are possible goal nodes for the random generation of jobs. All edges in this case are bi-directional.

4. Method

The approach to solving the problem introduced in the previous section is based on Reinforcement Learning (RL), as it has several desirable properties in this case. It is directly aimed at tackling problems that contain a decision-making process, and the problem of assigning scores to specific actions within that decision-making process. Furthermore, RL allows for relatively easy implementation of the environment. It also allows for an arbitrarily detailed environment, which can be important for future applications of the method and any variants. Multi-Agent Reinforcement Learning (MARL) methods already exist, which are an attractive option, as the problem can intuitively be split into a multi-agent RL setup where each agent is responsible for a job. Finally, there is the promise of scaling efficiently to larger problems through the application of Deep Reinforcement Learning (DRL).

4.1. Reinforcement Learning Formulation

To fit the problem into an RL context, we specify a state space, an action space, and a set of rewards.

State The global state consists of two parts: a strongly connected, directed, weighted graph, possibly with cycles, and a set of jobs (see also Figure 1). The graph represents the locations in the problem instance and all movement options between them. Each node contains a node ID and any additional attributes that may affect agent action choices. Each edge contains a non-negative cost value. Each job represents an instance of a routing job and consists of an agent ID, a starting location, a goal location, a starting time, and a deadline time. This information is embedded in the node that the agent is currently occupying, becoming attributes of the node. When an agent moves to another node, the agent's information moves with it. Start times and deadline times are tracked in a countdown fashion, i.e. they are decremented for each agent at each time step until they reach zero. This relieves the need for keeping track of a global state timer and providing it as part of the input. Because agents have different starting times, they enter the environment at different times. To prevent an agent from appearing at a node that is already being occupied by another agent, and immediately generating a collision, a dummy node is added for each agent, which is connected to its actual starting node. These dummy nodes only have a single outgoing edge which brings the agent into the actual problem environment.

The global state is not directly used to train the agent. Instead, at each time step and for each agent, a local observation is extracted from the global state. This observation is centered on the agent's current location and its size is determined by a global depth parameter d (see also Figure 2). The depth determines how far away from the agent's location other nodes can be while still being included in the observation (see Figure 2). Each node in the observation receives an additional attribute, containing the shortest distance from that node to the goal node. These values need only be pre-computed once for every problem instance. Information about other agents that happen to be inside the scope of the observation is included in the properties of the nodes.

Intuitively, the information that is available close to an agent is more likely to have a bigger impact on solution quality than information that is further away. By reducing the size of the state that is passed to the learning model, training and inference become cheaper, and scaling up the graph size no longer increases the amount of data that is presented to the RL agent at each time step.

A major advantage of a local observation is that under the assumption of a fixed depth, the method can be generalized to arbitrary graph sizes. Instead of having to accommodate the feeding of environments of different sizes into a model, a local representation with a fixed maximum size can be obtained from any environment. The natural disadvantage is that reducing the visible state means that there is less information available, which may impact the quality of the policy (see T_1's position in Figure 2, for example). Therefore, the depth parameter has to be chosen carefully.

Actions To manage the complexity of the problem, a small action space is preferable and the same action space should be shared among all agents, and across problem instances. Therefore, the action space is defined in terms of local movement options. As with classic

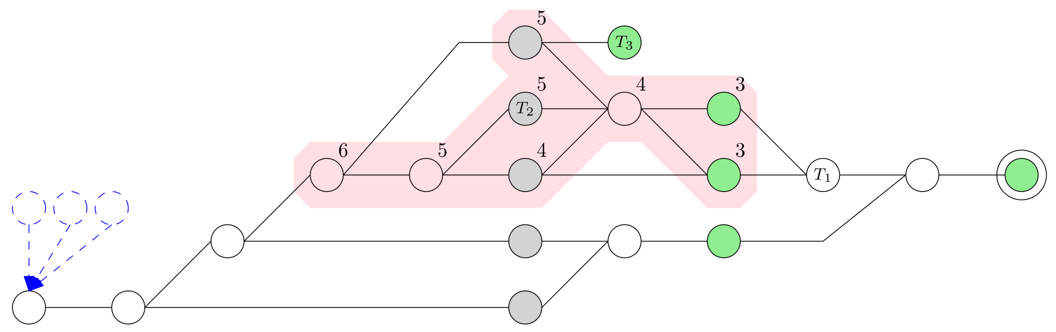

Figure 2: Example of a small environment that could represent a small train shunting yard, with the local observation for agent T_2 (which could represent a train), and depth $d = 2$. The numbers indicate the distance from that particular node to the goal node. All edges are bi-directional. The dashed nodes and edges on the left indicate dummy nodes. They are never included in any local observation. Note that with this limited depth, and with T_2's goal being the far-right node, the exclusion of T_1 location from T_2's local observation may become problematic in the future.

MAPF formulations, the available actions include movement actions across an adjacent, outgoing edge, and a "wait" action.

Rewards The reward function mostly consists of penalties for the various actions (see also Table 1). Movement incurs the lowest penalty. This is an incentive for the agents to move to their goal in the most direct possible path. Waiting is more heavily penalized because we want to incentivize the agents to keep moving if possible. Causing a collision incurs the heaviest penalty, and will also terminate the episode prematurely. If an agent chooses a move action that is not valid, i.e. a move action outside the range of outgoing edges of its current location, the action is treated as a "wait" action, and it is penalized accordingly. Positive rewards are only applied whenever an agent is at its goal node. Missing a deadline can be a cause to either terminate the episode prematurely (hard constraint) or to apply an additional penalty at each time step as long as the agent has not yet reached its goal (soft constraint).

An episode ends when either all agents have reached their goal, when a conflict has occurred which renders the current solution invalid, or when a predefined time limit has been reached (to prevent the algorithm from running indefinitely due to some locked state).

4.2. Model

The proposed approach is based on an RL framework known as PRIMAL by Sartoretti et al. (2019).

The original PRIMAL framework is limited to lattice graphs and requires a separate model input that describes the agent's job. In the proposed approach the Convolutional Neural Network that is used for processing the lattice graph is replaced by a Graph Convolutional Network (GCN), proposed by Kipf and Welling (2017), to allow processing arbitrary

Action	**Reward**
Move	-0.1
Wait (off-goal)	-0.2
Collision	-2
Wait (on-goal)	0.1
Complete episode	10
Missing deadline (strong)	-2
Missing deadline (weak)	-0.1

Table 1: Overview of rewards. The penalty for missing a deadline is applied on top of any other rewards. Reward values were empirically established, based on common practices.

graphs (see Figure 3). By including all agent information directly in the observation that is extracted from the environment for each agent at each time step, the separate input describing an agent's job can be avoided. To accommodate the usage of local observations, additional changes have to be made.

When taking local observations as input, an agent's goal node may fall outside of the local observations, which would mean that an agent cannot determine in which direction it has to travel to move closer to its goal. In grid worlds, this can be relieved by including the agent's and goal's coordinates as a separate input to the network. But since arbitrary graphs are considered here, such a coordinate system cannot easily be imposed. Instead, included in each local observation is the distance of each node to the goal node. These are embedded in the graph as node properties. This information shows the agent in which general direction its goal lies.

The neural model for a single agent takes at each time step as input the local observation graph with all of its node, edge, and agent properties. This is then processed by a GCN component to produce an embedding of the local observation. This embedding is further processed by a series of fully connected layers with ReLU activation functions, which finally determine the next action to be taken through the application of the softmax function to the output of the last fully connected layer. The model is essentially a Deep Q-Learning Network (DQN), as from Mnih et al. (2013), approximating the value of state-action pairs. To help stabilize learning, we employ both experience replay and a target network.

The neural model is re-used among all agents, as they are homogeneous. Network parameters are updated at regular intervals, or when an episode ends. The maximum number of time steps in an episode is twice the number of nodes in the environment graph. At each time step, an action is generated for each of the agents. As these may cause a conflict, care has to be taken in the order of application of the actions to the global state. There are situations in which the order of action processing determines whether a collision takes place. For example, if two neighboring agents move in the same direction, the agent that is "in front" has to move first to avoid a collision. Actions are ordered in such a way that collisions are avoided as much as possible, by checking the destination of all actions. If there are no conflicts at the destinations, then it is possible to order the actions in such a way that avoids collisions.

The graph processing portion of the model was implemented using the Deep Graph Library (DGL)[1] library, with the number of graph convolutional layers and fully connected layers depending on the particular experiment. Non-linearity between fully-connected layers was applied through ReLU activations. The overall model was implemented in PyTorch[2]. The optimizer used for all experiments was Adam.

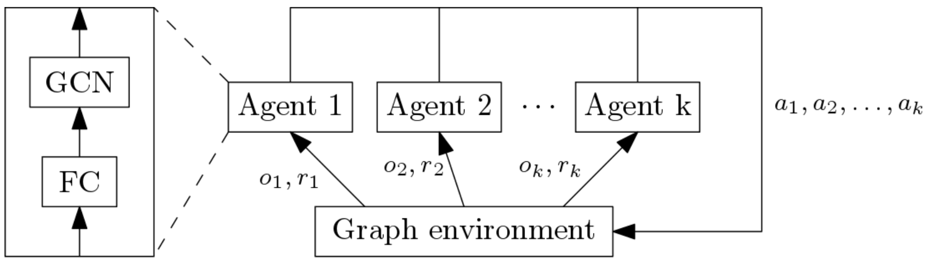

Figure 3: Multi-Agent Reinforcement Learning model. At each time step, each agent is presented with a local observation (o), and a reward for its previous action (r). The new local observation is processed by the neural network to produce an action (a) for the next time step. Note that the weights in the agent model are shared among all agents.

5. Experiments

For experiments, we consider two sets of environments. The first is a series of classic MAPF problems for which benchmarks exist. These allow for the comparison of the proposed method to existing MAPF solvers. Although the classic problem formulation is not the actual target of this method, strong performance would boost confidence in this new method, and place it among existing solutions. Note that all of these environments are grid worlds. For our purposes, these grid worlds are converted to directed graphs by interpreting each open cell as a node. Neighboring open cells are connected by bi-directional edges (see Figure 5).

The second set of environments consists of randomly generated graph environments that are inspired by train shunting yards (see Figure 4). Such yards pose an excellent practical application of the method, as they are very restricted in terms of movement, and usually deal with large numbers of agents with differing operational time windows. Each instance consists of two uni-directional lanes in the center of the graph. Other bi-directional lanes are connected to these central lanes and each other with the addition of diagonal edges. The layout is chosen so the graph is easy to plot. Nodes that have one or two neighbors are considered appropriate starting or goal locations, except for the central lanes (their endpoint nodes are included). The generation of these instances is parameterized by the width and height of the graph (in terms of the number of nodes), and the number of diagonal edges that connect the lanes. There is always at least one diagonal edge between two adjacent lanes to

1. https://www.dgl.ai/
2. https://pytorch.org/

ensure connectivity. The particular configurations that are of most interest are those that generate problems that are neither too simple nor too difficult. If, for example, the inter-job start delay is quite large, the problem may quickly devolve into multiple sequential shortest path problems. On the other hand, if the number of agents is quite large, the problem may quickly become impossible to solve due to gridlock.

All experiments were executed on an Intel Xeon E5-2698v4 @ 2.2GHz and an NVidia Tesla V100 with 16GB of RAM. Code, data, and random seeds are provided separately.

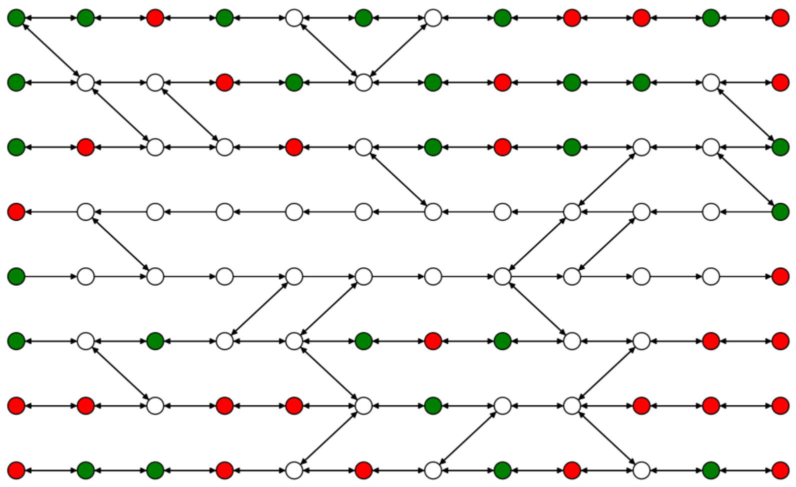

Figure 4: Example of a small randomly generated yard-based environment. Running through the center from left to right is a pair of “main lines” which do not contain any starting or goal locations, except at their ends. Labeled in green are available starting locations, and in red are available goal locations. Note that some edges are not bi-directional and that each node tends to have a lower degree than a node in a grid world.

6. Results

Presented below are experiments and related results that investigate the performance of the proposed method, as compared to existing solving methods, both Machine Learning-based ones and non-Machine Learning-based ones.

6.1. Multi-Agent Path Finding Benchmark

Based on the recent overview paper Stern et al. (2019), we apply the proposed approach to several classic MAPF problems for which benchmarks exist[3] (see Table 2). To construct a fair testing procedure, we need a method in which we can account for the existence of job starting times and deadlines. We can eliminate the influence of different starting times by converting dummy nodes into chains of dummy nodes that have a length equal to the

3. https://movingai.com/benchmarks/mapf.html

agent's starting time. While the starting time has not been reached yet, an agent will travel along the chain of dummy nodes, and only enter the actual problem environment when its start time is reached. The impact of deadlines is eliminated as much as possible by setting deadlines that equal the maximum episode duration. The grid worlds from the benchmark were converted to lattice graphs, leaving holes in the lattice where there are obstacles in the grid world.

Included in the comparison are A* with improvements that have been shown to work well for solving MAPF instances, from Goldenberg et al. (2014), and two state-of-the-art methods: Improved Conflict-Based Search(ICBS) from Boyarski et al. (2015), and Increasing Cost Tree Search (ICTS) from Sharon et al. (2013). We also make a comparison with the method that is closest in structure, from Sartoretti et al. (2019). The main differences here are that our approach allows arbitrary directed graphs as input through the use of a Graph Convolutional component, and supports the inclusion of starting times and deadline times. Although this means that the performance on time-constrained MAPF problems cannot be compared, we can still take a look at the difference in performance that the Graph Convolutional component may cause. All algorithms were run with a 30-second timeout. The offline components, such as model training, are not included in this timeout, since it is the online component of the solving process that is practically time-constrained.

Map	Size	A*	ICBS	ICTS	PRIMAL	RL
Berlin_1_256	256x256	577	892	903	873	888
Boston_0_256	256x256	484	718	720	659	701
Paris_01_256	256x256	534	805	799	773	780
brc202d	481x530	198	252	265	242	240
den312d	81x65	467	577	568	560	558
lak303d	194x194	233	377	382	425	403
random-32-32-10	32x32	732	1027	1035	1107	1076
random-32-32-20	32x32	589	862	863	887	865
random-64-64-10	64x64	225	450	492	477	461
random-64-64-20	64x64	618	1078	1112	1045	1127
room-32-32-4	32x32	385	469	480	592	472
room-64-64-16	64x64	542	629	624	686	643
room-64-64-8	64x64	295	360	345	433	381

Table 2: Number of solved instances for the MAPF benchmark for from Stern et al. (2019).

We inspect the performance on problems with deadlines through comparison with the work that introduced MAPF-DL, Ma et al. (2018). As above, different starting times are accounted for by adding dummy nodes. This time, however, the deadline times are not set equal to the episode length, which allows them to impact the solving process and solution quality.

The results show that the proposed method's performance is competitive with existing solvers. PRIMAL performs significantly better on the "room" maps, likely because its training focused heavily on resolving bottlenecks in otherwise somewhat open environments, which are a hallmark feature of these maps. Results for the proposed RL approach can rea-

Map	Size	CBS-DL	RL
Berlin_1_256	256x256	902	879
Boston_0_256	256x256	699	703
brc202d	481x530	247	238
den312d	81x65	580	576
random-64-64-10	64x64	476	474
random-64-64-20	64x64	1085	1113
room-64-64-16	64x64	645	641
room-64-64-8	64x64	375	364

Table 3: Number of solved instances for the MAPF benchmark for CBS-DL, Ma et al. (2018), and this work.

sonably be expected to improve if Imitation Learning is applied, as in PRIMAL. Sartoretti et al. (2019).

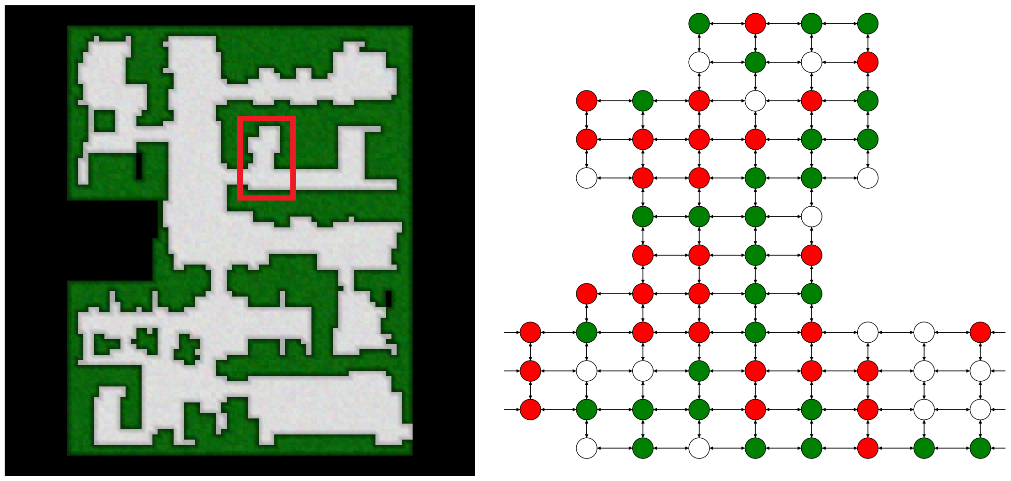

Figure 5: Example of the conversion of the `den312d` grid world to a directed graph. Each open cell is converted to a node that is connected to neighboring open cells by bi-directional edges. Labeled in green are available starting locations, and in red are available goal locations.

6.2. Multi-Agent Reinforcement Learning

Next, we test the performance of the multi-agent RL system against a similar single-agent RL system (see Table 4). In the single-agent case, the input to the model is the complete graph, including all agents, and the output is a vector for each agent that determines that agent's decision. For this and the following experiment, we use the train shunting yard-inspired environments. The size of the environments is set at 64 by 64 nodes when plotted out into a grid.

	Single		Multi	
Nodes	32 agents	64 agents	32 agents	64 agents
128	49.6%	34.7%	46.3%	32.3%
256	57.7%	54.2%	65.0%	57.4%
1024	29.2%	21.9%	67.3%	64.5%
2048	13.9%	7.6%	63.3%	58.2%

Table 4: Comparison of single-agent and multi-agent RL performance. Shown are the percentages of solved instances for given graph sizes and the number of agents. The training was limited to 24 hours in all cases. Note that for smaller graph sizes, the performance is heavily influenced by gridlock situations.

The multi-agent RL approach scales better as the number of agents grows and the size of the environment grows. Both of these severely degrade the computational efficiency of the single-agent setup, as it has to consider the entire environment at each time step and learn a centralized policy that encompasses all agents. Note furthermore that the single-agent RL setup is more difficult to scale to different numbers of agents in terms of training.

6.3. Local Observation Size

By reducing the view of the agent from the overall problem to a local view centered on the agent, we expect to exchange solution quality for efficiency. To ascertain this, we have conducted experiments that show the effect of the local observation size on the final performance (see Table 5).

	Agents			
Depth	16	32	64	128
1	7.09	4.83	-3.98	-12.60
2	8.34	5.22	-0.37	-8.23
3	10.10	8.94	7.11	5.52
4	12.62	10.49	9.29	8.09
5	12.56	11.12	11.84	10.62

Table 5: Mean RL agent score (total reward) for different numbers agents at different depths. Higher scores are better. The mean is taken across all agents in an episode across 100 randomly problem instances, each consisting of 1024 nodes. The same problem instances were re-used for each combination of number of agents and depth.

The results do show a trade-off between local observation size and agent performance. Especially as the number of agents grows, the impact of nearby agents on an agent increases. Being able to see these nearby agents in the local observation greatly benefits the agent.

Do note that depending on the degree distribution of the environment graph, the practical local observation sizes can increase dramatically as the depth parameter is increased. In this case, the node degrees were uniformly distributed across the graph-based environment.

7. Conclusion

We have introduced a variant of the MAPF problem extended with timing constraints, which is more suitable when the MAPF setting is part of a larger logistics system. Its potential application to real-life situations has been described, and a Deep Reinforcement Learning-based approach to solving it in arbitrary graphs has been presented. This approach is attractive because its online component scales better than classic solvers, can be completely data-driven, and can be specialized to arbitrary use cases by constraining the training data. The approach is shown to be competitive with existing MAPF solvers when considering classic MAPF problem instances, and to scale well on MAPF problems with timing constraints. Limiting the agent's input to a local observation of a pre-specified depth much improves the scaling of the approach, at the cost of some solution quality. This trade-off can be regulated by setting different values for the depth. The decision to use a local observation in the RL implementation is investigated, showing its performance when compared to a single-agent setting (that always uses the entire problem state). and solution quality. Both single-agent and multi-agent RL settings are considered, and the multi-agent setup is shown to scale better in terms of solved instances.

8. Future Work

Outlined in this section are some focuses for future work based on the problem formulation and solving method presented above.

As mentioned earlier, maintaining a fixed local observation size across all agents and time steps is unlikely to be the best approach to choosing local observation sizes. Different agents may benefit from different state sizes at different points in time, depending on their situation. For example, if an agent is in an area of a graph that is sparsely connected, it may be beneficial to have a larger depth, so the agent can look further ahead and avoid collisions. Conversely, in a densely connected area of a graph, an agent may not have to look so far ahead, as there are plenty of alternative paths that can be taken.

The current solution makes use of a DQN, which is just one of many variants of Deep Reinforcement Learning algorithms. Its intuitiveness and ease of implementation make it a good candidate for an initial study, but other RL methods have been shown to outperform DQN in a variety of settings. Of particular interest are the group of RL algorithms that are based on direct policy optimization, such as Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO), see Schulman et al. (2017).

Acknowledgments

This work is being funded by the EU ECSEL Joint Undertaking under grant agreement no. 737459 (Productive 4.0 project) and no. 826589 (MADEin4 project). A use case was provided by Dutch railway company Nationale Spoorwegen (NS), with special thanks to Wan-Jui Lee.

References

M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 961–962. IOS Press, 2014.

E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. Eyal Shimony. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 740–746. AAAI Press, 2015.

L. Buşoniu, R. Babuška, and B. De Schutter. *Multi-agent Reinforcement Learning: An Overview*, pages 183–221. Springer Berlin Heidelberg, 2010.

M. Cáp, P. Novák, M. Selecký, J. Faigl, and J. Vokffnek. Asynchronous decentralized prioritized planning for coordination in multi-robot system. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pages 3822–3829. IEEE, 2013.

R. Cui, B. Gao, and J. Guo. Pareto-optimal coordination of multiple robots with safety guarantees. *Auton. Robots*, 32(3):189–205, 2012.

J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2137–2145, 2016.

M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer. Enhanced partial expansion A. *J. Artif. Intell. Res.*, 50:141–187, 2014.

J. K. Gupta, M. Egorov, and M. J. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems - AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*, volume 10642 of *Lecture Notes in Computer Science*, pages 66–83. Springer, 2017.

J. Jiang, C. Dun, T. Huang, and Z. Lu. Graph convolutional reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

O. Kaduri, E. Boyarski, and R. Stern. Algorithm selection for optimal multi-agent pathfinding. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 161–165. AAAI Press, 2020.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

S. M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

S. Leroy, J. Laumond, and T. Siméon. Multiple path coordination for mobile robots: A geometric algorithm. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1118–1123. Morgan Kaufmann, 1999.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6379–6390, 2017.

H. Ma, G. Wagner, A. Felner, J. Li, T. K. Satish Kumar, and S. Koenig. Multi-agent path finding with deadlines. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 417–423, 2018.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. Satish Kumar, S. Koenig, and H. Choset. PRIMAL: pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics Autom. Lett.*, 4(3):2378–2385, 2019.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, 2012.

G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.*, 195:470–495, 2013.

R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Satish Kumar, R. Barták, and E. Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 151–159, 2019.

J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics Research - The 14th International Symposium, ISRR 2009, August 31 - September 3, 2009, Lucerne, Switzerland*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer, 2009.

G. Wagner and H. Choset. Subdimensional expansion for multirobot path planning. *Artif. Intell.*, 219:1–24, 2015.

K. Zhang, Z. Yang, and T. Basar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *CoRR*, abs/1911.10635, 2019.