Learning with Search-Based Guidance for Partially Observable Multi-Agent Path Finding via Potential-Based Reward Shaping

by

Danoosh Chamani

B.Sc., University of Tehran, 2019

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Danoosh Chamani 2022
SIMON FRASER UNIVERSITY
Fall 2022

Copyright in this work is held by the author. Please ensure that any reproduction
or re-use is done in accordance with the relevant national copyright legislation.

Declaration of Committee

Name:	**Danoosh Chamani**
Degree:	**Master of Science**
Thesis title:	**Learning with Search-Based Guidance for Partially Observable Multi-Agent Path Finding via Potential-Based Reward Shaping**
Committee:	**Chair:** Yuepeng Wang Assistant Professor, Computing Science

Hang Ma
Supervisor
Assistant Professor, Computing Science

Mo Chen
Committee Member
Assistant Professor, Computing Science

T. K. Satish Kumar
Committee Member
Assistant Professor, Computing Science
University of Southern California

Jason Peng
Examiner
Assistant Professor, Computing Science

Abstract

Reinforcement Learning (RL) is a promising approach for real-world applications of Multi-Agent Path Finding (MAPF). However, its success depends on a good reward function, which is difficult to design manually in this complex domain. PBRSS (Potential-Based Reward Shaping with Search), our MAPF planner, automatically generates potential functions to guide RL-based MAPF agents using potential-based reward shaping. It invokes the theoretical and empirical advantages of accelerated training and likely convergence to better policies. We first formulate an adapted version of the Partially Observable MAPF (PO-MAPF) problem to standardize the comparison of RL-based against search-based planners and cross-fertilize techniques between them. We develop Partially Observable Conflict-Based Search (PO-CBS) as a generalization of CBS in the PO-MAPF domain. We then design the potential functions required for reward shaping using the PO-CBS plans and single-agent shortest path computations. PBRSS can be applied to any RL-based MAPF planner to improve its generalizability and performance.

Keywords: Multi-Agent Path Finding; Reinforcement Learning; Potential-Based Reward Shaping; Partial Observability

To the unsung ones.

Acknowledgements

Without Dr. Hang Ma's guidance, I would have never been able to undertake this endeavor. Hang introduced me to the essentials of multi-robot systems and multi-agent path finding, patiently walking me through their complexities and offering insightful comments and advice as I stumbled through my research.

Dr. Mo Chen and the rest of the CS Robotics Lab have my sincere appreciation for providing us with a warm and welcoming space in which to share our ideas, get feedback from our peers, and grow professionally. And naturally, I'll never forget the pizzas we enjoyed during our group meetings!

In addition, I'd like to express my gratitude to my colleague, Dr. T. K. Satish Kumar, for all of his assistance and insight during the writing process for this research study. He also taught me big time about different writing styles and how to effectively express my research findings.

Special thanks to Dr. Jason Peng for being my examiner and Dr. Yuepeng Wang for being the chair for my thesis defence.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Multi-Agent Path Finding (MAPF) problem is a combinatorial problem in which multiple agents operating in the same environment are required to plan conflict-free paths from their start locations to their goal locations. MAPF problems arise at the core of many real-world problem domains, and therefore, MAPF techniques have applications in these domains. Examples include aircraft towing vehicles [36], automated warehousing [61], and traffic control [12].

In MAPF, minimizing either the flowtime (the total number of timesteps required for all agents to reach their destinations) or the makespan (the timestep at which all agents reach their destinations) is NP-hard to solve optimally and NP-hard to approximate within any constant factor less than 4/3 [31]. However, practically, the MAPF problem can be solved using reductions to Boolean Satisfiability [52], Integer Linear Programming [62], and Answer Set Programming [13]. Other dedicated optimal MAPF algorithms include Independence Detection with Operator Decomposition [47], Enhanced Partial Expansion A* [15], Increasing Cost Tree Search [44], M* [56], and Conflict-Based Search (CBS) [43, 3, 5]. Many suboptimal MAPF algorithms have also been developed. These include Windowed-Hierarchical Cooperative A* (WHCA*) [45, 50], Push and Swap/Rotate [27, 8], TASS [21], BIBOX [51], and MAPP [59]. Various generalizations of the MAPF problem have also been formulated and solved using algorithmic enhancements to the above solvers [17, 29, 18, 30, 28].

Despite the extensive work done in the MAPF domain, centralized MAPF planners have several disadvantages. They require the environment to be fully observable. In the real world, this may not always be feasible since agents typically have limited sensing capabilities. For example, cars, drones, and warehouse robots can only recognize other agents in their vicinity and are still required to coordinate globally. Decentralized planners have the advantage of being applicable in domains where the agents' sensing capabilities are limited. Moreover, they may also be required to improve the scalability of MAPF planners since centralized planners typically have the disadvantage of scaling exponentially with the number of agents.

Decentralized planners typically decompose the global problem into smaller subproblems. They carry out navigation and conflict avoidance for the agents while letting them interact with the environment [59]. Existing decentralized planners include WHCA* [45] and MAPP [59]. Although these decentralized planners are much more scalable than centralized planners, they are not complete and often impose rules or restrictions on the agents' movements, leading to poor solution quality [57]. Another approach is to use reactive planners that follow precomputed single-agent shortest paths but avoid conflicts at execution time by making various kinds of adjustments [57]. While some decentralized and reactive planners still need full observability, some others, like ORCA [55], can work in partially observable domains. However, many decentralized planners are often susceptible to deadlocks and livelocks in cluttered environments [42].

Reinforcement Learning (RL) has gained attention in the MAPF domain as being capable of simultaneously addressing the issues of partial observability, scalability, and susceptibility to deadlocks and livelocks. First, RL-based MAPF agents use a belief state space that does not require full observability. Second, each agent decides its actions based on its policy, and, therefore, the deliberation time of a team of agents scales only linearly with the number of agents. Third, having the ability to learn from previous experiences, the RL-based MAPF agents are more resistant to deadlocks and livelocks. Examples of RL-based MAPF planners include [42], [26], [32], and [6].

However, RL-based MAPF planners require tedious training processes and often converge very slowly because of partial observability, sparsity of rewards, and the complexity of the domain in general. To alleviate these issues, researchers have attempted to provide additional guidance to the RL-based MAPF agents in their training phase. For example, in PRIMAL [42] and DHC [32], single-agent shortest path computations are used in different ways to guide the agents. In [26] and [58], guidance for the agents is derived from single-agent planners that have access to the global map of the environment, while in PRIMAL [42] and Global-to-Local Autonomy Synthesis (GLAS) [41], guidance is derived from global centralized MAPF planners. On the one hand, using single-agent planners for reward manipulation requires careful examination since indiscriminate reward manipulation can be detrimental [40]. On the other hand, using global centralized MAPF planners for guidance is computationally expensive. Moreover, using plans generated by global centralized MAPF planners in imitation learning [42, 41] can lead to overfitting [38].

In this thesis, we use the potential-based reward shaping method [37] for guiding the RL-based MAPF agents. In general, this method relies on the design of potential functions that map each state to a real number. We design two potential functions for the MAPF domain: one is derived from single-agent planners, and the other is derived from our own novel multi-agent planner called Partially Observable Conflict-Based Search (PO-CBS). PO-CBS is used to solve an adapted version of the Partially Observable MAPF (PO-MAPF) problem. We formulate the adapted version of the PO-MAPF problem to normalize the

comparison of RL-based MAPF planners against search-based MAPF planners and cross-fertilize techniques between them. In our formulation, agents have a restricted Field of View (FoV) but have access to a global map of the environment. Agents can communicate with each other when they are in each others' FoV. We call our RL-based MAPF planner PBRSS (Potential-Based Reward Shaping with Search).

The issue of indiscriminate reward manipulation is addressed in PBRSS via the use of potential-based reward shaping. It is well known that potential-based reward shaping retains the optimal policy [37]. In fact, it invokes both theoretical and empirical advantages of accelerated training and likely convergence to higher-utility Nash Equilibria. Moreover, the guidance is required only during the training phase. The issue of computational expense incurred by global centralized MAPF planners is addressed in PBRSS via the use of the adapted version of the PO-MAPF problem. Here, the joint space of planning is factorized into dynamic subgroups of agents over rolling time horizons.

PBRSS does not require the manual design of potential functions that are used to enrich the reward signals of the environment. Moreover, because of the generality of the potential-based reward shaping method, our approach can be used with any RL-based MAPF planner to improve its performance.

Chapter 2

Background and Related Work

In this section, we describe the background and previous work related to the various components of PBRSS.

2.1 MAPF

The MAPF problem [49] is the combinatorial problem of finding a set of optimal conflict-free paths for a set of m agents $\{a_1, a_2, \ldots, a_m\}$ on a given graph $G = (V, E)$, where each agent a_i has a starting vertex $s_i \in V$ and a goal vertex $g_i \in V$. Time is discretized into timesteps, and, at each timestep, each agent can either *wait* on its current vertex or *move* to an adjacent vertex. Two agents conflict with each other if, at the same timestep, they either are on the same vertex or traverse the same edge in opposite directions. Two common objectives of the MAPF problem are to minimize either the flowtime, i.e., the total number of timesteps required for all agents to reach their goal vertices, or the makespan, i.e., the timestep at which all agents reach their goal vertices. The MAPF problem is NP-hard to solve optimally and NP-hard to approximate within any constant factor less than 4/3 for both of these objectives [31]. In this thesis, we focus on the makespan objective, which is popularly used by RL-based MAPF planners.

Centralized MAPF planners search in the joint space of all the agents. Among them, CBS [43] is a MAPF planner that operates on the conflict-resolution space and finds a set of optimal, collision-free paths for all agents. CBS is a bi-level search algorithm where the two levels are referred to as the high level and the low level. Both the high-level and the low-level searches of CBS can be guided using clever heuristics [23, 14], constraint propagation [63], and Machine Learning [20]. Another CBS enhancement is the idea of disjoint splitting [24], in which subtrees of the high-level search are designed to have empty intersections to avoid redundant work. Despite incorporating various algorithmic ingredients, CBS invests exponential time in resolving conflicts as the complexity of the environment increases.

In our approach, i.e., in PO-CBS, we use CBS with disjoint splitting for solving PO-MAPF problems.

2.1.1 CBS

As mentioned before, CBS is a bi-level search algorithm [43]. Using space-time A*, it first plans paths independently for each agent and then uses a high-level search tree, called a constraint tree, to resolve any conflicts between them. The high-level search node will impose spatiotemporal constraints, and the low-level search will find a new path that satisfies those constraints for the agent that is in conflict. Algorithm 1, adapted from Algorithm 2 in [43], shows the pseudocode for the high-level search of CBS. Although the high-level search tree expands the nodes with the lowest flowtime (sum-of-costs), it can be easily modified to utilize the makespan objective instead. It's important to highlight line 11 of Algorithm 1, where a conflict is represented as the tuple (a_i, a_j, v, t), showing that agents a_i and a_j have a conflict on vertex v at timestep t. This representation only shows the vertex conflicts. Similarly, an edge conflict can be represented as the tuple (a_i, a_j, v_1, v_2, t), where agent a_i's transition from vertex v_1 to v_2 at timestep t causes an edge conflict with agent a_j's transition from vertex v_2 to v_1 at the same timestep. For ease of exposition, however, the edge conflicts are not shown in the pseudocode.

Disjoint splitting is a strategy that allows CBS to find a solution faster [24]. In the high-level search of CBS, the constraints are added for each of the conflicting agents, resulting in one child node for each agent, prohibiting them from being on a given vertex or traversing a given edge at the specified timestep. However, the disjoint splitting strategy generates two distinct child nodes for only one of the conflicting agents. The constraint added to one child node prevents the selected agent from being on a given vertex or traversing a given edge at the specified timestep. The constraint added to the other child node requires the selected agent to be on a given vertex or traverse a given edge at the specified timestep. It has been shown empirically that disjoint splitting speeds up CBS and more quickly finds optimal, collision-free paths.

2.2 RL-Based MAPF

RL can be viewed as the problem of maximizing the expected total reward accumulated by an agent operating in an environment over a certain time horizon [53]. The agent takes appropriate actions by learning a mapping function that maps each state to a distribution over actions. More specifically, the task can be formalized in the framework of a Markov Decision Process (MDP). An MDP M is defined by a tuple (S, A, T, γ, R), where S is the finite set of states, A is the finite set of actions, T is the transition function that defines the probability of transitioning from state $s \in S$ to the next state $s' \in S$ having taken a specific action $a \in A$, $\gamma \in (0, 1]$ is the discount factor, and R is the reward function that maps

Algorithm 1: High Level of CBS

Input: a MAPF instance on graph $G = (V, E)$ with agents $\{a_1, a_2, \ldots, a_m\}$
Result: a set of conflict-free paths for all agents, if exists

$Root.constraints \leftarrow \emptyset$
$Root.solution \leftarrow$ find individual paths for each agents a_i using the low level
$Root.cost \leftarrow$ sum-of-costs($Root.solution$)
insert $Root$ to $OPEN$
while $OPEN \neq \emptyset$ **do**
  $P \leftarrow$ a node with the lowest sum-of-costs from $OPEN$
  $C \leftarrow$ find conflicts for the paths in P
  if $C = \emptyset$ **then**
    return $P.solution$
  end
  $c \leftarrow$ the first conflict (a_i, a_j, v, t) in C
  foreach a_i *in* c **do**
    $A \leftarrow$ new node
    $A.constraints \leftarrow P.constraints \cup (a_i, v, t)$
    $A.solution \leftarrow P.solution$
    update $A.solution[a_i]$ using the low level
    $A.cost \leftarrow$ sum-of-costs($A.solution$)
    if $A.cost < \infty$ **then**
      insert A to $OPEN$
    end
  end
end

each state-action pair to a numerical reward. A policy π maps each state to a distribution over actions. The objective of an RL agent is to learn a policy that maximizes the expected total reward accumulated over a time horizon, discounting the future rewards by γ. This is formulated as $\mathbb{E}[\Sigma_{t=0}^{\infty} \gamma^t r_t]$.

Generally speaking, there are two ways to maximize the aforementioned objective. The first is to find the optimal policy π^* that maps each state to a distribution over the actions that yields the highest expected future rewards from that state. These methods are called policy-based methods because they directly compute the policy. The REINFORCE algorithm [54] is a well-known technique for finding π^* for a given MDP. Some other RL methods fall under the category of value-based methods. In these methods, the optimal policy is not computed directly. Instead, the Q-value function is computed, which returns the expected future rewards by taking action $a \in A$ in state $s \in S$ under policy π. This function indicates how valuable it is to perform a particular action in a particular state. From equation $Q(s, a)^* = \max_\pi Q(s, a)^\pi$, the optimal Q-value function $Q(s, a)^*$ can be easily derived. Consequently, an RL agent can act optimally by greedily taking the best actions using $Q(s, a)^*$. In particular, equation $\pi^* = \arg\max_a Q(s, a)^*$ can be used to determine the

optimal policy. Q-learning and DQN are well-known methods for finding $Q(s,a)^*$ for a particular MDP [35]. Nowadays, due to the complexity of the problems and the size of their state spaces, neural networks are utilized to approximate the Q-value or policy function.

In the MAPF domain, an extension of the MDP framework is more appropriate. This is the framework of the Markov Game [25]. Here, multiple RL agents interact with the environment and with each other. The environment can also be only partially observable for each agent. Therefore, we include a finite set of agents $\{a_1, a_2, \ldots, a_m\}$ and a finite set of observations O in the MDP formalization.

There are several RL-based MAPF planners, including PRIMAL [42], MAPPER [26], and DHC [32]. PRIMAL uses a combination of RL and imitation learning. To address the complexity of the problem, these planners use additional guidance derived from single-agent shortest path computations. PRIMAL penalizes agents that block other agents from reaching their goal. MAPPER penalizes agents that do not follow their reference shortest path. DHC measures progress towards the goal using the single-agent shortest path distance and represents it in the observation space.

While the use of single-agent shortest paths for guidance is both convenient and beneficial, it can also be detrimental. This can happen when the guidance manipulates the reward function indiscriminately, bearing the potential to change the optimal policy [40, 37]. For instance, in MAPPER, an oscillating behavior is observed when the reward signals are changed directly. Therefore, an additional penalty is placed on the agents that return back to their previous position. In PRIMAL, the risk of directly manipulating the reward signals is averted by creating an additional output in the actor-critic network and a corresponding additional term in the loss function for blocking other agents' paths. In PBRSS, we argue for the guidance derived from the single-agent shortest paths to be used in the framework of potential-based reward shaping. Potential-based reward shaping is known to be both necessary and sufficient for shaping the reward signals without changing the optimal policy [37]. Moreover, since it treats the RL agents as a black box [10], our approach can be used to improve any RL-based MAPF planner.

Some other works attempt to guide the RL agents using entire MAPF plans generated by centralized MAPF planners. For instance, in PRIMAL, the imitation learning component uses a centralized MAPF planner called ODrM* [57] for behavioral cloning. Similarly, GLAS [41] extracts demonstrations from local trajectories of global MAPF plans and uses them for imitation learning. Despite the promise of using centralized planners for guidance, the training time increases exponentially with the number of agents. In PBRSS, we address this issue via the formulation of the PO-MAPF problem and the development of a solver called PO-CBS.

2.2.1 DHC

As discussed previously, DHC [32] is one of the successful RL-based MAPF planners. The DHC method relies on three major components for its success. First, to speed up the training process, the Ape-X architecture [19] is utilized to parallelize the data gathering and gradient computations. More specifically, multiple copies of the environment are made, which are called actors. RL-based MAPF agents interact with these environments to collect data in a global buffer. Simultaneously, a single learner learns by sampling data from the global buffer and performing gradient updates on the neural network model. The learner then periodically updates the parameters of the networks used by the actors. In this manner, data gathering and neural network updates can be distributed across multiple computational resources. The overview of the Ape-X architecture is provided in Figure 3 of [32].

Second, as previously mentioned, DHC provides RL-based MAPF agents with guidance in their observation space. The single-agent shortest distances to the goal locations are computed at the start of each episode of interacting with the environment for each agent. This information is then represented in four distinct binary channels, known as heuristic channels, in the observation space of each agent. Each channel corresponds to one of the *move* actions (up, down, left, or right), and each location inside the FoV is filled with 1 only if the agent get closer to its goal by taking the corresponding action at that location. In other words, they provide the agents with all possible single-agent shortest paths in their observation space. An example of the heuristic channels is provided in Figure 2 of [32].

Thirdly, in DHC, they argued that the ability of RL-based MAPF agents to communicate with each other is crucial to their success in a partially observable environment. By communicating, an agent can obtain useful information from its nearby agents (the agents inside its FoV) and develop policies that are more collaborative and effective. In order to achieve this, an encoded version of each agent's observation is passed to its nearby agents. Later, graph convolution is used to derive communication, wherein each agent is treated as a node, and the graph is constructed by connecting the nearby agents.

DHC's neural network architecture consists of three modules: the observation encoder, the communication blocks, and the Q-network. These modules are the same for each agent but have different inputs. The observation encoder encodes the agent's observation from the current timestep and its latent message from the previous timestep to produce an intermediate message. The first communication block ingests the intermediate messages of the nearby agents and feeds the output to the second communication block that produces the latent messages for each of the agents. The structure of these two consecutive communication blocks is the same, but the inputs are different. The communication block employs convolution operations so that information percolates between nearby agents. The latent message of each agent is passed to the Q-network to produce the Q-values in a dueling manner [60]. The overview of the neural network architecture is provided in Figure 1 of [32].

2.3 Potential-Based Reward Shaping

Reward shaping is the process of enriching the reward signals of the environment to help the RL agents converge faster [37]. The new reward function R' can be written as $R' = R + F$, where R is the reward function of the environment in the original MDP and F is the shaping reward function. Technically, F can be any reward function. However, a poor choice of F can change the optimal policy and even cause oscillating behavior [40]. A potential-based reward shaping function is a function F that is defined as $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$, where s' is the next state reached by taking action a in state s, and $\Phi(s)$ is a real-valued function on individual states. γ is the same as the discount factor in the original MDP. Using such a function is both necessary and sufficient for policy invariance, i.e., the optimal policy remains unchanged [37]. Empirically, the use of such a function has been effective in many different domains [22, 7].

Several benefits of potential-based reward shaping have also been investigated in multi-agent settings. For example, [9] shows that the set of Nash equilibria remains unchanged and, with a well-designed potential function, the RL agents are more likely to converge to a higher utility Nash equilibrium. Other works [34, 1, 10] show that potential-based reward shaping also accelerates the training process and improves the solution quality in multi-agent settings.

Potential-based reward shaping can be extended to dynamic potential functions as well [11]. Such a function $\Phi(s, t)$ is defined on a state s and a time t at which the agent visits s. The reward shaping function is defined as

$$F(s, t, s', t') = \gamma\Phi(s', t') - \Phi(s, t), \tag{2.1}$$

where t is the time at which the agent visits s and t' is the time at which the agent visits the next state s'. The use of dynamic potential functions allows additional flexibility in reward shaping while preserving policy invariance in single-agent settings and the set of Nash equilibria in multi-agent settings. Furthermore, dynamic potential-based reward shaping has been empirically shown to be beneficial in both settings [11, 33, 4].

Chapter 3

Partially Observable MAPF

In this section, we introduce and formulate the PO-MAPF problem. Several RL-based MAPF planners address PO-MAPF without a standard formulation by merely viewing it as MAPF in partially observable environments [42, 26, 32, 6]. However, the absence of standardization may present some difficulties. For example, the formulation used in [46] is not consistent with those used by many RL-based MAPF planners [42, 26, 32, 6]. In general, we would like to formulate PO-MAPF by allowing the agents to have a full map of the environment. This is not an impediment in the real world since a map of the operating environment is usually readily available. In turn, this also allows RL-based MAPF planners to use globally computed single-agent shortest paths.

As mentioned before, MAPF [49] is the combinatorial problem of finding a set of optimal conflict-free paths for a set of m agents $\{a_1, a_2, \ldots, a_m\}$ on a given graph $G = (V, E)$, where each agent a_i has a starting vertex $s_i \in V$ and a goal vertex $g_i \in V$. Time is discretized into timesteps, and, at each timestep, each agent can either *wait* on its current vertex or *move* to an adjacent vertex. Two agents conflict with each other if, at the same timestep, they either are on the same vertex or traverse the same edge in opposite directions. Two common objectives of the MAPF problem are to minimize either the flowtime, i.e., the total number of timesteps required for all agents to reach their goal vertices, or the makespan, i.e., the timestep at which all agents reach their goal vertices.

Like MAPF, PO-MAPF aims to find a set of collision-free paths for a given set of m agents $\{a_1, a_2, \ldots, a_m\}$ on a given undirected graph $G = (V, E)$. The *wait* actions, *move* actions, vertex collisions, and edge collisions are defined as before. The objective is to minimize the flowtime or the makespan. However, unlike MAPF, PO-MAPF bestows a limited FoV of length l_i on each agent a_i. While l_i can be defined in graph-theoretic terms, the resulting FoV can represent different geometries depending on G. In the case that G is a 2D grid world, the FoV translates to a square of area $(2l_i + 1) \times (2l_i + 1)$ centered around the current location of a_i. Any observation that a_i makes within this FoV at timestep t is denoted by o_i^t. The agent a_i cannot observe anything, including the location of another

agent, if it is outside its FoV. Therefore, each agent operates in a partially observable environment. It does not have access to the global state of the environment.

The above formulation of PO-MAPF is consistent with that proposed in [46] but is not consistent with those used in most existing RL-based MAPF planners. To bridge this gap, we propose the following two enhancements to the above formulation: First, although each agent does not have access to the global state of the environment, it is able to access the full map of the environment. Second, when two agents are within each others' FoV, they can communicate with each other. Communication between agents is known to be useful in the MAPF domain [32]. In our framework, we allow the communicating agents to plan on their joint space.

Chapter 4

Partially Observable CBS

In this section, we present a novel adaptation of the CBS algorithm, called PO-CBS. While CBS is a bi-level search algorithm, PO-CBS is a tri-level search and execution algorithm. The three levels of PO-CBS are referred to as the high level, the mid level, and the low level. The mid-level and low-level searches of PO-CBS correspond to the high-level and low-level searches of CBS, respectively. The high-level search and execution of PO-CBS is based on a partitioning of the vertices of a graph that represents communicating agents.

4.1 High-Level Search and Execution

Algorithm 2 shows the high-level search and execution procedure of PO-CBS. It assumes that the FoV l_i for each agent a_i is the same. While this assumption can be easily relaxed, it is currently used for ease of exposition. Algorithm 2 operates on a dynamic undirected graph H, called the FoV graph, that changes at each timestep t. At any timestep, vertices of H correspond to agents, and edges of H correspond to pairs of agents that are in each others' FoV. The algorithm partitions the vertices of H based on (a) the connected components of H, (b) a user-specified maximum number of agents allowed in a partition, and (c) an algorithmic parameter adjustable within the limits of the FoV. On each of these partitions, the algorithm invokes the mid-level search of PO-CBS, which resolves conflicts between the agents in that partition within the boundary of their FoVs. The actions recommended for each agent are derived either from the mid-level search or from other conditions. They are executed for all agents across all partitions simultaneously.

After initialization in lines 1 to 3, Algorithm 2 constructs the FoV graph H in line 4 at each timestep t. In line 5, it partitions the vertices of H, i.e., $\{a_1, a_2, \ldots, a_m\}$, into W. Each $w \in W$ represents a non-empty subset of vertices, where $w_i \cap w_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{|W|} w_i = \{a_1, a_2, \ldots, a_m\}$. In lines 6 to 20, Algorithm 2 iterates over all the partitions $w \in W$. Each w is restricted to be a connected component of H. It contains a single agent if that agent is not in the FoV of any other agent. In such a case, lines 7 to 10, any neighboring vertex of the agent's location that is closer to its goal vertex qualifies as the

Algorithm 2: High Level and Execution Procedure of PO-CBS

Input: a MAPF instance on graph G with agents $\{a_1, a_2, \ldots, a_m\}$, max timestep T
Result: execution of an action for each agent at each $t \leq T$

```
1  t ← 0
2  while (∃ agent a_i not at its goal vertex g_i) ∧ t ≤ T do
3      A ← ∅
4      H ← the FoV graph, in which vertices represent agents
5      W ← the partitions of the vertices of H
6      foreach w ∈ W do
7          if |w| = 1 then
8              Let a_i be the only agent in w
9              act(i, t) ← follow a neighbor that is on any shortest path of a_i
10             A ← A ∪ act(i, t)
11         else
12             b ← collision-avoidance region on G w.r.t. w
13             I ← MAPF sub-instance w.r.t. w
14             P ← collision-free paths, p_i for each agent a_i, obtained using mid-level
                search on (I, b)
15             foreach a_i ∈ w do
16                 if P ≠ ∅ then
17                     act(i, t) ← follow path p_i
18                 else
19                     act(i, t) ← a random action
20                 end
21                 A ← A ∪ act(i, t)
22             end
23         end
24     end
25     Perform all actions act(i, t) ∈ A on the environment
26     t ← t + 1
27 end
```

next *move* action for it. When a connected component becomes larger than a user-specified threshold parameter, the mid-level search is disabled at that timestep for that partition. Another way to limit the size of the connected components is to artificially limit the FoV to be no greater than l_i. For ease of exposition, these two conditions are not described in the pseudocode of Algorithm 2.

In general, if w contains multiple agents, Algorithm 2 invokes the mid-level search on them. In line 12, it identifies a region b on G where conflicts have to be avoided. This region is computed to be the union of all pairwise intersections of the FoVs of the agents in w. It is referred to as the FoV of partition w. In line 13, a MAPF sub-instance is conceived on the agents in w. In line 14, the mid-level search is called to generate paths p_i for each agent $a_i \in w$ that leads a_i from its current location to its goal vertex on the global map.

However, the paths avoid collisions with each other only within the region b. This minimizes the computational effort in the mid-level search and is also consistent with the partial observability of the agents. Lines 15 to 20 iteratively gather the actions recommended for the agents. Line 16 tests the outcome of the mid-level search. If the search has succeeded, line 17 records the recommended *wait* or *move* action for the relevant agent. If the search has failed, line 19 records a random action instead. In line 21, the algorithm executes the gathered actions for all agents simultaneously.

The planning component (lines 6 to 20) factorizes the joint space of planning on all agents into smaller joint spaces on agents in each partition. It can thus be parallelized at each timestep and separate computational resources can be allocated to each partition.

4.2 Mid-Level and Low-Level Searches

The mid-level and low-level searches of PO-CBS are similar to the high-level and low-level searches of CBS [43], respectively. Analogous to the high-level search of CBS shown in Algorithm 2 of [43], the mid-level search of PO-CBS searches a constraint tree and resolves collisions between the agents' paths by branching and appropriately constraining the low-level searches of the generated child nodes. Similarly, the low-level search of PO-CBS is analogous to the low-level search of CBS that computes paths for individual agents under the spatiotemporal constraints imposed by the higher-level search nodes.

Despite the similarities, there are a few critical differences, particularly between the mid-level search of PO-CBS and the high-level search of CBS. First, the mid-level search of PO-CBS resolves collisions only within the FoV of the concerned partition, while the high-level search of CBS resolves collisions in the entire environment. In effect, the mid-level search of PO-CBS operates on a comparatively smaller joint space and on a smaller region. Therefore, it is expected to be faster and, consequently, is given a time limit of a few seconds. Second, the mid-level search of PO-CBS expands the nodes of the constraint tree according to their g-values given by the makespan, while the high-level search of CBS uses the g-values given by the flowtime.

Figure 4.1 shows the drawback of using flowtime minimization in the mid-level search of PO-CBS. In general, flowtime minimization can lead to livelocks since PO-CBS creates partitions based on limited FoVs. For example, in the left picture, Agent 2 and Agent 3 are in each others' FoV. Flowtime minimization in this situation leads to the red path for Agent 2. However, executing one step of it leads to the situation in the right picture. Here, Agent 2 and Agent 3 are not in each others' FoV. Consequently, a single-agent shortest path is planned for Agent 2, executing one step of which brings it back to the situation in the left picture, creating a livelock. However, makespan minimization does not result in such livelocks. In particular, for the situation in the left picture, makespan minimization yields

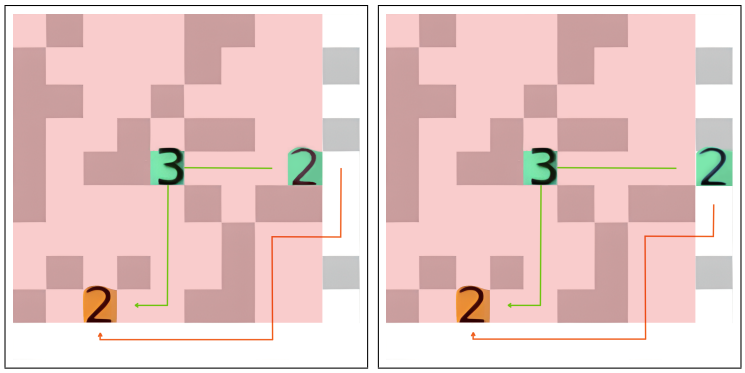


Figure 4.1: A livelock exhibited by flowtime minimization in the mid-level search of PO-CBS. Two agents, Agent 2 and Agent 3, are shown as part of a larger system. The orange color indicates the goal location, and the green color indicates the current location of the agents. Agent 3 is already at its goal location. The pink region indicates the FoV of Agent 3. For clarity, the FoV of Agent 2 is not indicated. Agent 2 and Agent 3 are in each others' FoV on the left picture but not on the right picture. Flowtime minimization leads to a livelock between these two situations.

the green path for Agent 2 while Agent 3 unblocks this path by slipping in and out of its goal location.

In general, the choice of makespan minimization is consistent with prior work on other RL-based MAPF planners [32, 6]. While flowtime minimization can also be done in the mid-level search of PO-CBS, its inability to resolve livelocks often leads to lower success rates. The low-level search of PO-CBS is identical to the low-level search of CBS, except that it terminates when it reaches a cell outside the FoV of the agent. A novel tie-breaking rule, explained in the next section, is also used in the low-level search.

4.3 PO-CBS Enhancements

Two major enhancements can be added to PO-CBS to improve its performance. The first enhancement is in the high-level search and the second enhancement is in the low-level search.

As discussed in the previous subsection, some livelocks can be avoided by using the objective of makespan minimization instead of flowtime minimization. However, even with makespan minimization, livelocks are not completely ruled out. For example, two agents heading in opposite directions through a narrow corridor can enter a livelock when the exits of the corridor are not observable to either of them. Even though conflict-free paths can exist for the two agents using detours outside the corridor, their limited FoVs force the mid-level search to oscillate. When the two agents are in each others' FoV, the necessary detours can be generated. However, while following these detours, the agents can get out of each others'

FoV and subsequently generate single-agent shortest paths through the corridor. Following their respective shortest paths brings them back to each others' FoV, creating a livelock.

The first enhancement is added in the high-level search to minimize such livelocks. It is a simple mechanism that stores o_i^t, for each agent a_i in each partition $w \in W$ on which the mid-level search is invoked at timestep t. At a future timestep $t' > t$, if the same partition reappears, the mechanism compares o_i^t to $o_i^{t'}$ and scores their similarity. The similarity metric is an element-wise comparison of the FoV cells in them, accounting for the fraction of cells that are either free, have obstacles, or have agents, at both timesteps. The percentage of unchanged elements is computed for each agent a_i and the overall product of these percentages μ_w is obtained for that partition. The mid-level search is invoked on w with probability $1 - \mu_w \eta_w$, where η_w is a decay factor initially set to 0.95 and is squared every time the mid-level search is skipped for that partition. As per line 19 of Algorithm 2, we note that the agents inside a partition act randomly when the mid-level search for it is skipped.

The first enhancement has two benefits. First, it detects many livelocks and resolves them using randomness. Second, it avoids the repeated computational efforts of the mid-level search in livelocks. These savings are significant particularly because livelocks tend to occur in cluttered regions of the environment involving numerous agents.

The low-level search of PO-CBS computes the shortest paths for individual agents under the spatiotemporal constraints imposed by the mid-level search. The second enhancement is used here to decrease the likelihood of creating cluttered regions. It is a simple mechanism that breaks ties between the shortest paths in favor of those that have smaller segments within b, the collision-avoidance region defined in line 12 of Algorithm 2. The mechanism is implemented by maintaining, for each search node in the region b, a b-value in addition to the f-value. The b-value is pre-computed to be the smallest horizontal or vertical distance to a cell outside b. The adapted space-time A* search proceeds as usual by expanding nodes with the smallest f-value but breaks ties in favor of those with smaller b-values. This enhancement has a tendency to avoid cluttered regions since it leads agents outside the FoV of other agents as soon as possible, dispersing them in the process.

Chapter 5

Potential-Based Reward Shaping with Search

In this section, we describe the main components of our approach, encapsulated in our planner PBRSS. One of the main ideas in PBRSS is to derive the potential function for reward shaping from single-agent shortest path computations as well as from PO-CBS. A dynamic potential function derived from PO-CBS can enhance one that is derived from single-agent shortest path computations since PO-CBS incorporates elements of multi-agent collision avoidance.

5.1 Single-Agent Shortest Path Potential Function: Φ_1

In this method, we generate the potential function Φ_1^i for each agent a_i using a single-agent shortest path algorithm such as A*. $\Phi_1^i(s)$ for a state s, corresponding to a location, is defined to be the shortest path distance from s to the goal location of agent a_i. Although Φ_1^i does not include any information about the other agents, it captures important information about the environment beyond agent a_i's FoV.

While DHC [32] demonstrates the usefulness of single-agent shortest path computations in the observation space, we use Φ_1^i in the reward function instead of the observation space. This enables agent a_i to receive an immediate positive or negative reward for following or deviating from its shortest path, respectively, but without changing the optimal policy. It is ideal to make $\Phi_1^i(s)$ correspond to the negative shortest path distance from s to the goal location. If s_0 is the initial location of agent a_i, we set $\Phi_1^i(s_0) = 0$ and define Φ_1^i recursively as follows:

$$\Phi_1^i(s') = \Phi_1^i(s) - \Delta. \tag{5.1}$$

Here, $\Delta = dist(s', g) - dist(s, g)$, s is the current state, s' is the next state, and $dist(s, g)$ is the single-agent shortest path distance from state s to the goal location g. The recursive form in Equation 5.1 is convenient for generalization and combination with other terms. It also makes the definition of Φ_1^i agnostic to the size of the map. Notably, recursively defining

the potential functions does not affect the theoretical guarantees of potential-based reward shaping fulfilling policy invariance, as the potential functions continue to be independent of actions. Consequently, the same theoretical guarantees can be applied once more [37, 11]. Considering $\gamma = 1$, the maximum shaping reward that agent a_i can receive using Φ_1^i is $+1$ at each timestep.

5.2 PO-CBS-Based Potential Function: Φ_2

The potential function Φ_1^i encourages agents to follow their individual shortest paths but has the drawback of not considering interactions between them. It also has the second drawback of being static and therefore unable to deliver guidance for *wait* actions since the shaping reward for them would be zero. In the MAPF domain, interactions between agents in the form of collision avoidance as well as *wait* actions are both key aspects of MAPF that make it NP-hard. Therefore, there is significant scope for improvement upon Φ_1^i.

We address the drawbacks of Φ_1^i by generating a dynamic potential function Φ_2^i for each agent a_i using the PO-CBS planner. Since the PO-CBS planner generates valid MAPF solutions, it gives us a chance to incorporate shaping rewards for collision avoidance as well as *wait* actions. In particular, useful information can be derived from PO-CBS on partitions with two or more agents.

PO-CBS generates a sequence of locations for each agent in each partition. For each agent a_i, a sequence of (s, t) tuples is prescribed, where s is a location and t is a timestep. We define the dynamic potential function Φ_2^i on the same (s, t) tuples. Specifically, $\Phi_2^i(s, t)$ is designed such that, considering $\gamma = 1$, transitioning to the next state prescribed by PO-CBS delivers a positive shaping reward $+p$ and transitioning to any other state delivers a negative shaping reward $-p$. $\Phi_2^i(s, t)$ delivers a shaping reward of 0 if the PO-CBS plan does not exist for agent a_i (either because PO-CBS times out or because it cannot find a solution in a deadlock or a livelock situation).

If s_0 is the initial location of agent a_i, we set $\Phi_2^i(s_0, 0) = 0$ and define Φ_2^i recursively as follows:

$$\Phi_2^i(s', t+1) = \Phi_2^i(s, t) + p\beta(a_i, s', t+1). \tag{5.2}$$

Here, s is the current state, t is the current timestep, and s' is the next state. $\beta(a_i, s', t+1)$ is an indicator function that returns $+1$, -1, or 0 depending on the agreement of s' with the PO-CBS plan, as discussed above. p is a quantification of the fraction of agents in that partition that adhere to the plan, as explained later.

Reward shaping for the individual agents does not force them to follow the PO-CBS paths. An agent that adheres to its PO-CBS path can still collide with another agent that does not adhere to its PO-CBS path. Therefore, indiscriminately giving a positive shaping reward to the adhering agent could be misleading. To address this issue, a proper *credit assignment* strategy should be employed. In general, credit assignment strategies determine

how the collective reward resulting from the agents' joint actions is distributed among them [39].

We use the fraction p in Equation 5.2 for the required credit assignment in our domain. It is guided by the following intuition: In a partition, the more agents there are that adhere to their PO-CBS paths, the more we want to reward them and the more we want to punish the non-adhering agents. This is because the PO-CBS plan is largely valid and the desire is to maintain its validity by rewarding the adhering agents and punishing the non-adhering ones. Similarly, the more non-adhering agents there are, the less we want to punish them and the less we want to reward the adhering agents. This is because the PO-CBS plan is largely invalid and the desire is to reduce the reward for the adhering agents that no longer have a reason to follow the PO-CBS plan and distribute the blame among the non-adhering ones.

5.3 Fine-Tuning the Potential Functions

As mentioned before, considering $\gamma = 1$, the maximum and minimum shaping rewards that Φ_1^i can deliver are $+1$ and -1, respectively. The same is true for Φ_2^i as well. However, these unit shaping rewards should be appropriately proportioned against the environment's reward signals. Previous theoretical and experimental studies have indicated that the magnitudes of the shaping rewards relative to the environment's reward signals bear a huge impact on the performance of the RL agents [37].

In principle, the ideal potential function is the optimal value function V^*, defined as per the environment's reward signals [37]. Although our potential functions can be used regardless of what the environment's reward signals are, it is desirable to introduce a space of possibilities within them to get as close to V^* as possible. Towards this end, we introduce the hyperparameters δ_1 and δ_2.

Equations 5.1 and 5.2 respectively become

$$\Phi_1^i(s') = \Phi_1^i(s) - \delta_1 \Delta \tag{5.3}$$

and

$$\Phi_2^i(s', t+1) = \Phi_2^i(s, t) + \delta_2 p \beta(a_i, s', t+1). \tag{5.4}$$

The two potential functions can also be combined in various ways to yield a larger set of possibilities. In this thesis, we experiment with a combination defined as follows:

$$\Phi_{12}^i(s', t+1) = (\Phi_1^i(s) - \delta_1 \Delta) + \delta_2 p \beta(a_i, s', t+1) \tag{5.5}$$

Algorithm 3 shows the overall control structure of PBRSS that builds and utilizes the potential functions Φ_1^i and Φ_{12}^i.

Algorithm 3: Potential-Based Reward Shaping with Search (PBRSS)

Input: multi-agent RL algorithm, PO-MAPF environment
Result: trained multi-agent RL model

foreach *episode* **do**
  $\Phi_1^i(s_0^i) \leftarrow 0$ for each agent a_i
  $\Phi_{12}^i(s_0^i, 0) \leftarrow 0$ for each agent a_i
  repeat
    Update $\Phi_1^i(s_+^i)$ for all reachable next states s_+^i of each agent a_i
    $P \leftarrow$ PO-CBS plan
    Get the action set A, recommending an action for each agent, using the multi-agent RL algorithm
    Execute the actions in A; obtain the rewards r^i and the next states s_+^i for each agent a_i
    Update $\Phi_{12}^i(s_+^i, t+1)$ using Equation 5.5 w.r.t. P for each agent a_i
    $F^i \leftarrow \gamma\Phi_{12}^i(s_+^i, t+1) - \Phi_{12}^i(s^i, t)$ for each agent a_i
    $r^i \leftarrow r^i + F^i$ for each agent a_i
    Use the new rewards r^i for each agent a_i in the multi-agent RL algorithm
    $s^i \leftarrow s_+^i$ for each agent a_i
  until *for all agents* a_i, s^i *is a terminal state*;
end

Chapter 6

Experiments

In this section, we present experimental results that demonstrate how PBRSS enhances DHC, a value-based RL MAPF planner [32], with respect to several performance metrics. Although DHC is chosen for this demonstration, PBRSS can enhance any other RL-based MAPF planner as well. On the performance metric of success rate, we compare DHC, PBRSS, and PO-CBS. We also evaluate PO-CBS with several timeout values for its mid-level search based on the success rate and the total wall-clock time performance metrics.

6.1 Setup

For each algorithm, its various performance metrics are gathered at testing time from its runtime behavior on 200 different MAPF instances. These MAPF instances are posed on 2D grid-world environments, where the grid maps are of size 40×40 with 30% randomly placed obstacles. The number of agents varies in the set $\{4, 8, 16, 32, 64\}$. At each timestep, agents act simultaneously, and an individual agent can either *move* up, down, right, or left, or *wait* at its current location. The MAPF instances used for evaluation are the ones provided in the GITHUB repository associated with DHC.[1]

In an episode, an RL-based algorithm is given a limited number of timesteps on a MAPF instance. It is deemed successful if all agents are routed to their destinations before timing out. In our experiments, the time limit is set to 256 timesteps. In addition to the DHC's MAPF instances, we also perform experiments on MAPF instances derived from more structured environments, such as warehouses, rooms, and mazes [48].

The training process implements the Ape-X architecture [19]. Here, as mentioned before, multiple actors, one for each environment (MAPF instance), continually generate experiences and store them in a global buffer. In parallel, a single learner is continually trained on these experiences, and the learned model is periodically copied out to all the RL agents.

[1] `https://github.com/ZiyuanMa/DHC`

The learned model is often a single Q-value network. In our experiments, we copy out the learned model every 400 timesteps, referred to as an epoch.

A performance metric related to training is the speed of learning. It is rendered as a curve that measures the number of epochs on the x-axis and the mean episode length, until success or timeout, on the y-axis. A similar performance metric plots the number of successful episodes on the y-axis.

We use another performance metric to measure the speed of learning. This metric arises from the context of curriculum learning. In general, the idea of curriculum learning is to graduate a learning agent through higher and higher levels of complexity [2]. Once the agent scores well at a certain level, it unlocks the next level. After transitioning to the next level, it is presented with more complex instances. The related performance metric is rendered as a curve that measures the number of epochs on the x-axis and the number of unlocked curriculum levels on the y-axis.

As previously used in DHC, a curriculum level for the MAPF domain is a tuple (l_i, l_j), where l_i is the number of agents and $l_j \times l_j$ is the map size. The next curriculum levels $(l_i + 1, l_j)$ and $(l_i, l_j + 5)$ are unlocked when our learned model solves at least 90% of the 200 most recently generated random MAPF instances at the current level. The first level has $l_i = 1$ and $l_j = 10$, while the last level has $l_i = 12$ and $l_j = 40$, resulting in a total of 84 different levels.

It is important to note that the success rate performance metric is the percentage of successfully solved instances, while the total wall-clock time is the total amount of time PO-CBS runs when trying to solve all of the instances, regardless of being successful or not.

6.2 Design Choices

To ensure proper comparison, we largely retain the design choices of DHC [32], except wherever required otherwise. Below, we describe the commonalities and differences. We also discuss the considerations under which we set an appropriate value of γ for PBRSS.

The general RL formulation is identical to that of DHC. The reward function of the environment remains unchanged. Each *wait* action at a non-goal location and each *move* action is given a slightly negative reward of -0.075 to avoid extraneous movements and incentivize reaching the goal location faster. Each *wait* action at the goal location is given a 0 reward. All agents receive a positive reward of $+3$ upon successfully completing an episode. Any action that results in a collision with an obstacle or another agent is given a negative reward of -0.5. If an action results in a collision, the colliding agent(s) are reverted to their previous states. Table 6.1 shows the described reward function.

The observation space is also identical to that of DHC. The entire observation space is a binary matrix of free cells and obstructions. An obstruction could be an obstacle or another agent. Each agent has only partial observability with a limited FoV. Within its

Action	Reward
Move	-0.075
Wait (off/on goal)	-0.075, 0
Collision	-0.5
All agents on goal	+3

Table 6.1: The overview of the environment's reward function.

FoV, an agent can observe the obstructions and categorize them as obstacles or agents in two different input channels. The observation space also has four channels, corresponding to each of the *move* actions, that provide binary information to the agent about getting closer to the goal location (see Figure 2 of [32]).

The neural network architecture is also identical to that of DHC (see Figure 1 of [32]). The hyperparameters, including the ones used in training, are also identical to those of DHC. For example, the batch size in training, the number of actors in the Ape-X architecture, the sequence length for the recurrent units, the number of hidden layers in the Q-network, and the maximum number of communicating agents are all kept the same. We also set a timeout of 3 seconds for the mid-level search of PO-CBS. This value is chosen empirically based on the comparisons we made between several values of timeouts.

Despite the similarities with DHC, we architect some important differences. This is done in consideration of PO-CBS being more expensive and the effects thereof when the learned model is updated independently of the actors. Since PO-CBS invests a fair amount of computation to generate collision-free paths, the global buffer is populated with less new data and the updates of the learned model are based on poor sample efficiency. To alleviate this, we introduce a synchronization protocol between data gathering and the learning process. The learned model is updated only when there is sufficient new data in the global buffer. While this synchronization increases the wall-clock time, it fixes the issue of sample efficiency. Importantly, it also allows us to analyze the benefits of PBRSS by standardizing the sample efficiency of the competing methods.

To set the appropriate value of the discount factor γ, we consider the following. On the one hand, the theoretical property of potential-based reward shaping on policy invariance is guaranteed only when the value of γ matches that of the original MDP. On the other hand, further studies in [16] have demonstrated that, for long enough episodes with $\gamma < 1$, the shaping reward can become negative when moving to a higher potential state, even when the potential values are positive. This can adversely affect the performance of the RL agents. Through experiments, we resolved the inconsistent suggestions made by the above arguments. We found that using the value of γ in the original MDP as it is but setting the value of γ for potential-based reward shaping to 1 compromises the theoretical guarantees on policy invariance but has huge benefits in practice.

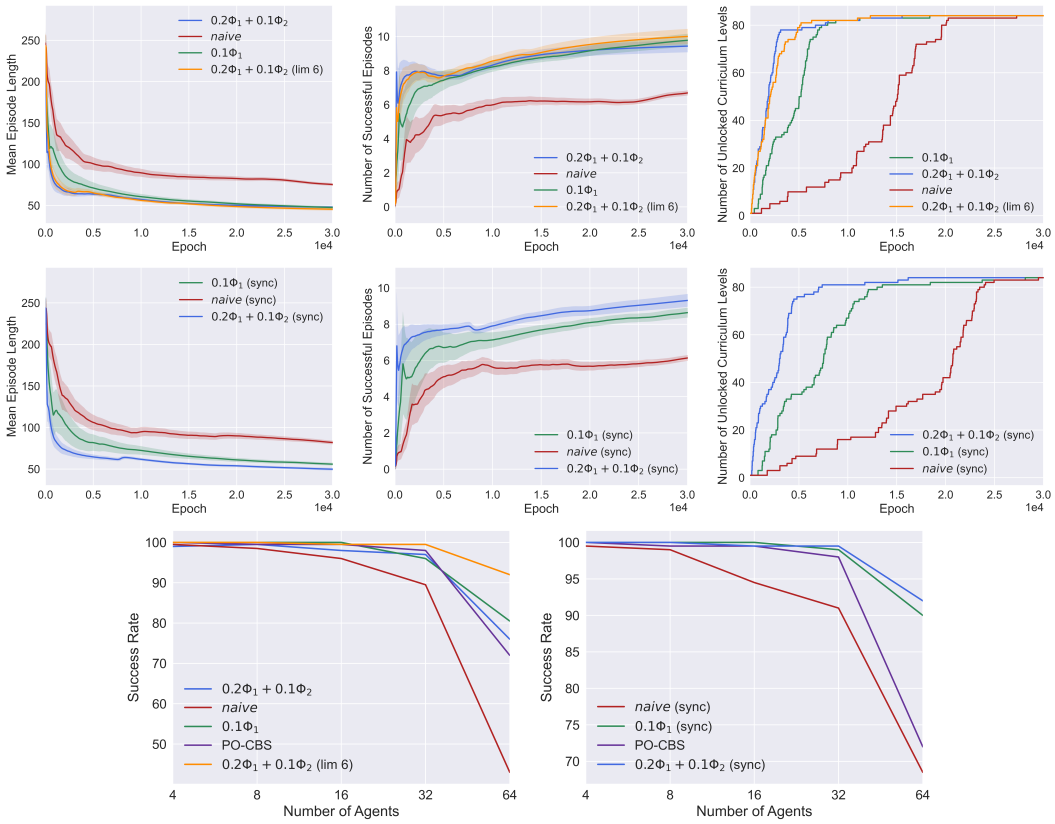


Figure 6.1: Different performance metrics related to learning for the asynchronous (first row and the left picture in the last row) and the synchronous (second row and the right picture in the last row) versions of DHC with PBRSS. The first, second, and third columns in the first two rows indicate the mean episode length, the number of successful episodes, and the number of unlocked curriculum levels, respectively. The third row indicates the success rate. The inlaid legends indicate the potential function(s) in the first two rows and PO-CBS additionally in the third row. The shaded regions around the curves in the first two columns of the first two rows indicate variance over 20 actors. 'lim 6' indicates the limit of 6 agents in the high-level search of PO-CBS. Φ_1 and Φ_2 that utilize δ_1 and δ_2, respectively, are indicated by $\delta_1\Phi_1$ and $\delta_2\Phi_2$, respectively.

6.3 Results and Analysis

We first test the learning capabilities of PBRSS by evaluating performance metrics related to training. The three columns in the first two rows of Figure 6.1 show these results for the mean episode length, the number of successful episodes, and the number of unlocked curriculum levels, respectively. We observe that, on all these metrics, PBRSS is significantly better than naive DHC in both the asynchronous and synchronous versions. This shows that potential-based reward shaping is beneficial in the MAPF domain. We also observe that PBRSS with the combination of Φ_1 and Φ_2 is better than PBRSS with just Φ_1, barring

map	**warehouse-10-20-10-2-1**			**room-32-32-4**			**room-64-64-8**		**maze-32-32-2**		**maze-32-32-4**	
num agents	32	64	128	32	64	74	32	64	32	64	32	64
0.2 Φ_1 **+ 0.1** Φ_2	86.5	50.5	1.5	99.0	84.0	55.5	83.5	43.5	87.5	6.0	66.5	29.0
0.2 Φ_1 **+ 0.1** Φ_2 **(lim 6)**	70.0	24.0	1.0	100.0	98.5	96.0	77.5	59.5	98.5	25.5	68.5	35.5
0.1 Φ_1	93.0	76.0	37.0	90.0	72.0	79.5	92.0	78.0	92.0	53.5	83.5	58.0
naive	16.0	0.0	0.0	88.0	23.0	9.0	74.0	43.5	59.0	0.5	15.5	0.0
0.2 Φ_1 **+ 0.1** Φ_2 **(sync)**	87.5	58.5	8.0	100.0	89.0	67.0	98.5	80.0	99.0	27.5	77.5	12.0
0.1 Φ_1 **(sync)**	95.5	92.0	65.0	99.0	78.5	59.0	100.0	93.5	97.5	27.5	99.5	63.0
naive (sync)	25.5	3.5	0.0	80.5	40.5	24.5	45.5	23.5	87.5	4.5	52.5	4.5

Table 6.2: Success rates of the synchronous ('sync') and asynchronous versions of DHC with PBRSS on benchmark maps with a varying number of agents. The rows indicate the potential function(s) used for reward shaping.

the number of successful episodes in the asynchronous version while leveling off.[2] Generally speaking, this shows that potential-based reward shaping with guidance from PO-CBS is beneficial. We further observe that limiting the number of agents to 6 in the high-level search of PO-CBS in the asynchronous version also improves performance.[3] This shows that making PO-CBS more efficient is beneficial.

The third row of Figure 6.1 shows the comparative performances for the success rate related to testing. The test instances are from DHC's GITHUB repository. We observe that, on this metric, PBRSS is better than naive DHC in both the asynchronous and synchronous versions. Both versions of PBRSS are also better than PO-CBS. This shows that potential-based reward shaping is superior to both DHC and PO-CBS that do not use reward shaping. We also observe that PBRSS with the combination of Φ_1 and Φ_2 is better than PBRSS with just Φ_1 in the synchronous version. This shows that potential-based reward shaping with guidance from PO-CBS is beneficial, provided that the sample efficiency is normalized. Otherwise, their comparative performances are reversed in the asynchronous version, since drawing guidance from PO-CBS takes more time. For the same reason, limiting the number of agents to 6 in the high-level search of PO-CBS increases the success rate in the asynchronous version.

Table 6.2 shows the comparative performances for the success rate in another testing phase. Here, the test cases come from structured maps [48], although training has been carried out on random maps. This setup is intended to test the generalizability of the competing methods. We observe that PBRSS is significantly better than naive DHC in both the asynchronous and synchronous versions. Once again, this shows the benefits of

[2]Here, δ_1 and δ_2 are empirically set to optimize performance.

[3]barring a few regions in the number of unlocked curriculum levels

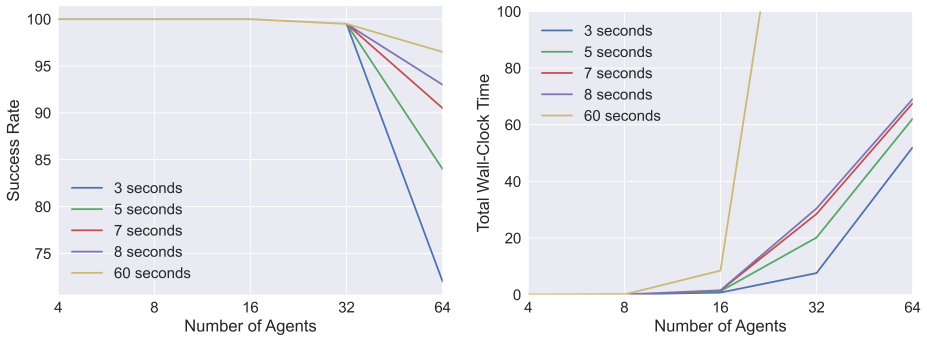


Figure 6.2: Different performance metrics related to several timeout values for the mid-level search of PO-CBS. The left picture indicates the success rate and the right one indicates the total wall-clock time. The performance metric of total wall-clock time is represented in minutes. The y-axis for the metric of total wall-clock time is limited to 100 minutes so that the differences between the curves can be shown clearly. For the timeout of 60 seconds, the total wall-clock time is 229 minutes for 32 agents and 354 minutes for 64 agents. The inlaid legends indicate the timeout values for the mid-level search of PO-CBS.

potential-based reward shaping. We also observe that PBRSS with the combination of Φ_1 and Φ_2 is not always better than PBRSS with just Φ_1. In this context, characterizing the benefits of using different potential functions may require training to be done on structured maps as well. This study is delegated to future work. Figure A.1 shows the map structures for this testing phase, which are borrowed from MOVING AI's website.[4]

Figure 6.2 compares the results of several timeout values for the mid-level search of PO-CBS. We note that raising the timeout value only increases the success rate for 64 agents, whereas the outcomes remain the same for fewer than 64 agents. Additionally, we find that choosing 3 seconds for the timeout has the least total wall-clock time. The reason for selecting a timeout of 3 seconds for the mid-level search of PO-CBS when used in PBRSS is that our training instances are not normally overcrowded with agents. Because of this, we expect that a timeout of 3 seconds will give us the same or similar performance as 60 seconds while keeping the wall-clock time lower.

We also demonstrate how advantageous it is to use the second potential function when the reward function is less sparse. To do this, we alter the original reward function of the environment so that agents are penalized by -0.025 when they are one *move* away from other agents. Using the new reward function in the asynchronous Ape-X architecture, we train both the naive version and PBRSS with the combination of Φ_1 and Φ_2 (lim 6). Figure 6.3 shows the comparative results between the learning models using the old and new reward functions of the environment. Even without the new, less sparse reward function, the number of successful episodes and the mean episode length are comparable to those with

[4] `https://movingai.com/benchmarks/mapf/index.html`

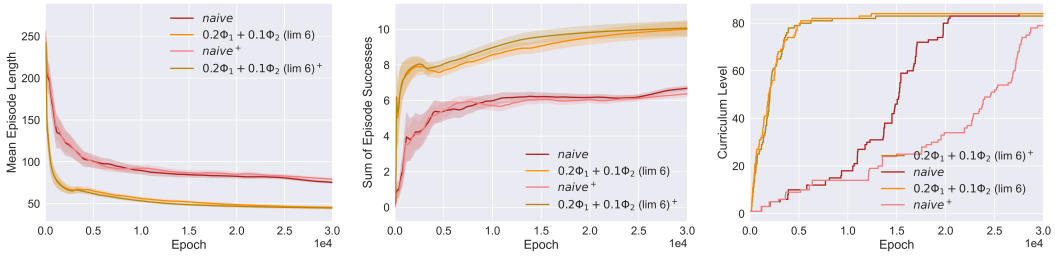


Figure 6.3: Different performance metrics related to learning for the asynchronous version of DHC with PBRSS using the old and the new reward functions of the environment. The first, second, and third columns indicate the mean episode length, the number of successful episodes, and the number of unlocked curriculum levels, respectively. The inlaid legends indicate the potential function(s). The shaded regions around the curves indicate variance over 20 actors. 'lim 6' indicates the limit of 6 agents in the high-level search of PO-CBS. Φ_1 and Φ_2 that utilize δ_1 and δ_2, respectively, are indicated by $\delta_1\Phi_1$ and $\delta_2\Phi_2$, respectively. '$^+$' indicates the models that are trained using the new reward function of the environment.

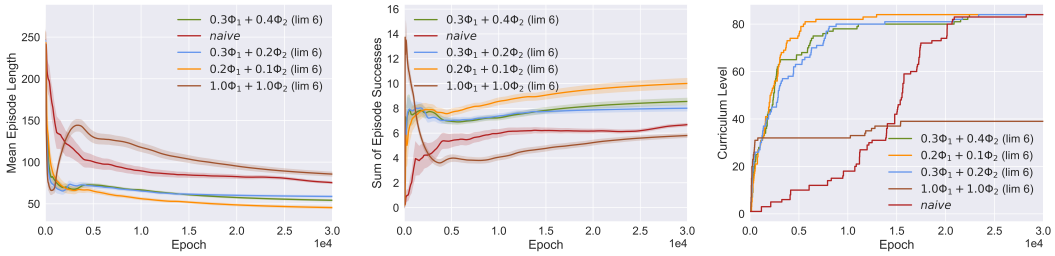


Figure 6.4: Different performance metrics related to learning for the asynchronous version of DHC with PBRSS utilizing various values for δ_1 and δ_2. The first, second, and third columns indicate the mean episode length, the number of successful episodes, and the number of unlocked curriculum levels, respectively. The inlaid legends indicate the potential function(s). The shaded regions around the curves indicate variance over 20 actors. 'lim 6' indicates the limit of 6 agents in the high-level search of PO-CBS. Φ_1 and Φ_2 that utilize δ_1 and δ_2, respectively, are indicated by $\delta_1\Phi_1$ and $\delta_2\Phi_2$, respectively.

the new, less sparse reward function. Furthermore, we find that PBRSS enables agents to gain more from the new reward function than the naive version. On the performance metric of the number of unlocked curriculum levels, however, we observe that models trained on the old reward function unlock the last levels of the curriculum faster, indicating that the new, manually designed reward function is not advantageous in more difficult instances.

Finally, we demonstrate the degree of hyperparameter adjustment required for δ_1 and δ_2. We arbitrarily select a few values for these two hyperparameters and train PBRSS using the combination of Φ_1 and Φ_2 (lim 6) in the asynchronous Ape-X architecture. As shown in Figure 6.4, keeping the magnitudes of both δ_1 and δ_2 close to the original reward function of the environment is sufficient to yield superior results compared to the naive version.

Chapter 7

Conclusions and Future Work

In this thesis, we presented PBRSS, a method that uses potential-based reward shaping for RL-based MAPF agents. PBRSS is able to draw guidance from search-based methods such as PO-CBS, our generalization of CBS to partially observable environments. We first formulated a version of PO-MAPF that is more relevant to existing RL-based MAPF planners, in which agents have a restricted FoV but a global map of the environment. We used this problem to normalize the comparison of RL-based MAPF planners against search-based MAPF planners and cross-fertilize techniques between them. Then, we developed PO-CBS by generalizing CBS to a partitioning of the agents. Subsequently, we used the plans generated by PO-CBS, along with other single-agent shortest path computations, to design the potential functions required for PBRSS. PBRSS can be used with any RL-based MAPF planner, invoking both the theoretical and empirical advantages of accelerated training and likely convergence to better policies. Through various experiments, we demonstrated the benefits of PBRSS over relevant baseline methods.

There are many avenues for future work. First, we would like to speed up PO-CBS to boost the performance of PBRSS in the Ape-X architecture by increasing sample efficiency. Second, we would like to further study the benefits of different potential functions when training is carried out on structured maps. Third, we would like to design more refined strategies for credit assignment in the potential functions.

Bibliography

[1] Monica Babes, Enrique Munoz De Cote, and Michael L Littman. Social reward shaping in the prisoner's dilemma (short paper). In *Proc. of AAMAS*, 2008.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[3] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Eyal Shimony. Icbs: Improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[4] Alberto Camacho, Oscar Chen, Scott Sanner, and Sheila A McIlraith. Non-markovian rewards expressed in ltl: Guiding search via reward shaping (extended version). In *GoalsRL, a workshop collocated with ICML/IJCAI/AAMAS*, 2018.

[5] Liron Cohen, Tansel Uras, TK Satish Kumar, Hong Xu, Nora Ayanian, and Sven Koenig. Improved solvers for bounded-suboptimal multi-agent path finding. In *IJCAI*, pages 3067–3074, 2016.

[6] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2):2666–2673, 2021.

[7] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–10, 2018.

[8] Boris De Wilde, Adriaan W Ter Mors, and Cees Witteveen. Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 87–94, 2013.

[9] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 225–232. ACM, 2011.

[10] Sam Devlin, Daniel Kudenko, and Marek Grześ. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(02):251–278, 2011.

[11] Sam Michael Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems*, pages 433–440. IFAAMAS, 2012.

[12] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, 31:591–656, 2008.

[13] Esra Erdem, Doga Gizem Kisa, Umut Oztok, and Peter Schüller. A general formal framework for pathfinding problems with multiple agents. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[14] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, TK Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 83–87, 2018.

[15] Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan Sturtevant, Robert C Holte, and Jonathan Schaeffer. Enhanced partial expansion a. *Journal of Artificial Intelligence Research*, 50:141–187, 2014.

[16] Marek Grzes and Daniel Kudenko. Theoretical and empirical analysis of reward shaping in reinforcement learning. In *2009 International Conference on Machine Learning and Applications*, pages 337–344. IEEE, 2009.

[17] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

[18] Wolfgang Hönig, TK Satish Kumar, Hang Ma, Sven Koenig, and Nora Ayanian. Formation change for robot groups in occluded environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4836–4842. IEEE, 2016.

[19] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[20] Taoan Huang, Sven Koenig, and Bistra Dilkina. Learning to resolve conflicts for multi-agent path finding with conflict-based search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11246–11253, 2021.

[21] Mokhtar M Khorshid, Robert C Holte, and Nathan R Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Fourth Annual Symposium on Combinatorial Search*, 2011.

[22] H Kim, Michael Jordan, Shankar Sastry, and Andrew Ng. Autonomous helicopter flight via reinforcement learning. *Advances in neural information processing systems*, 16, 2003.

[23] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, volume 2019, pages 442–449, 2019.

[24] Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. Disjoint splitting for multi-agent path finding with conflict-based search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 279–283, 2019.

[25] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[26] 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. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11748–11754. IEEE, 2020.

[27] Ryan J Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[28] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. *arXiv preprint arXiv:1612.05693*, 2016.

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

[30] Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[31] Hang Ma, Craig Tovey, Guni Sharon, TK Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[32] Ziyuan Ma, Yudong Luo, and Hang Ma. Distributed heuristic multi-agent path finding with communication. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8699–8705. IEEE, 2021.

[33] Patrick Mannion, Karl Mason, Sam Devlin, Jim Duggan, and Enda Howley. Dynamic economic emissions dispatch optimisation using multi-agent reinforcement learning. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2016)*, 2016.

[34] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th International Conference on Machine learning*, pages 601–608, 2007.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[36] Robert Morris, Corina S Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[37] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping.

[38] Xinlei Pan, Weiyao Wang, Xiaoshuai Zhang, Bo Li, Jinfeng Yi, and Dawn Song. How you act tells a lot: Privacy-leaking attack on deep reinforcement learning. In *AAMAS*, pages 368–376, 2019.

[39] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[40] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, volume 98, pages 463–471. Citeseer, 1998.

[41] Benjamin Riviere, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung. Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics and Automation Letters*, 5(3):4249–4256, 2020.

[42] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK 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, 2019.

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

[44] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence*, 195:470–495, 2013.

[45] David Silver. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, pages 117–122, 2005.

[46] Alexey Skrynnik, Alexandra Yakovleva, Vasilii Davydov, Konstantin Yakovlev, and Aleksandr I Panov. Hybrid policy learning for multi-agent pathfinding. *IEEE Access*, 9:126034–126047, 2021.

[47] Trevor Scott Standley and Richard Korf. Complete algorithms for cooperative pathfinding problems. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[48] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019.

[49] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[50] Nathan Sturtevant and Michael Buro. Improving collaborative pathfinding using map abstraction. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 2, pages 80–85, 2006.

[51] Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation*, pages 3613–3619. IEEE, 2009.

[52] Pavel Surynek. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[53] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[54] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[55] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE international conference on robotics and automation*, pages 1928–1935. Ieee, 2008.

[56] Glenn Wagner. Subdimensional expansion: A framework for computationally tractable multirobot path planning. 2015.

[57] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial intelligence*, 219:1–24, 2015.

[58] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020.

[59] Ko-Hsin Cindy Wang and Adi Botea. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[60] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

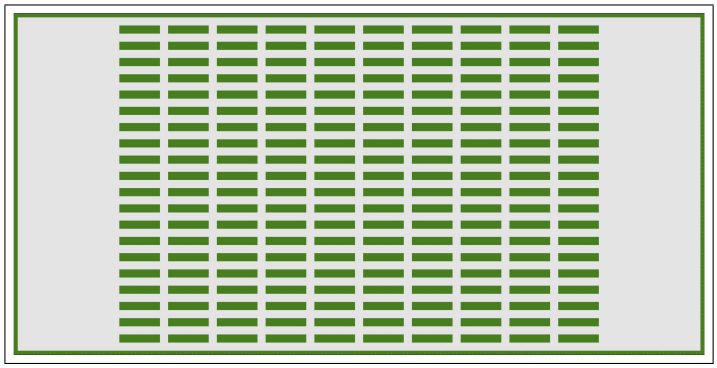
[61] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

[62] Jingjin Yu and Steven M LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 3612–3617. IEEE, 2013.

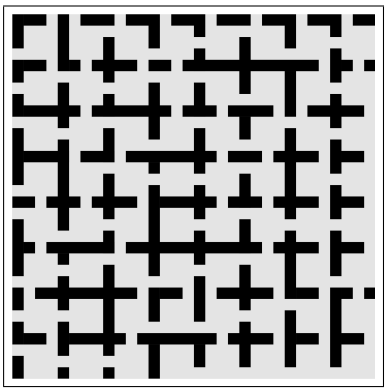
[63] Han Zhang, Jiaoyang Li, Pavel Surynek, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with mutex propagation. *Artificial Intelligence*, page 103766, 2022.

Appendix A

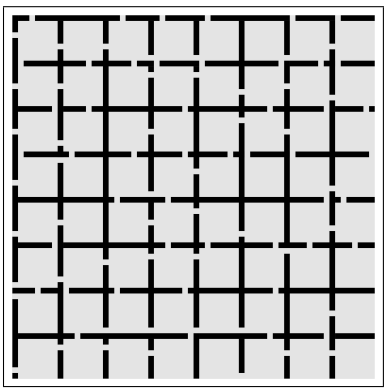
MAPF Benchmark Maps



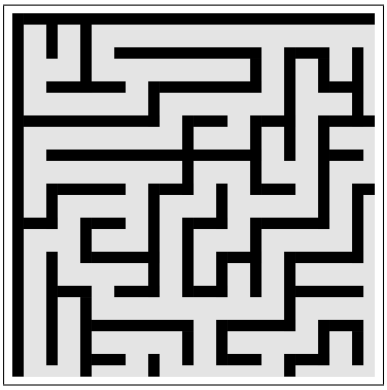
(a) warehouse-10-20-10-2-2



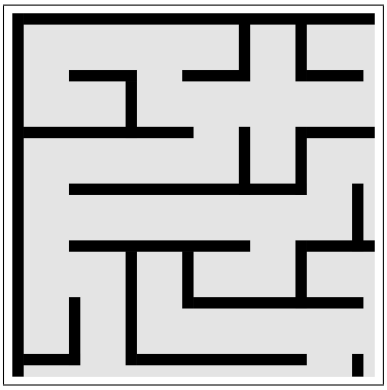
(b) room-32-32-4



(c) room-64-64-8



(d) maze-32-32-2



(e) maze-32-32-4

Figure A.1: MAPF Benchmark Maps [48]. These data are made available under the Open Data Commons Attribution License.