Abstract

Multi-agent pathfinding (MAPF) is a complex problem involving the coordination of movements among multiple agents to achieve a common objective in a shared environment. This problem has numerous industrial and research applications. However, in many practical cases, agents are unable to observe the entire environment in which they operate and must rely on local observations to make decisions. This scenario is known as partially observable MAPF (PO-MAPF). Recently, there has been a growing interest in utilizing learnable algorithms to tackle the MAPF problem and solve the PO-MAPF problem in a decentralized manner; yet the performance of these algorithms is oftentimes not validated out-of-distribution (OOD) or in adversarial scenarios, and the code is not properly open-sourced. This thesis focuses on decentralized PO-MAPF and aims to address the generalizability of learnable algorithms by conducting empirical evaluations beyond the training distribution, closing the gap in previous works. To achieve this, we extensively evaluate one of the state-of-the-art algorithms that utilizes communication between agents, Distributed Heuristic Communication (DHC). Through comprehensive experiments, we observe that the performance of DHC degrades when agents experience complete packet loss during communication. To mitigate this, we propose a novel algorithm, DHC-R. We provide detailed implementation specifics, including open-sourced model weights and pseudo-code.

Задача мультиагентной навигации (multi-agent pathfinding, MAPF) заключается в координации нескольких агентов в общей среде для достижения целей. Она имеет широкое применение в разных областях науки и техники, включая автоматизацию складских помещений. Обычно предполагается наличие центрального планировщика, обладающего информацией о нахождении агентов. Однако данная работа рассматривает сценарий частично наблюдаемого MAPF (partially observable MAPF, PO-MAPF), в котором агенты имеют ограниченное представление о среде и принимают решения на основе локальной информации. Такой сценарий может быть решен децентрализовано. В последние годы, благодаря доступности данных и вычислительных ресурсов, к PO-MAPF стали примененять обучаемые алгоритмы. Цель данной работы состоит в исследовании обобщаемости обучаемых методов для децентрализованного PO-MAPF. Для этого рассматривается один из наиболее эффективных алгоритмов - Distributed Heuristic Communication (DHC). Результаты обширного экспериментального анализа показывают, что качество работы DHC снижается в среде с полной потерей пакетов сообщений между агентами. Для улучшения работы в таких сценариях в работе представлен новый метод, DHC-R. Реализация, псевдокод и веса моделей доступны в открытом доступе.

Ключевые слова: partially-observable decentralized multi-agent pathfinding, reinforcement learning, out-of-distribution, navigation, AI safety, обучение с подкреплением, навигация в клеточных средах, децентрализованная навигация, частичная наблюдаемость.

Contents

List of Tables

Chapter 1

Introduction

Over the past few years, there has been unprecedented growth in the usage of autonomous systems, with major corporations investing millions of dollars to support this development. For instance, Amazon has deployed pathfinding systems in its automated warehouses (fig. 1.1) to optimize and coordinate the movements of hundreds of robots, ensuring that orders are picked and shipped quickly. Similarly, Yandex.Rover (fig. 1.2), a Russian autonomous delivery robot, plans optimal paths to navigate busy urban environments and deliver packages promptly. This trend is expected to continue, and the autonomous cars market is forecasted to generate a revenue of USD 15.55 billion by 2030[1].

Figure 1.1: Amazon automated warehouse; image from www.economist.com.

These examples highlight the importance of a multi-agent pathfinding (MAPF) problem which involves coordinating the movements of multiple agents to achieve a common goal in a shared environment. One of the most challenging scenarios in MAPF arises when agents cannot observe the entire environment in which they operate and must make decisions

[1]https://www.bloomberg.com/press-releases/2023-02-03/autonomous-cars-market-is-expected-to-generate-a-revenue-of-usd-15-55-billion-by-2030-globally-at-31-19-cagr-verified-market

Figure 1.2: Yandex.Rover; image from sdg.yandex.com.

based on partial observations. In such cases, communication with a central controller is also typically not allowed. This scenario, known as partially observable multi-agent pathfinding (PO-MAPF) or decentralized distributed pathfinding, poses a significant challenge as agents must cooperate without explicit coordination.

Addressing this challenging task has traditionally relied on classical computer science or heuristic algorithms. However, with the increasing availability of data and advancements in computational capabilities, researchers have begun exploring learnable algorithms for these problems. Reinforcement learning (RL),[2] in particular, has emerged as a promising tool for addressing these challenges.

In this work, our aim is to contribute to the field of MAPF by analyzing decentralized reinforcement learning methods for partially-observable multi-agent pathfinding scenarios. Specifically, we focus on extending the empirical evaluation for communication-based algorithms to scenarios beyond the training distribution. Our main goals are:

1. To implement the Distributed Heuristic Communication (DHC) algorithm [1] and reproduce the results from the original paper by Ma *et al.*

2. To validate the algorithm's performance on a wider range of maps, including those with a different structure from the training ones and out-of-distribution (OOD).

3. To analyze the agents' ability to navigate in environments that require complex navigation or cooperation.

[2]RL is a type of machine learning that focuses on training agents to make decisions based on rewards and penalties received from the environment

4. To explore and improve the algorithm's performance under adversarial scenarios, such as complete communication failure.

In addition to these primary goals, we focus on some minor, but equally important objectives:

1. To fully open-source the code base so it can be reused by researchers to analyze one of the state-of-the-art algorithms. We also set up programming instruments for easy development and experiments, such as a linter and automatic code formatter.

2. To open-source the model weights, including the weights of the model trained specifically for adversarial scenarios.

3. To provide a small, self-contained pseudo-code for the DHC algorithm [1] which researchers can use to analyze implementation details without the need to read a complicated multi-file code base.

In summary, our work investigates the ability of the recent state-of-the-art decentralized PO-MAPF algorithm by Ma *et al.* to work on maps outside of the training distribution and to be resistant to packet loss. To support our findings, we have also open-sourced the code base and model weights: https://github.com/acforvs/dhc-robust-mapf.

Chapter 2

Background

This chapter provides an overview of the foundational concepts necessary for this thesis.

2.1 Multi-Agent Pathfinding (MAPF)

The problem of multi-agent pathfinding consists of the computation of collision-free paths for a group of agents from their location to an assigned target [2].

The elements of a classical MAPF problem are the following:

- An undirected graph $G = (V, E)$;
- A set of N agents; each agent i is associated with its starting point $s_i \in V$ and the unique target point $g_i \in V$.

It is assumed that the time is discrete and that each agent can perform one action at each timestep. There are usually two types of actions available for each agent: *wait* or *move*. During the former, the agent remains in the same node of the graph, and during the latter, the agent moves to an adjacent one.

Path, or **plan**, for the i-th agent is a sequence of vertices $v_0, v_1, \ldots, v_N : v_j \in V; v_0 = s_i; v_N = g_i$. The vertex v_j is a result of a taken action (*wait* or *move*) from the previous vertex v_{j-1}, e.g. for every $j \geq 1$ either $v_j = v_{j-1}$ or $(v_{j-1}, v_j) \in E$, meaning that two consecutive vertices in a path are either adjacent or identical.

The **collision** between two agents occurs when one of the following transpires:

- At timestamp t two agents end up being in the same vertex $s \in V$. This can happen

if they either came from different vertexes to s or one of the agents stayed in place, and the other moved to s at timestamp t;

- Two agents, being located at $s_1 \in V$ and $s_2 \in V$ at timestamp $t-1$ changed their positions to s_2 and s_1 correspondingly at t.

The **solution** for the MAPF problem is a set of N collision-free paths, one for each agent.

2.2 Partially-Observable Multi-agent Pathfinding (PO-MAPF)

In the partially-observable setting, agents do not receive the full graph as an input. Instead, at each timestamp, they are able to observe only the part of the environment as well as some metadata that depends on the specific task.

Markov Decision Process (MDP)

In general, Markov decision process (MDP) is a formalization of online sequential decision-making in which making a decision affects future situations, as noted by Sutton and Barto [3].

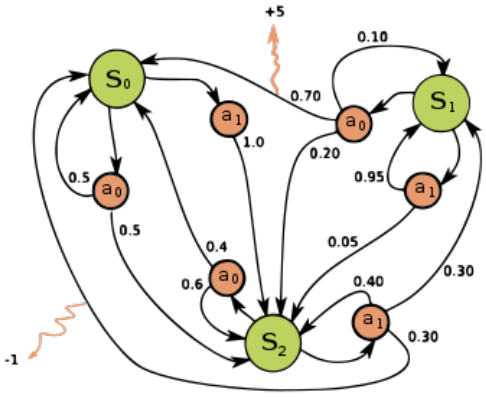


Figure 2.1: Example of a MDP; image from en.wikipedia.org.

The PO-MAPF problem can be formulated as an MDP and described by a 7-tuple

$$(N, \mathcal{S}, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, \{\mathcal{R}_i\}, P, \gamma) :$$

Figure 2.2: Example of a PO-MAPF task; the dashed red line marks field of view (FOV) of the red agent who cannot observe the rest of the map; image from https://github.com/AIRI-Institute/pogema.

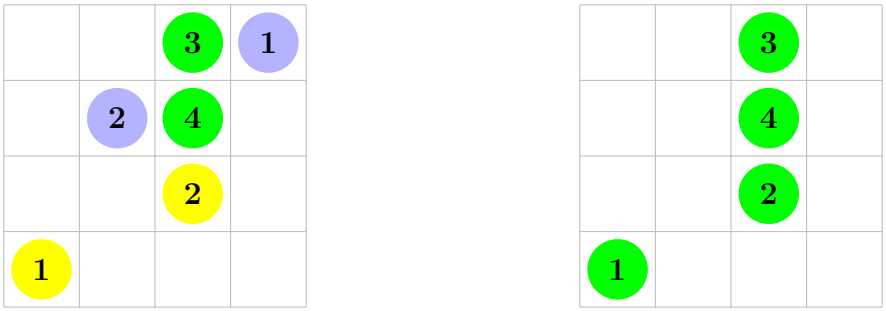
- N - number of agents;
- $\mathcal{S}$ is a set of states. A state is a representation of the environment;
- $\mathcal{A}_i$ is a set of actions allowed for agent i. $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$ - the joint action space;
- $\mathcal{O}_i$ is a set of observations for agent i. $\mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$ is a joint observation space;
- P is the dynamics of the MDP. $P(s', \vec{o}, r \mid s, \vec{a})$ outputs the probability of moving to the state s' from s and receiving an observation $\vec{o}$ and a reward r for it by selecting an action $\vec{a}$;
- $\mathcal{R}_i : \mathcal{S} \times A \to \mathbb{R}$ is a reward function for the i-th agent that outputs a numerical signal for the action that was taken from a given state;
- γ is a discount rate, $0 < \gamma < 1$ that determines how important the future rewards are to the current state.

In a *finite* MDP, $\mathcal{A}_i, \mathcal{O}_i, \mathcal{S}_i$ are finite sets.

A **policy** maps each state s in the state space to a probability $\pi(a \mid s)$ for each action a in the action space. The **solution** to the PO-MAPF problem is a policy that maximizes the sum of discounted rewards. Refer to the section 2.4 for more details.

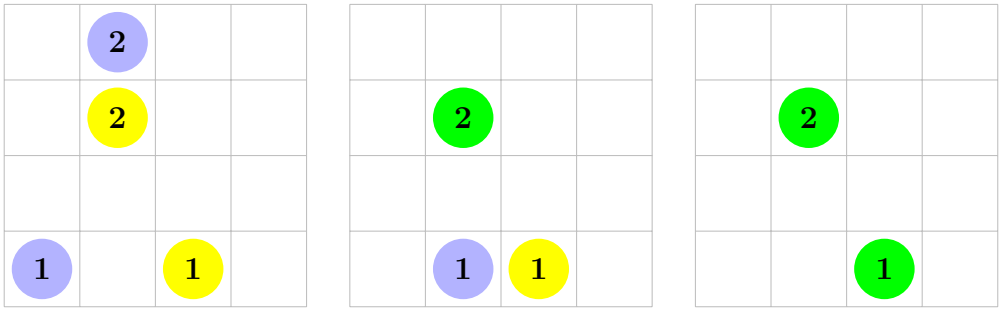
2.3 Problem Setting

In this manuscript, we consider PO-MAPF problem to be located on a 2d map, called a **grid**, which consists of *empty* cells where agents can be located and *blocked* cells occupied by obstacles (see fig. 2.2 for the example). In our setting, each agent can undertake 1 out of 5 available actions: move left, right, up or down, or stay still. Collisions, described in the section 2.1, are also not allowed. The goal of each agent is to reach the final destination, the **target**. In addition, agents can only observe a small part of the map, called FOV. Inside the field of view, agents can see obstacles and other agents. In the example shown in the fig. 2.2, FOV is represented by a small square window around the agent.



(a) ISR is 50%, CSR is 0%. (b) ISR is 100%, CSR is 100%.

Figure 2.3: CSR and ISR calculation. Agents are colored in blue, while the respective targets are colored in yellow. The green color indicates that an agent has reached its goal.



(a) t = 0 (b) t = 1 (c) t = 2

Figure 2.4: Makespan and flowtime calculation. Agents are colored in blue, while the targets are colored in yellow. The green color indicates that an agent has reached its goal. The makespan is calculated as $\max(1, 2) = 2$, and the flowtime is calculated as $1 + 2 = 3$.

In this thesis, a policy is a neural net that receives an encoded state/observation from

the environment and outputs $\pi(a \mid s)$. To evaluate a policy once it is found, several metrics are usually utilized:

- **Makespan:** The timestamp at which all agents have arrived at their corresponding locations, or the maximum arrival time of an agent at its goal location among all agents. The lower, the better;
- **Flowtime:** The sum of the arrival times of all agents at their goal locations. The lower, the better;
- **Collective success rate (CSR):** The percentage of tasks that were fully solved, meaning that each agent successfully reached its corresponding goal. The higher, the better;
- **Individual success rate (ISR):** The percentage of agents that arrived at their goals. The higher, the better.

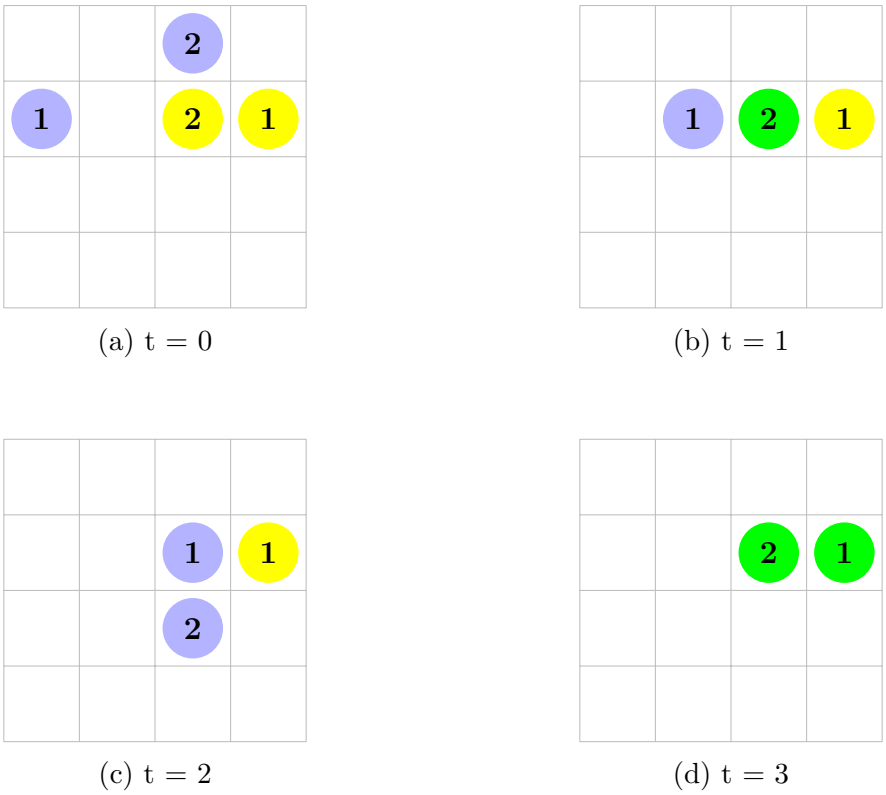
An example of how to calculate CSR and ISR can be found in fig. 2.3. During the test invocation, these metrics are usually averaged across different runs. In the situation depicted in fig. 2.3, the averaged result would be 75% for ISR and 50% for CSR. Examples of makespan and flowtime calculations can be found in fig. 2.4 and fig. 2.5.

2.4 Reinforcement Learning (RL)

Reinforcement learning [3] is learning from interactions with an environment to maximize a numerical signal, called reward. In this paradigm, an agent interacts with an environment by taking actions and observing the immediate reward and the new state of the environment. This process can be formulated mathematically as a Markov decision process.

Episodes and Returns

In general, there are two types of tasks in RL: *episodic* and *continuing* tasks. In episodic tasks, the agent's interactions with the environment break into sub-sequences called episodes. Each episode ends in a terminal state, which is a special state that resets the agent's state



(a) t = 0 (b) t = 1

(c) t = 2 (d) t = 3

Figure 2.5: Makespan and flowtime calculation. Starting position of an agent is colored blue, and the target is colored yellow. Green denotes that the agent reached its goal. Makespan = 3, flowtime = 3 + 3 = 6.

to one of the starting states with a zero reward. In continuing tasks, terminal states are not present, and the agent interacts with the environment forever.

The goal of a reinforcement learning agent is to maximize the expected **return** after timestep t which can be defined as follows:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

In the equation above, r_i is the reward received at timestamp i. If the task is episodic with the terminal state at timestamp T, all $r_{T+1}, r_{T+2} \ldots$ are zero.

Policy and Value Function

A **policy** determines the behavior of the agent. A policy $\pi : S \times A \to [0, 1]$ is a mapping from states to actions; $\pi(a \mid s)$ is the probability of choosing the action a in s, similarly to the previous section.

The value function of a state s under policy π, $V_\pi(s)$ is the expected return that the agent receives starting at state s and following the given policy π. $V_\pi(s)$ is called the **state value function** and can be formally defined as follows:

$$V_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s\right]$$

The **state-action value function** $Q_\pi(s, a)$ outputs the expected return when following the policy π starting at s and taking the action a as the initial action.

$$Q_\pi(s, a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s, A_t = a\right]$$

In addition, an **advantage function** is usually defined as the difference between the Q-function and a V-function:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

In the past, numerous methods have been suggested to find the optimal policy that maximizes the sum of discounted rewards. While these methods are beyond the scope of this study, one commonly used approach is to approximate or precisely calculate the Q-function. In this work, we adopt this approach and train a neural network to approximate the Q-function that can be then written as $Q(s, a, \theta)$. Once the network is trained, agents can utilize it to select the action with the highest expected return.

Chapter 3

Related Work

Pathfinding is a crucial and versatile problem that has many different approaches to solving it. In this chapter, we discuss related research papers that have previously addressed this problem from different perspectives, namely: heuristic and learning-based. In particular, we focus on a multi-agent scenario but briefly discuss relevant single-agent work as well.

3.1 Heuristic Algorithms

Single-agent Perspective

In a single-agent setting, the goal is to find the optimal path from a starting point to a finishing point. The standard approach for this problem is to use the A* algorithm [4]. This algorithm uses a heuristic function to estimate the distance from the current location of the agent to its goal and guide the algorithm toward the optimal solution, which is similar to the way the Dijkstra algorithm works.

Although the A* algorithm can find the optimal solution, it may not be suitable for real-world usage due to its time complexity. The algorithm can take a lot of time to converge on big or complex maps. To address this issue, many improvements have been made, such as IDA* (iteration deepening A*) [5]. This algorithm gradually improves the depth limit at which A* is interrupted, and the limit is increased if, and only if, the previous one was not sufficient to find a solution.

There are many other single-agent algorithms, but since they are not our main focus in this thesis, we will not be investigating them further.

Multi-agent Perspective

For multi-agent scenarios, there exist several methods that extend the A* algorithm. As a basic modification, one could run the A* algorithm in a joint space of actions. However, this would result in an exponential growth of the number of vertexes in a graph, and the algorithm would take a long time to output a solution for a large number of agents. LRA* [6] is another possible algorithm that utilizes A* at its core. In LRA*, each agent is planned independently. If a collision occurs, the agents' paths are re-planned. Another algorithm, M* [7], plans each agent independently but transfers the planning to a higher-dimensional space in case of a collision, which would reduce the exponential growth of the search space. A further improvement of M* is OdRM* [8] which integrates it with operator decomposition and performs a lazy search of the out-neighbors of a given vertex. Additionally, combinatorial optimization techniques such as Large Neighborhood Search [9] have been used to improve suboptimal solutions.

Moreover, researchers have explored the use of *hierarchical* algorithms for the MAPF problem. These algorithms partition the search space into multiple levels, with results from lower levels reused to solve higher and more complex levels. Notable examples include work by Silver or Sharon *et al.* [10, 11]. For instance, in "Cooperative Pathfinding", agents are planned using A*, with high-level features of the environment used at lower levels and more precise representations at higher levels.

The well-known Conflict-Based Search (CBS) algorithm [12] is another heuristic algorithm that has undergone various improvements, such as the Iterative-Deepening Conflict-Based Search [13], which increases the allowed makespan of the solution step-by-step while resolving conflicts between agents, and suboptimal versions of CBS [14], such as Greedy-CBS, Bounded CBS and Enhanced CBS, which trade solution optimality for search speed.

Another group of approaches reduces the MAPF problem to a SAT problem, such as "Efficient SAT Approach to Multi-Agent Path Finding Under the Sum of Costs Objective" [15] which uses time expansion graphs for the reduction. In "Branch-and-Cut-and-Price for Multi-Agent Pathfinding" [16], CBS is combined with mixed-integer programming.

However, in heuristic approaches, agents are often planned using search algorithms and

require knowledge of the entire map. Although optimal algorithms may provide optimal solutions, they are not polynomial and can take a long time to converge. In situations where optimality is not guaranteed, conflicts can arise during plan execution, and **agents may need to communicate with a central controller for re-planning**. This may result in frequent calls to the global expert, causing agents to wait for a response and remain idle.

3.2 Learning-based Algorithms

In response to the challenges outlined above, researchers have proposed several non-heuristic methods for MAPF, with learning-based algorithms gaining increased popularity in recent years. Among these, RL approaches have been particularly appealing.

While some RL-based methods may still rely on a central planner, they use it not as a sole point of coordination (which can create a single point of failure), but as an expert to guide learning during training. The expert generates trajectories, and agents try to replicate them. These algorithms typically use imitation learning (IL) and can be executed in a decentralized manner. They can also accommodate partially observable maps, where agents can only observe a small part of the environment, called FOV and often represented as a 9×9 square with the agent at the center.

For instance, in "PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning" [17], a switch determines whether the next stage will be RL- or IL-based (see fig. 3.1). During the IL phase, the central planner, OdRM* [8], generates new trajectories that the agent can follow using behavioral cloning. During the RL stage, agents regress on generated trajectories to learn. In "Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning"[18], a decoupled approach is used, where agents initially plan their trajectories using the A* algorithm but are allowed to deviate from them if local observations suggest an alternative path.

However, IL builds upon the idea that experts generate trajectories of good quality. This is not the only problem with the described approach. First, the experts themselves may be too complex and time-consuming to run, making it impractical to use them for training. Additionally, while regressing on expert demonstrations, the model may not learn how to

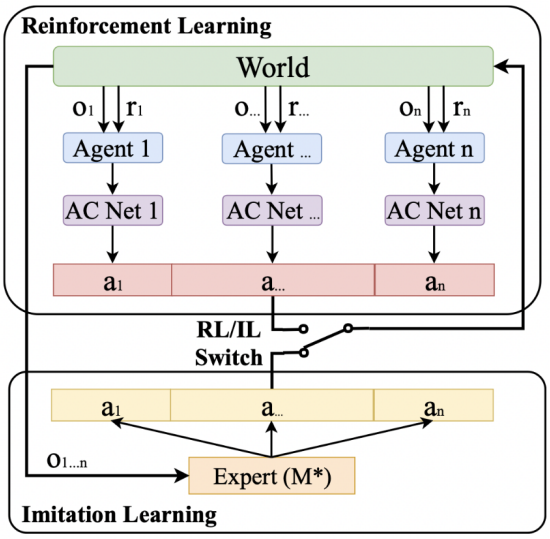


Figure 3.1: PRIMAL; training procedure.

efficiently deviate from them even if doing so would be beneficial, as discussed in [1].

Communication-based Methods

Recently, there has been growing interest in communication-based algorithms that leverage the exchange of packets [messages] between agents to improve coordination. Early works in this area used a collective reward to guide agent behavior, but later approaches employed individual rewards, enabling decentralized execution. However, in real-world systems, excessive communication can lead to unnecessary overhead, so it is important to limit the number of messages exchanged. Several recent papers have explored techniques for reducing message traffic without compromising performance.

For instance, Singh *et al.* [19] use a gating function to selectively broadcast messages, while each agent maintains a personal LSTM-based network. Kim *et al.* [20] allow a limited number of agents (k) to generate messages, and these agents are selected based on the trained weights, either by choosing the $top(k)$ agents with the highest weights or by sampling agents from an empirical probability distribution imposed by the weights. In "Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control" [21], messages are only submitted to the system if they have a high variance, which indicates

that they provide meaningful information. "Succinct and Robust Multi-Agent Communication with Temporal Message Control" [22] employs a message buffer to prioritize messages that add new information. **However, all of these methods have been trained in a centralized manner.**

More recent works have explored decentralized training methods as well. For example, in "Multi-Agent Path Finding with Prioritized Communication Learning" [23], agents form clusters based on learned priorities, and communication occurs only within clusters. Central agents receive and aggregate messages from others, then transmit the results back to their clusters.

Graph Neural Networks

The development of graph neural networks (GNNs) has led to recent works leveraging this powerful architecture for the MAPF problem. One early work is "Graph Neural Networks for Decentralized Path Planning" [24], which combines imitation learning with communication using CBS as a central planner. After an agent receives a feature vector by passing its observation matrix through a convolutional neural network (CNN), it transmits this vector to its neighbors located within a communication radius R_{comm}. Notably, this communication radius may be larger than the agent's field of view. In "Message-Aware Graph Attention Networks for Large-Scale Multi-Robot Path Planning" [25], the results of the previous paper were improved by using Enhanced CBS (ECBS) as the central planner and graph attention networks (GATs) instead of GNNs.

However, it is worth noting that the number of works that utilize communication and decentralized training together remains limited. For instance, in "Distributed Heuristic Multi-Agent Path Finding with Communication" [1], agents receive information from all neighbors and consider messages only from the two closest ones. Another study, "Learning Selective Communication for Multi-Agent Path Finding" [26], adopts a similar architecture but employs a request-reply mechanism instead of broadcasting, allowing only the most crucial agents to exchange information.

3.3 Other Methods

Due to the overall development of deep learning systems, other methods have been highly utilized as well. Transformers, for example, have recently gained attention in this field, with their use in "Multi-Agent Path Finding Using Imitation-Reinforcement Learning with Transformer" [27] to encode observations, and in *TransPath: Learning Heuristics For Grid-Based Pathfinding via Transformers* [28] to learn adjusting constants for heuristic functions.

Furthermore, other formal definitions of the MAPF problem have been explored beyond the classical setting. In the *lifelong* setting, for instance, a new goal is generated for an agent once it reaches its previous one [29]. This ensures a fixed number of agents in the environment, with each agent having a goal to reach at any given point. Another variant is the Robust MAPF [30], which seeks to find stable trajectories even if an agent decides to stay idle for no more than k steps during plan execution, provided that other agents continue acting according to their original plan.

3.4 Summary

In conclusion, a wide range of heuristic and learning-based algorithms have been suggested to address single- and multi-agent pathfinding problems. While some heuristic approaches are capable of guaranteeing optimal solutions, their applicability is often limited due to the significant computational time required and the difficulty in applying them to partially-observable settings without a central planner. In contrast, learning-based methods are often able to handle partially observable maps and operate in a decentralized manner, with each agent using a local copy of the trained policy and treating other agents as moving obstacles. These approaches may also incorporate a centralized planner or use imitation learning to guide the path-planning process or leverage communication to help agents cooperate.

Based on the reviewed papers and ideas, we can outline the following key takeaways about learnable algorithms:

- Fully decentralized algorithms are still limited in number, and centralized training is often necessary;

- State-of-the-art algorithms that focus on advancing goal-reaching capabilities rarely test for generalization, i.e., the ability to work on maps outside of the training distribution and be resistant to packet loss in communication-based methods;
- Over-reliance on imitation learning can lead to suboptimal solutions and slower training processes, especially for complex maps where expert knowledge may not be sufficient;
- The official code for many algorithms is either not open-sourced at all or is difficult to reproduce.

To address these issues, we provide a thorough empirical evaluation of the fully decentralized learning-based algorithm that utilizes communication, "Distributed Heuristic Multi-Agent Path Finding with Communication". Furthermore, motivated by the work of Phuong and Hutter [31], we believe that providing pseudo-code implementation can be useful for theorists or those who want to implement the algorithm from scratch. Therefore, we provide a self-contained pseudo-code implementation of DHC in the next chapter. To encourage future research in the area of RL-based algorithms for MAPF, we have also open-sourced an easy-to-use and reproducible implementation[1] of DHC, along with the model weights.

[1]In contrast to the original repository, which we found to have a different configuration from the one used in the paper, making it difficult to reproduce the results.

Chapter 4

Method

In this chapter, we describe the algorithm used to analyze the performance of decentralized learnable algorithms in partially observable environments. Two algorithms were presented in the previous chapter that can be used for this task:

1. "Distributed Heuristic Multi-Agent Path Finding with Communication" by Ma *et al.* [1], introduces Distributed Heuristic Communication (DHC).

2. "Learning Selective Communication for Multi-Agent Path Finding" by Ma *et al.* [26], introduces Decision Causal Communication (DCC).

Both are learnable algorithms that leverage communication and have been presented at major international conferences. However, since DHC uses broadcasting communication where agents transmit their packets [messages] without waiting for others to accept it, which we consider less faulty in real-world scenarios since it does not require a "handshake" from other agents, we decided to select DHC as the main algorithm for this work. We have employed three DHC-based algorithms in our research:

1. **DHC-paper:** This algorithm represents the original DHC algorithm as described by its authors. We have utilized the weights provided by the authors along with their original implementation. The DHC-paper serves as our baseline for comparison and evaluation.

2. **DHC-ours:** We have implemented the DHC algorithm based on the original implementation but trained it using our own computing resources. The purpose of

DHC-ours is twofold: to validate the correctness of the original DHC algorithm and to expand the empirical evaluation beyond the scope of the original work.

3. **DHC-R:** DHC-R is a robust version of the DHC algorithm built upon DHC-ours. It is a novel algorithm that addresses the challenge of complete packet loss.

By employing these three algorithms, DHC-paper, DHC-ours and DHC-R, we aim to comprehensively evaluate and enhance the performance of the DHC algorithm in various settings. In the sections below, we present the architecture of these algorithms and discuss the training process employed.

4.1 Architecture

DHC-paper and DHC-ours

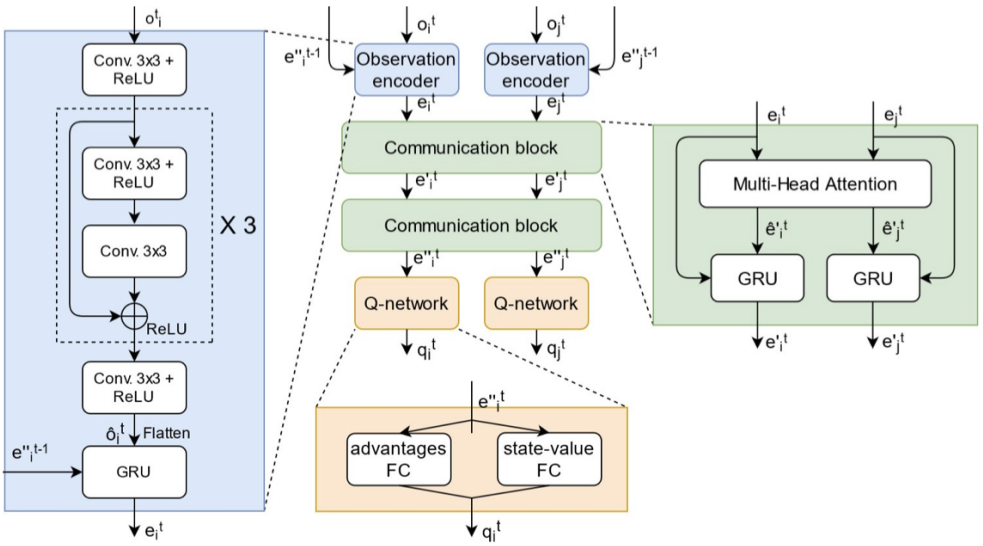
The original DHC algorithm, DHC-paper, as well as DHC-ours, share the same architecture which consists of three main components:

1. **Observation encoder:** receives stacked matrixes that represent a single observation of an agent and outputs an embedding. It consists of eight convolutional layers organized into three residual blocks and two additional independent layers, and a GRU. The GRU, in addition to the convoluted observation, receives the result of the communication from the previous step as a hidden state.

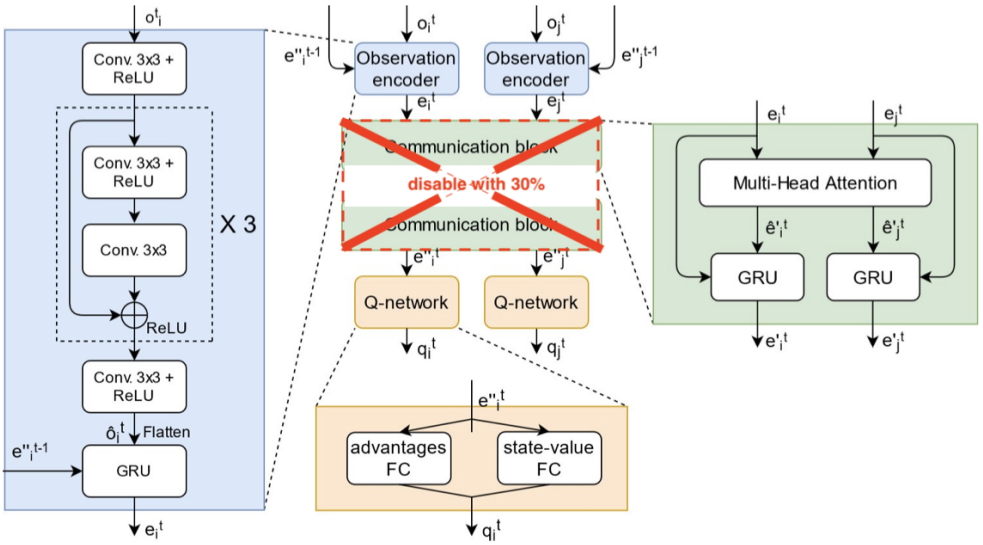
2. **Communication block:** is an attention-based neural network that outputs an embedding that incorporates information from the two nearest neighbors, if any. Each agent sends the result of its observation encoder to every other agent inside its field of view. When messages are received by an agent, only those from the two nearest neighbors are selected for further processing. Then, the selected observations are passed through two communication blocks consisting of a multi-head attention and a GRU.

3. **Q-network:** consists of two heads, the advantage and state-value (see section 2.4 for more information). It combines their outputs to approximate the value of the Q-function. The final encoding is passed through the Q-network, which calculates the

value $V(s)$ and a vector of advantages $\{A(s, a_i)\}_i$ for every action a_i. These values are then summed to obtain a vector $\{Q(s, a_i)\}$.



(a) DHC architecture (DHC-paper and DHC-ours); image from "Distributed Heuristic Multi-Agent Path Finding with Communication".



(b) DHC-R (robust DHC) architecture.

Figure 4.1: DHC and DHC-R architecture.

For a visual representation of the original architecture, refer to fig. 4.1a.

DHC-R

DHC-R, the robust version of DHC, follows the same architecture as DHC-paper and DHC-ours but introduces randomness in the communication block. During training, there is a 30% chance of disabling the communication block, simulating communication failure scenarios. We propose this algorithm due to the anticipated poor performance of classical DHC in adversarial scenarios, particularly in cases involving communication loss. The architecture of DHC-R is depicted in fig. 4.1b.

4.2 Observations

In this section, we provide an overview of the approach employed in the original paper [1] as well as in all three of our implementations. We also delve into the implementation details.

Observations are represented by a $6 \times N \times N$ tensor, with $N \times N$ being the FOV of the agent. The tensor uses six channels to encode the following information:

- **Binary mask encoding agents inside the FOV:** a value of 1 represents the position of an agent, while 0 denotes an empty position;
- **Binary mask encoding obstacles inside the FOV:** a value of 1 represents an obstacle, while 0 marks an empty cell;
- **Four heuristic channels**: a representation of the potential direction an agent should take to get closer to its target.

Each heuristic channel indicates whether an agent gets closer to its goal by taking a certain action. Refer to fig. 4.2 for an example.

Heuristic channels are necessary as the agents do not rely on imitation learning and need to acquire navigation skills. These channels provide agents with information about the location of the goal which helps them to navigate towards it.

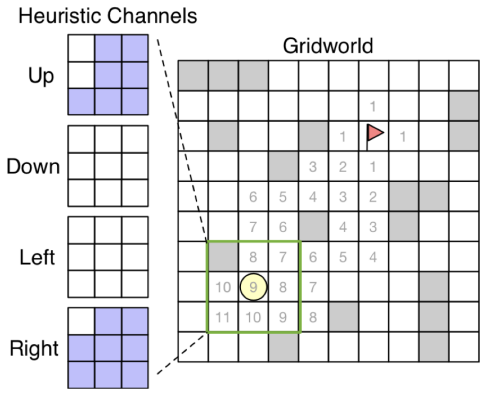


Figure 4.2: Heuristic channel; image from "Distributed Heuristic Multi-Agent Path Finding with Communication".

4.3 Training

In the original paper, the DHC algorithm was trained on maps with randomly generated obstacles. To train DHC-R and DHC-ours, we utilize the same type of maps, which we will refer to as *random maps*. We generate these maps and corresponding scenarios using the `GenerateRandomScen` procedure, presented in algorithm 1. This procedure is discussed in detail in this section.

- First, an empty matrix $M \times M$ is created;
- Next, for each cell in the map, an obstacle marked with 1 is generated with probability p, later referred to as *density*. If an obstacle is not generated, which happens with probability $1 - p$, the cell remains empty (fig. 4.3);
- The map is then partitioned into individual connected components (fig. 4.4);
- If the resulting list does not have a component with ≥ 2 nodes, the process is repeated from the beginning.

If the list contains a component that is ≥ 2 nodes, the following procedure is run N times to generate the start and end points for each agent:

- One of the empty cells is uniformly selected from any component of size ≥ 2;
- The component to which the cell belongs is determined;

Algorithm 1 `GenerateRandomScen`: Generating Random Maps and Scenarios

Input: size of map M, density of obstacles p, number of agents N
Output: Map and N pairs of start and end points for each agent
procedure GENERATE_RANDOM_MAPS_AND_SCENARIOS(M, p, N)
    Create an empty map Map of size $M \times M$
    Add obstacles to the Map with probability p
    Partition the Map into individual connected components
    $start\_end\_pairs \leftarrow \emptyset$
    for $i \leftarrow 1$ to N **do**
        if no connected component with ≥ 2 nodes **then**
            Output GENERATE_RANDOM_MAPS_AND_SCENARIOS(M, p, N)
        end if
        $component \leftarrow$ select a random component with ≥ 2 nodes from the list of connected components
        Select two random cells *start* and *end* inside *component*
        $start\_end\_pairs$.append((*start*, *end*))
        Remove *start* and *end* from *component*
    end for
    Output Map, $start\_end\_pairs$
end procedure

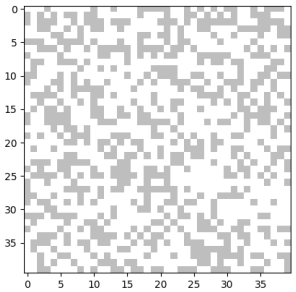


Figure 4.3: Map with obstacles (shown in gray).

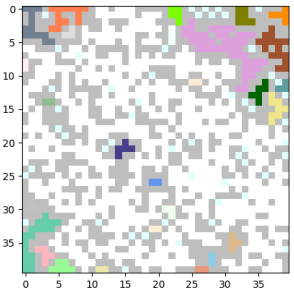


Figure 4.4: The map is partitioned into several connected components, each marked in its own color.

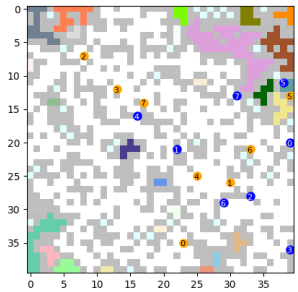


Figure 4.5: The starting and ending positions are generated inside one component.

- Two random cells are generated inside this component; these cells correspond to the starting and ending positions of the agent (fig. 4.5);
- The chosen cells are removed from the component so that they cannot be selected again.

In comparison to the original implementation, we speed up the `GenerateRandomScen` procedure by up to 400 times for large map instances. It is guaranteed that each agent can reach its end point when no other agents are present when the described algorithm is applied. However, to speed up the generation process, the authors do not validate that each end goal can be reached from the starting point when other agents are present, and we employ the same strategy. As a result, a situation depicted in fig. 4.6 may arise. However, if the target becomes unreachable for one of the agents, the experimental results of different algorithms would be equally impacted since the test set is generated beforehand.

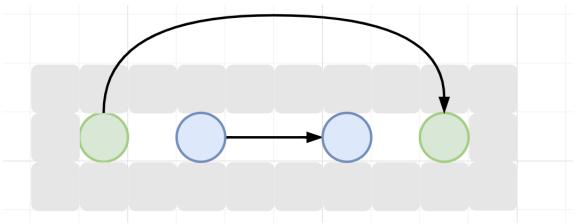


Figure 4.6: An example of a generation where the goal of the green agent remains unreachable.

The loss function is a multi-step TD error:

$$\mathcal{L}(\theta) = Huber(G_t - Q(s_t, a_t, \theta), \kappa)$$

where G_t is the total return of an agent, $G_t = r_t + r_{t+1}\gamma + \cdots + \gamma^n Q(s_{t+n}, a_{t+n}, \bar{\theta})$, and $\bar{\theta}$ is the parameter of a target network. Huber loss is defined as follows:

$$Huber(td, \kappa) = \begin{cases} \frac{1}{2}td^2 & \text{for } |td| \leq \kappa, \\ \kappa \cdot \left(|td| - \frac{1}{2}\kappa\right), & \text{otherwise.} \end{cases}$$

The authors utilize a reward design depicted in fig. 4.7. The same design is used in our algorithm, DHC-R.

Actions	Reward
Move (Up/Down/Left/Right)	-0.075
Stay (on goal, away goal)	0, -0.075
Collision (obstacle/agents)	-0.5
Finish	3

Figure 4.7: Reward design utilized in the original DHC paper.

The authors employ curriculum learning to handle tasks with complex configurations on large maps with a large number of agents. They start by training a single agent on a 10×10 map to learn how to navigate and reach goals in small-sized environments using generated trajectories. The policy is trained from a single-agent perspective, treating other agents as part of the environment. During training, 16 actors are set up on the CPU, and a single learner is set up on the GPU. The actors generate episodes with a copy of the global network, which are collected into a shared prioritized replay buffer. An ϵ-greedy exploration strategy is used during training, where each actor chooses a random action with a probability ranging from 0.0008 to 0.4, with actor i having a probability of $0.4^{1+7i/(num\_actors)-1}$. During inference, the action with the maximum Q-value is selected. The learner uses the most useful experience from the buffer to update the priorities and weights of the shared network. To implement curriculum learning, a list of settings in the form of (Map size, Number of agents) is initialized with $(10, 1)$. As actors interact with the environment, they store states, actions, rewards, and any other useful information obtained from the environment in the local buffer. Every 400 steps, the collected transitions are added to the global prioritized buffer. Every 10 seconds, statistics are calculated for each setting based on the information in the global buffer. If the success rate for a specific configuration exceeds 90%, the list of settings is extended with two additional configurations: one with the same number of agents but a larger map (10 cells added to each dimension), and the other with the same map size but one additional agent added.

```

class Learner:
    model = initialize_model()
    target_model = copy_model(model)

    def train():
        steps = 0
        while not is_done(global_buffer) and steps < MAX_STEPS:
            steps += 1
            data = sample_data(global_buffer)
            target_q_value = predict(target_model, data)
            q_value = predict(model, data)
            td_error = get_td_error(q_value, target_q_value, data)
            loss = calculate_huber_loss(td_error)
            optimize(model, loss)
            update_priorities(global_buffer, td_error)
```

```
            update_target_network(target_model, model)
```

Listing 4.1: DHC Learner

```

class Actor:
    def reset_env():
        map_size, num_agents = uniformly_choose_env_settings()
        prob = choose_prob_from_triangular(low=0, peak=0.5)
        env = setup_env(map_size, num_agents, prob)  // Calls GenerateRandomScen
        return env, observe(env)

    def run():
        steps = 0
        env, observation, positions = reset_env()
        local_buffer = create_local_buffer()
        while True:
            steps += 1
            actions, q_value = get_model_prediction(
                model, observation, positions,
            )
            with probability = eps:
                actions[0] = choose_random_action()
            info = do_env_step(env, actions)
            add_to_local_buffer(local_buffer, q_value, actions, info)
            if episode_finished() or steps >= MAX_STEPS:
                copy_to_global(local_buffer, global_buffer)
                steps = 0
                env, observation, positions = reset_env()
                local_buffer = create_local_buffer()
            if steps % 400 == 0:
                load_weights_from_learner()
```

Listing 4.2: DHC Actor

```
def DHC():
    env_settings = [(10, 1)]
    global_buffer = create_global_buffer()
    actors = setup_actors(16)
    for actor in actors:
        run(actor)
    learner = create_learner()
    train(learner)
    every_N_steps:
        CSRs = evaluate(global_buffer, env_settings)
        for CSR, map_size, num_agents in CSRs, env_settings:
            if CSR_over_last_200_runs >= 0.9:
                if map_size + 10 <= MAX_MAP_SIZE:
                    add_to_env_settings(
                        env_settings, map_size + 10, num_agents,
                    )
                if num_agents <= MAX_NUM_AGENTS:
                    add_to_env_settings(
                        env_settings, map_size, num_agents + 1,
                    )
```

Listing 4.3: DHC Algorithm

The described approach is outlined in pseudo-code in listing 4.1, listing 4.2 and listing 4.3.

The training stops either when the maximum possible configuration is reached and successfully solved, which is determined by a CSR exceeding 90% of the last 200 runs, or when 600,000 training steps have been completed. We train each algorithm on a single A100 GPU, and the training takes around 3-4 full days.

Chapter 5

Empirical Evaluation

This chapter presents the experimental setup and results of our study. The accompanying code can be found at https://github.com/acforvs/dhc-robust-mapf.

5.1 Random Maps

We start by replicating the paper and comparing our implementation, DHC-ours, with the weights provided by the authors, DHC-paper. To evaluate the performance of the algorithm, we generate a set of 200 tests on an $M \times M$ random map with density p using the procedure `GenerateRandomScen` explained in detail in algorithm 1. In each test, agents act in parallel by choosing one of five allowed actions: move left, right, up, down, or stay. Upon reaching the goal, an agent remains stationary and does **not** receive a new target. Note that an agent can still communicate with other agents to coordinate movements, such as stepping aside to allow another agent to pass. Execution stops either when all agents have reached their respective targets or upon reaching a terminal episode, which is typically set to 256, but may be set to 512 or 1024 depending on the complexity of the map. We report an average metric across all 200 runs.

The original paper reports results for fixed-density (0.3) maps of size 40×40 and 80×80, with varying numbers of agents (4, 8, 16, 32 and 64). To validate and extend these results, we generate test tasks with all possible combinations of the following parameters:

- Map size $\in 10 \times 10, 20 \times 20, 40 \times 40, 80 \times 80$;
- Density $\in 0.1, 0.3$;

Table 5.1: Maps used for evaluation: DHC-ours / DHC-paper. In the evaluation setup, test cases marked in red represent scenarios specific to our setup. Test cases marked in blue indicate the common configuration used for evaluation. Additionally, gray is used to mark parameters that were not tested in this evaluation.

Density	Map size	Number of Agents				
		4	8	16	32	64
0.1	10×10	+/-	+/-	+/-	+/-	-/-
	20×20	+/-	+/-	+/-	+/-	-/-
	30×30	+/-	+/-	+/-	+/-	-/-
	40×40	+/-	+/-	+/-	+/-	-/-
	80×80	+/-	+/-	+/-	+/-	-/-
0.3	10×10	+/-	+/-	+/-	+/-	-/-
	20×20	+/-	+/-	+/-	+/-	-/-
	30×30	+/-	+/-	+/-	+/-	-/-
	40×40	+/+	+/+	+/+	+/+	+/+
	80×80	+/+	+/+	+/+	+/+	+/+

- Number of agents $\in 4, 8, 16, 32$.

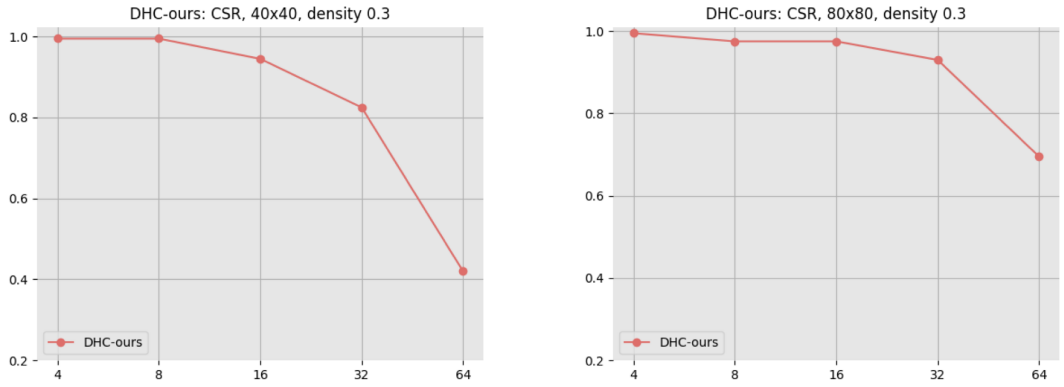
The difference in evaluation setup between our implementation DHC-ours and the original DHC-paper implementation is outlined in table 5.1.

As we can see from fig. 5.2 and fig. 5.1, our implementation of the DHC algorithm, DHC-ours, is similar to the original implementation DHC-paper and can be used in the next sections for a more thorough empirical evaluation.

Results

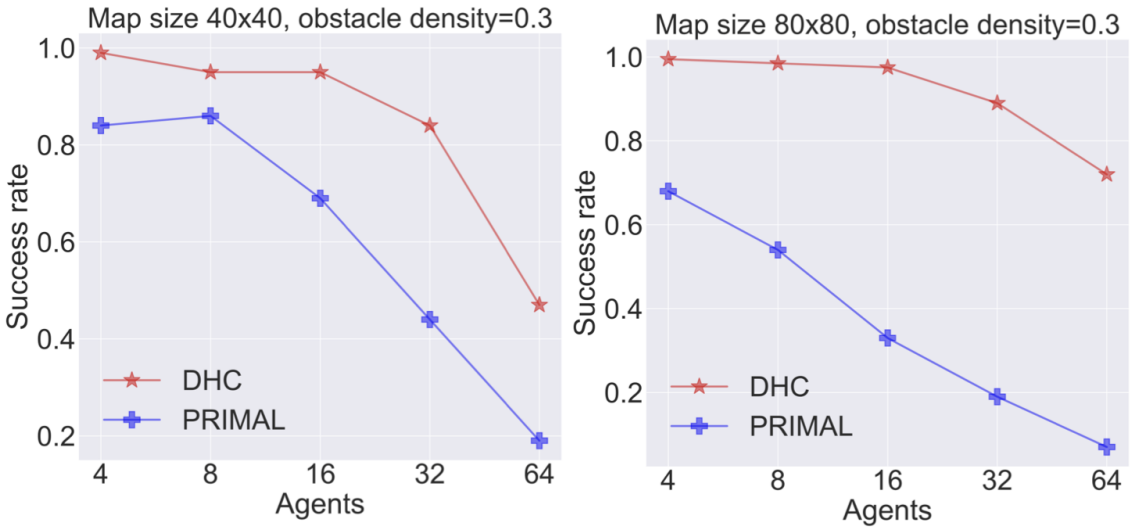
Table 5.2 summarizes the performance of the DHC-ours algorithm on the test set of 200 randomly generated tasks. The evaluation uses three metrics: CSR, ISR and makespan. For each metric, we report the mean value averaged over 200 test cases on a single map. An example of a single test invocation can be found in fig. 5.3.

Moreover, we observe that as the complexity of the map increases, the performance



(a) 40×40, density 0.3. (b) 80×80, density 0.3.

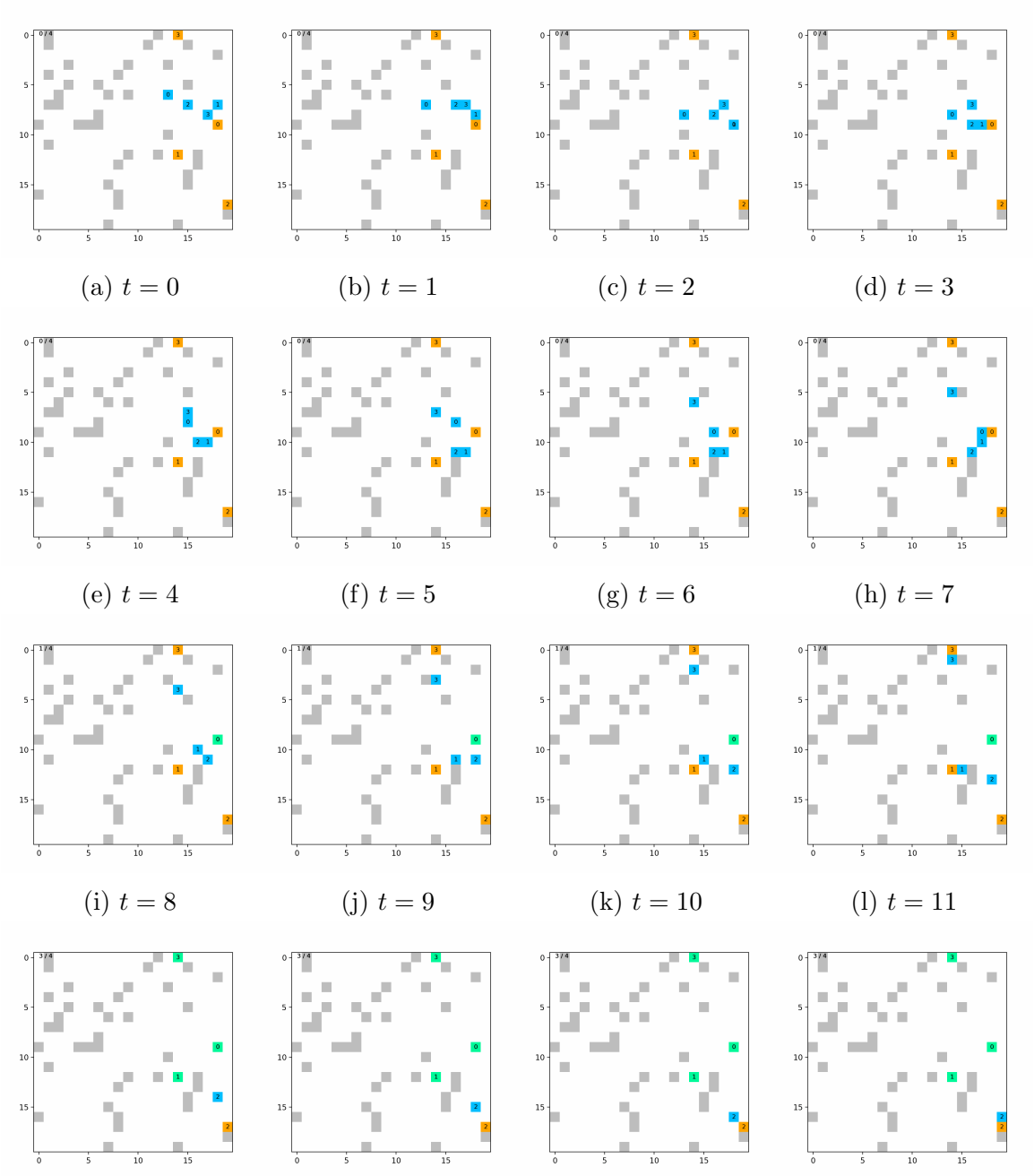
Figure 5.1: DHC-ours, CSR, density 0.3.



(a) 40×40, density 0.3. (b) 80×80, density 0.3.

Figure 5.2: DHC-paper, CSR, density 0.3. Plots from "Distributed Heuristic Multi-Agent Path Finding with Communication".

of the algorithm decreases (see 10×10 map with a density of 0.3 and 16 agents as an example). This is expected, as the number of agents increases or the map size decreases, the coordination and communication between agents become more difficult, and as the map size increases, the agents need to navigate longer distances. However, while the CSR experiences significant drops in some cases, such as the 10×10 map with a density of 0.1 and 32 agents, the ISR remains consistently high, indicating that the algorithm can still effectively coordinate agents even when not all agents are successful in reaching their targets.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$

(e) $t = 4$ (f) $t = 5$ (g) $t = 6$ (h) $t = 7$

(i) $t = 8$ (j) $t = 9$ (k) $t = 10$ (l) $t = 11$

(m) $t = 12$ (n) $t = 13$ (o) $t = 14$ (p) $t = 15$

Figure 5.3: Four agents successfully coordinate to reach their respective targets on a map with dimensions of 20×20 and a density of 0.3. The agents are denoted by blue markers, while their targets are indicated by orange markers.

Overall, these results suggest that DHC with communication is a promising approach for solving the multi-agent coordination problem on randomly generated maps.

Table 5.2: Performance results for random maps with 200 test cases, DHC-ours.

Map configuration		Density 0.1			Density 0.3		
Agents	Map	CSR, %	ISR, %	Makespan	CSR, %	ISR, %	Makespan
4	10×10	99.50	99.80	13.32	97.05	99.38	23.35
	20×20	99.50	99.88	13.32	99.50	99.88	29.63
	40×40	99.50	99.88	49.44	99.50	99.88	56.58
	80×80	97.50	99.25	110.76	99.50	99.88	103.18
8	10×10	100.00	100.00	17.88	89.50	97.19	56.77
	20×20	100.00	100.00	26.65	100.00	100.00	25.00
	40×40	99.50	99.94	60.22	99.50	99.94	68.81
	80×80	99.50	99.94	103.72	97.50	99.69	136.97
16	10×10	96.50	99.69	37.75	40.00	86.50	185.13
	20×20	99.50	99.94	33.66	89.00	98.41	77.02
	40×40	100.00	100.00	61.19	94.50	99.62	127.26
	80×80	98.50	99.91	127.33	97.50	99.84	145.68
32	10×10	15.50	89.88	239.41	-	-	-
	20×20	98.50	99.94	45.65	38.50	91.70	195.77
	40×40	100.00	100.00	64.60	82.50	99.30	255.28
	80×80	99.00	99.97	134.12	93.00	99.70	203.62

Maps with a size of 10×10, 32 agents and a density of 0.3 were not tested, since out of the expected 70 empty cells, 64 of them must be used for starting and ending positions, leaving only a small number of cells for navigation.

5.2 Out-of-Distribution Testing on MovingAI Maps

Results of this section are presented in table 5.3.

Adaptability to unfamiliar environments is a critical factor for running the algorithm in the real world. The environment may differ not only in structure but also in minor details. In the previous section, we tested the algorithm's performance on maps with similar structures to the training maps, with small variations in parameter values. This section aims to assess

the algorithm's generalizability on non-randomly structured maps, which is an essential test for evaluating the behavior OOD.

For this study, we select multi-agent scenarios from the MovingAI [32] collection, which includes maps with different structures and characteristics. Specifically, we chose four maps from this collection: lak303d.map, den520d, Berlin_1_256.map and ht_chantry.map. These maps are displayed in fig. 5.4.

(a) lak303d (b) den520d (c) Berlin_1_256 (d) ht_chantry

Figure 5.4: Used MovingAI maps.

The MovingAI website offers various scenarios for each map, where each scenario consists of a series of rows representing the start and goal positions for a single agent. Typically, these scenarios are used to evaluate single-agent performance. To generate test sets for multi-agent pathfinding problems with N agents, we selected consecutive batches of N rows from each scenario, as illustrated in fig. 5.5.

	ID	Map	W	H	X0	Y0	X1	Y1	Dist
Test 1	25	den520d.map	256	256	146	105	104	158	101.0832611
	85	den520d.map	256	256	124	13	8	214	343.3502883
	31	den520d.map	256	256	97	52	206	82	125.5685424
	46	den520d.map	256	256	109	46	198	169	185.6396102
Test 2	20	den520d.map	256	256	150	30	197	84	81.66904755
	84	den520d.map	256	256	14	213	105	41	339.0660171
	18	den520d.map	256	256	193	85	186	148	72.28427124
	53	den520d.map	256	256	241	199	208	37	214.9238815
	13	den520d.map	256	256	124	56	157	53	54.97056274
	26	den520d.map	256	256	73	58	148	100	104.0121933

Figure 5.5: Creating a multi-agent scenario using the .scen file from MovingAI.

During the experiments, we set the maximum episode length to 1024 steps. As shown in table 5.3, the value of ISR consistently exceeded 90%, even on maps with narrow passages, long obstacles and structures completely different from those of random maps. However, while most agents individually achieved their goals, the collective metric on this set of experiments was significantly worse than on random maps. These results suggest that communication can sometimes mislead an agent and negatively affect cooperation. To further investigate the importance of communication in navigation on different maps, we analyzed performance under the following settings:

- Small hand-drawn maps where cooperation is necessary to reach the goals;
- Small manually drawn maps where communication and cooperation should not affect reaching the goal, but the path itself may be long;
- Complete rejection of communication during test invocation.

The next section describes our findings in these scenarios.

5.3 Cooperation/Navigation-Intensive Maps

This section presents the results of testing the algorithm on custom maps specifically designed to challenge the agents' abilities in certain tasks, such as navigating through a narrow corridor to reach their goals. All scenarios on these maps were designed to have a feasible path for all agents to reach their respective targets.

Navigation-Intensive Scenarios

This section evaluates the agents' ability to navigate in environments with long obstacles without visible narrow spaces. To assess this, we manually created three maps, depicted in fig. 5.6. For each map, we randomly generated 100 test cases by selecting $2 \cdot N$ free cells at random, where the first N represents the starting positions and the last N indicates the targets.

The results of these tests, along with those conducted on MovingAI game maps, demonstrate the agents' proficiency in navigating through environments where communication is less critical. Table 5.4 summarizes the findings.

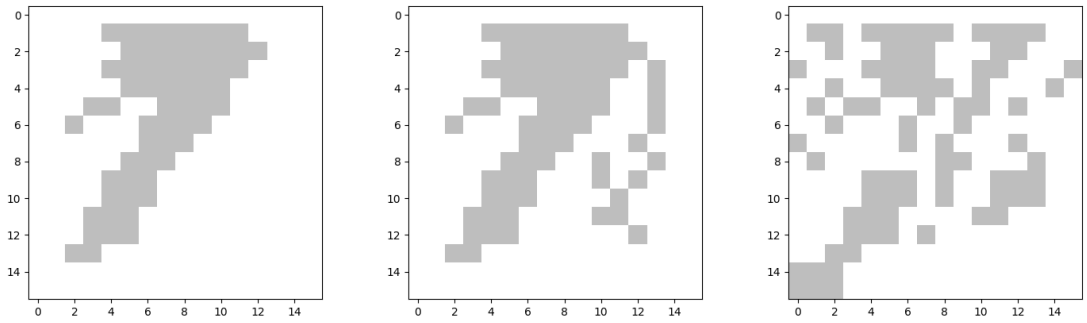
Table 5.3: Performance results on MovingAI maps (*max_episode_len* = 1024).

Agents	Map	Map size	CSR, %	ISR, %	Makespan
1	lak303d.map	194 × 194	99.84	99.84	258.41
	den520d.map	256 × 256	99.06	99.06	214.98
	Berlin_1_256.map	256 × 256	99.22	99.22	167.35
	ht_chantry.map	141 × 141	99.57	99.57	115.90
8	lak303d.map	194 × 194	100.00	100.00	473.58
	den520d.map	256 × 256	92.50	98.91	426.06
	Berlin_1_256.map	256 × 256	93.75	99.22	232.03
	ht_chantry.map	141 × 141	96.49	99.56	233.74
32	lak303d.map	194 × 194	80.00	97.97	681.15
	den520d.map	256 × 256	80.00	98.75	539.25
	Berlin_1_256.map	256 × 256	75.00	98.75	397.00
	ht_chantry.map	141 × 141	64.29	97.77	533.43
64	lak303d.map	194 × 194	70.00	95.78	773.10
	den520d.map	256 × 256	50.00	96.41	753.80
	Berlin_1_256.map	256 × 256	50.00	96.09	625.40
	ht_chantry.map	141 × 141	30.00	92.86	843.14

Table 5.4: Performance results on maps with long obstacles (100 tests, max_episode_length of 256).

Agents	Map	Map size	CSR, %	ISR, %	Makespan
8	long-obstacle-1.map	16 × 16	95.0	99.0	44.8
	long-obstacle-2.map	16 × 16	99.0	99.9	38.1
	long-obstacle-3.map	16 × 16	88.0	98.0	67.5

While the algorithm performed well on most of the maps, a potential issue was observed on the long-obstacle-3 map (demonstrated in fig. 5.7).



(a) long-obstacle-1 (b) long-obstacle-2 (c) long-obstacle-3

Figure 5.6: Maps featuring long obstacles.

In cases where an agent has to choose between a shorter path that requires communication and a longer path that does not, the agent tends to opt for the shorter path. However, if communication does not contribute to reaching the goal, the agent can get blocked and fail to reach its destination using either path.

Cooperation-Intensive Scenarios

To evaluate the agents' communication and cooperation skills, we created an 8×8 map with a narrow corridor (width 1), as shown in fig. 5.8. In the middle of the corridor, there is a dead-end branch that agents can use to let other agents pass. We also created 10 navigation scenarios, some of which involve goals located in the dead-end branch. These scenarios are specifically designed to test the agents' ability to coordinate with each other to navigate through the narrow corridor and reach their respective goals.

The testing results revealed scenarios where communication between agents did not improve their performance. As shown in fig. 5.9, agents are stuck in a situation where only one agent (blue) has not yet reached its goal (marked orange), while the other agents (green) have already done so. The communication links between agents, represented by arrows in the figure, are ineffective in this scenario because the only agent capable of moving away (number 6) is transmitting information to agent 0 (marked in red) which is irrelevant to resolving the conflict. Agent 3, which is blocked, receives information from other blocked agents (numbers 2 and 7) to make its decisions.

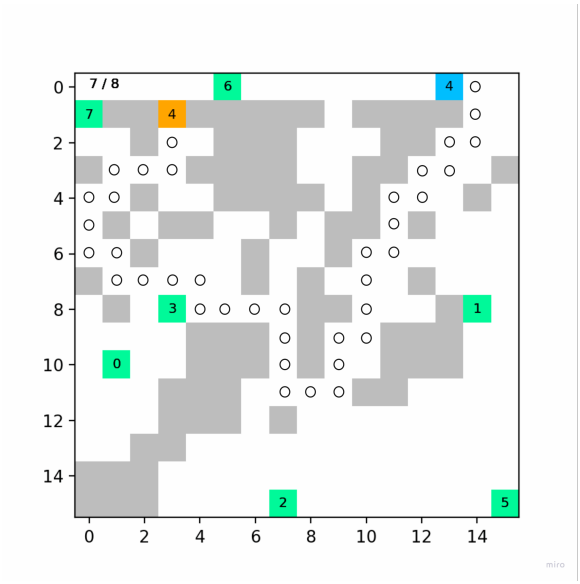


Figure 5.7: An example scenario where the blue agent 4 is unable to reach its target by taking the short path due to being blocked by agent number 6. Despite having an alternative longer path available (marked on the map), the agent does not attempt to navigate through it, resulting in failure to reach the goal.

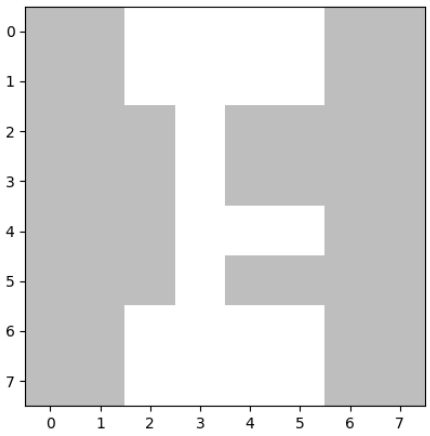


Figure 5.8: Structure of the custom map.

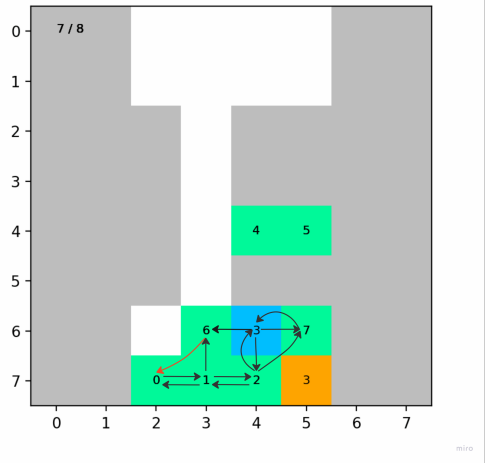


Figure 5.9: Blocked agent 3 unable to receive vital navigation information.

The results presented in table 5.5 provide support for our hypothesis, stated at the end of section 5.2, that communication may mislead agents. These findings prompt us to further investigate the impact of fully disabling communication during test invocations. Consequently, we proceed with empirical evaluations in this particular setting.

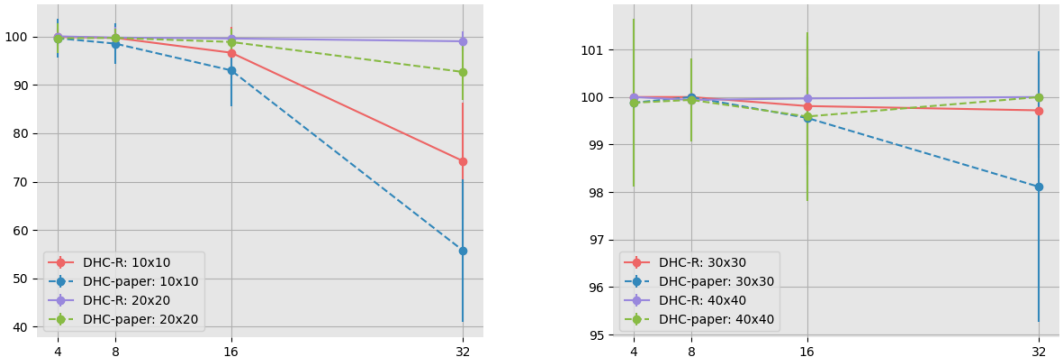
Table 5.5: Performance results on custom maps requiring cooperation (10 Tests, *max_episode_len* = 256).

Agents	Map	Map size	CSR, %	ISR, %	Makespan
8	corridor-2-rooms.map	8 × 8	27.27	73.86	203.45

5.4 Communication Failure

In real-world scenarios, perfect communication between agents cannot always be guaranteed. Weather conditions, deceptive agents in the environment, or the malfunction of an agent can result in the loss of data packets [messages]. As communication is the primary tool for cooperation in this algorithm, the loss of messages can potentially impact the performance of the system. This section analyzes the algorithm's performance when messages are completely lost during deployment of the algorithm, i.e., when agents cannot communicate with each other at all. Since this is an extreme case of communication disruption, it is the most interesting setting to consider. We also suggest ways to improve the system's performance in this scenario.

To analyze the impact of message loss on the performance of the algorithm, we evaluate two different algorithms: DHC-R and DHC-paper.

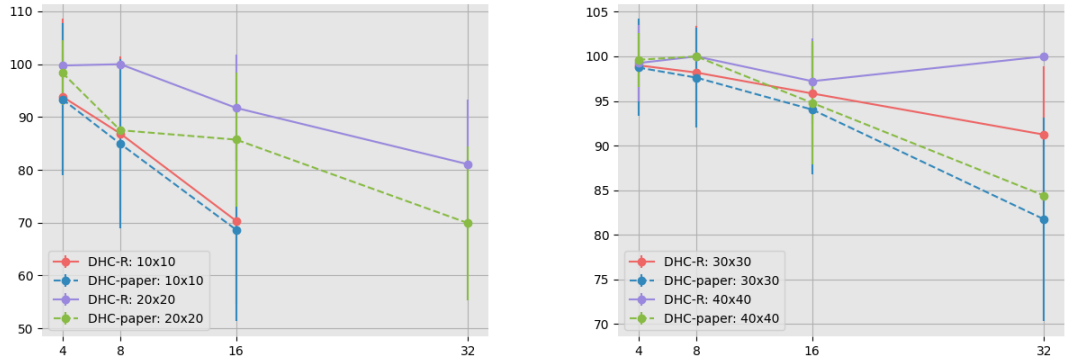


(a) Maps 10 × 10 and 20 × 20.

(b) Maps 30 × 30 and 40 × 40.

Figure 5.10: DHC-R vs. DHC-paper, density 0.1. ISR.

After completing the training process which was described chapter 4, we evaluated the performance of two algorithms on randomly generated maps with varying obstacle densities,



(a) Maps 10×10 and 20×20.

(b) Maps 30×30 and 40×40.

Figure 5.11: DHC-R vs. DHC-paper, density 0.3. ISR.

where message passing was not allowed at all. We used the same set of scenarios as in section 5.1. Plots of the individual success rates for each agent in scenarios with obstacle densities of 0.1 and 0.3 are presented in fig. 5.10 and fig. 5.11, respectively. Full results are presented in appendix B table B.1 and table B.2 for obstacle densities of 0.1 and 0.3, respectively.

As evident from the results in fig. 5.10, DHC-R demonstrates significant improvement over DHC on small maps with a density of 0.1, as observed by the 20% increase in performance indicated by the red line compared to the dashed blue line. Similarly, in fig. 5.11, a comparison between the green dashed line (representing the performance of DHC-paper) and the purple line (representing the performance of DHC-R) on 20×20 maps with density 0.3 reveals a notable difference of nearly 10%. The same trend can be observed in the right sub-figure with the red and blue lines

The suggested training cycle significantly improves the performance of the algorithm under message loss. In settings where agents do not expect message loss to occur, they may fail to reach their targets completely, as shown in the example of navigation presented in appendix A. The new training cycle mitigates these issues.

Chapter 6

Conclusion and Future Work

This thesis has achieved several objectives:

1. Chapter 2 provided the necessary background information for multi-agent pathfinding research.

2. Chapter 3 reviewed the state-of-the-art solutions in the field of multi-agent pathfinding, identifying a lack of experiments on generalization, a crucial factor for the real-world application of RL-based algorithms. Additionally, it highlighted that the decentralized RL methods for multi-agent pathfinding have not been thoroughly explored in academic research creating a gap in the literature.

3. Chapter 4 provided detailed implementation specifics for an existing decentralized method, filling in gaps left by previous works.

4. Chapter 5 extensively analyzed the existing DHC algorithm in various scenarios. The paper's original experimental setup was replicated and extended, and novel experiments on MovingAI maps and custom maps were conducted to test agent performance in navigation- or cooperation-intensive scenarios. Furthermore, this study introduced a new experiment testing the performance of the algorithm under complete message loss, revealing a significant drop in performance. We suggested a straightforward modification to the algorithm, named DHC-R, to mitigate these issues and improve the algorithm's performance under these conditions.

To conclude, this thesis has contributed to the field of multi-agent pathfinding by con-

ducting a comprehensive analysis of the DHC algorithm and proposing modifications to enhance its performance in extreme conditions. Future work could explore the effectiveness of these modifications under varying degrees of message loss or replicate the proposed empirical setup for other state-of-the-art MAPF algorithms. Moreover, there is a possibility of exploring changes in the architecture, such as the use of transformers.

Bibliography

[1] Z. Ma, Y. Luo, and H. Ma, "Distributed heuristic multi-agent path finding with communication", in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 8699–8705. DOI: 10.1109/ICRA48506.2021.9560748.

[2] Wikipedia, *Multi-agent pathfinding — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Multi-agent%20pathfinding&oldid=1138935477, [Online; accessed 10-March-2023], 2023.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0-262-19398-1. [Online]. Available: http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[5] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search", *Artificial Intelligence*, vol. 27, no. 1, pp. 97–109, 1985, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(85)90084-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370285900840.

[6] A. Zelinsky, "A mobile robot exploration algorithm", *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, pp. 707–717, 1992. DOI: 10.1109/70.182671.

[7] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds", in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3260–3267. DOI: 10.1109/IROS.2011.6095022.

[8] C. Ferner, G. Wagner, and H. Choset, "Odrm* optimal multirobot path planning in low dimensional search spaces", in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3854–3859. DOI: 10.1109/ICRA.2013.6631119.

[9] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search", in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed., Main Track, International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 4127–4135. DOI: 10.24963/ijcai.2021/568. [Online]. Available: https://doi.org/10.24963/ijcai.2021/568.

[10] D. Silver, "Cooperative pathfinding", *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, no. 1, pp. 117–122, 2021. DOI: 10.1609/aiide.v1i1.18726. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/18726.

[11] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding", *Artificial Intelligence*, vol. 195, pp. 470–495, 2013, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2012.11.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370212001543.

[12] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding", *Artificial Intelligence*, vol. 219, pp. 40–66, 2015, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.11.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001386.

[13] E. Boyarski *et al.*, "Iterative-deepening conflict-based search", in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed., Main track, International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 4084–4090. DOI: 10.24963/ijcai.2020/565. [Online]. Available: https://doi.org/10.24963/ijcai.2020/565.

[14] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem", *Frontiers in Artificial Intelligence and Applications*, vol. 263, pp. 961–962, Jan. 2014. DOI: 10.3233/978-1-61499-419-0-961.

[15] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Efficient sat approach to multi-agent path finding under the sum of costs objective", Sep. 2016. DOI: 10.3233/978-1-61499-672-9-810.

[16] E. Lam, P. Le Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding", in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 1289–1296. DOI: 10.24963/ijcai.2019/179. [Online]. Available: https://doi.org/10.24963/ijcai.2019/179.

[17] G. Sartoretti *et al.*, "Primal: Pathfinding via reinforcement and imitation multi-agent learning", *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019. DOI: 10.1109/LRA.2019.2903261.

[18] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning", *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932–6939, 2020. DOI: 10.1109/LRA.2020.3026638.

[19] A. Singh, T. Jain, and S. Sukhbaatar, "Individualized controlled continuous communication model for multiagent cooperative and competitive tasks", in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rye7knCqK7.

[20] D. Kim *et al.*, "Learning to schedule communication in multi-agent reinforcement learning", in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=SJxu5iR9KQ.

[21] S. Q. Zhang, Q. Zhang, and J. Lin, "Efficient communication in multi-agent reinforcement learning via variance based control", in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[22] S. Q. Zhang, J. Lin, and Q. Zhang, "Succinct and robust multi-agent communication with temporal message control", in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[23] W. Li, H. Chen, B. Jin, W. Tan, H. Zha, and X. Wang, "Multi-agent path finding with prioritized communication learning", in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10 695–10 701. DOI: 10.1109/ICRA46639.2022.9811643.

[24] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized path planning", in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20, Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, 1901–1903, ISBN: 9781450375184.

[25] Q. Li, W. Lin, Z. Liu, and A. Prorok, "Message-aware graph attention networks for large-scale multi-robot path planning", *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5533–5540, 2021. DOI: 10.1109/LRA.2021.3077863.

[26] Z. Ma, Y. Luo, and J. Pan, "Learning selective communication for multi-agent path finding", *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1455–1462, 2022. DOI: 10.1109/LRA.2021.3139145.

[27] L. Chen, Y. Wang, Z. Miao, Y. Mo, M. Feng, and Z. Zhou, "Multi-agent path finding using imitation-reinforcement learning with transformer", in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 445–450. DOI: 10.1109/ROBIO55434.2022.10011833.

[28] D. Kirilenko, A. Andreychuk, A. Panov, and K. Yakovlev, *Transpath: Learning heuristics for grid-based pathfinding via transformers*, 2022. arXiv: 2212.11730 [cs.AI].

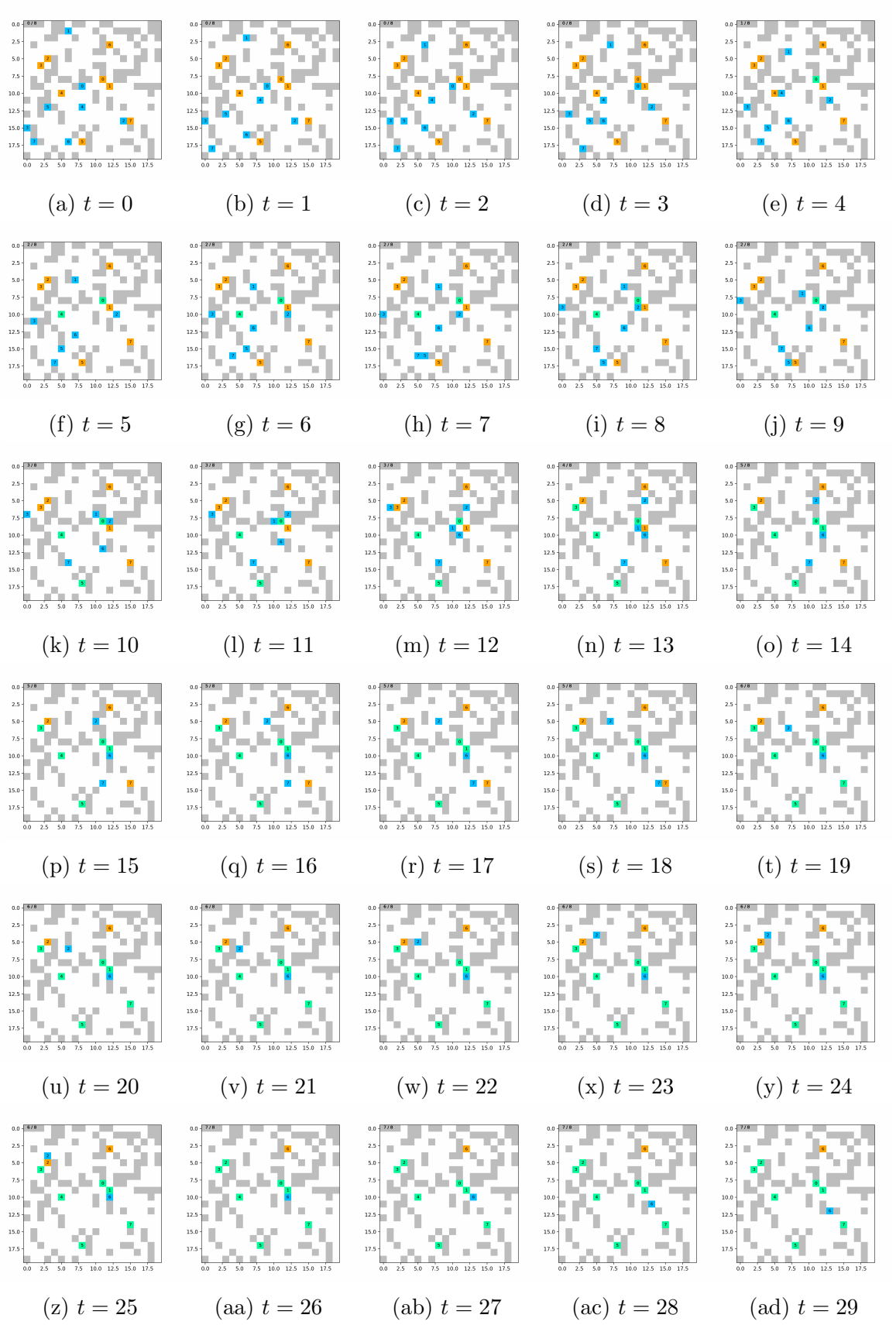
[29] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 11 272–11 281, 2021. DOI: 10.1609/aaai.v35i13.17344. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17344.

[30] D. Atzmon, R. Stern, A. Felner, N. R. Sturtevant, and S. Koenig, "Probabilistic robust multi-agent path finding", *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, no. 1, pp. 29–37, 2020. DOI: 10.1609/icaps.v30i1.6642. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/6642.

[31] M. Phuong and M. Hutter, "Formal algorithms for transformers", 2022. arXiv: 2207.09238 [cs.LG].

[32] R. Stern *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks", *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.

Appendix A: DHC-R vs. DHC-paper, Navigation



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$ (e) $t = 4$

(f) $t = 5$ (g) $t = 6$ (h) $t = 7$ (i) $t = 8$ (j) $t = 9$

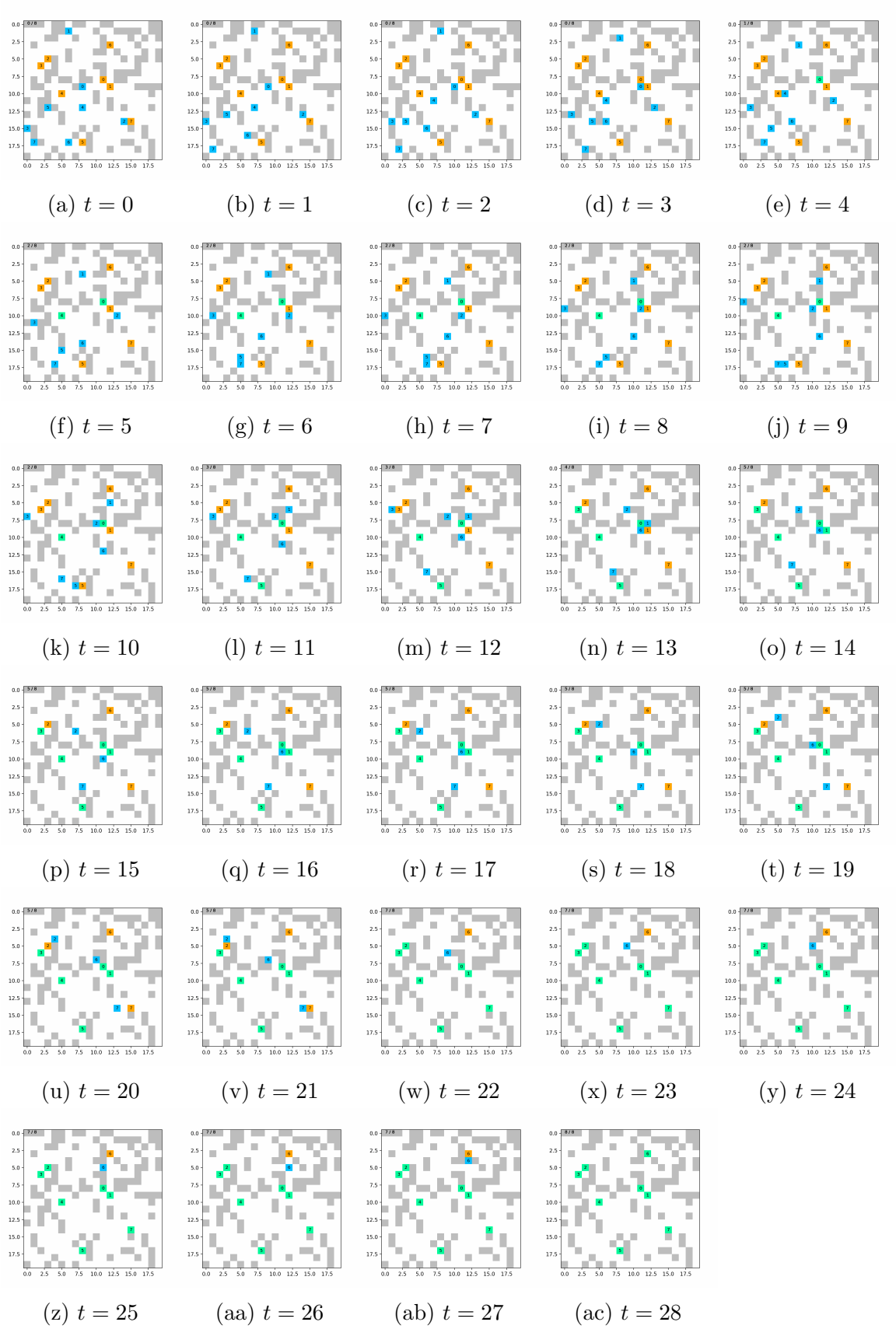
(k) $t = 10$ (l) $t = 11$ (m) $t = 12$ (n) $t = 13$ (o) $t = 14$

(p) $t = 15$ (q) $t = 16$ (r) $t = 17$ (s) $t = 18$ (t) $t = 19$

(u) $t = 20$ (v) $t = 21$ (w) $t = 22$ (x) $t = 23$ (y) $t = 24$

(z) $t = 25$ (aa) $t = 26$ (ab) $t = 27$ (ac) $t = 28$ (ad) $t = 29$

Figure A.1: Agents cannot communicate. Trained in a standard way with the ability to communicate (DHC-paper).



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$ (e) $t = 4$

(f) $t = 5$ (g) $t = 6$ (h) $t = 7$ (i) $t = 8$ (j) $t = 9$

(k) $t = 10$ (l) $t = 11$ (m) $t = 12$ (n) $t = 13$ (o) $t = 14$

(p) $t = 15$ (q) $t = 16$ (r) $t = 17$ (s) $t = 18$ (t) $t = 19$

(u) $t = 20$ (v) $t = 21$ (w) $t = 22$ (x) $t = 23$ (y) $t = 24$

(z) $t = 25$ (aa) $t = 26$ (ab) $t = 27$ (ac) $t = 28$

Figure A.2: Agents cannot communicate. Trained with potential message losses with probability of 30% (DHC-R).

Appendix B: DHC-R vs. DHC-paper, Tables

Table B.1: Communication turned off. DHC-R vs. DHC-paper. Density 0.1.

Map configuration		Metrics	Density 0.1	
# Agents	Size		DHC-R	DHC-paper
4	10×10	CSR, %	**99.50**	99.00
		ISR, %	**99.88 ± 1.76**	99.62 ± 3.94
	20×20	CSR, %	**100.00**	98.50
		ISR, %	**100.00 ± 0.00**	99.62 ± 3.04
	30×30	CSR, %	**100.00**	99.50
		ISR, %	**100.00 ± 0.00**	99.88 ± 1.76
	40×40	CSR, %	**100.00**	99.50
		ISR, %	**100.00 ± 0.00**	99.88 ± 1.76
8	10×10	CSR, %	**97.50**	88.50
		ISR, %	**99.69 ± 1.95**	98.50 ± 4.25
	20×20	CSR, %	**98.50**	97.50
		ISR, %	**99.75 ± 2.15**	99.69 ± 1.95
	30×30	CSR, %	**100.00**	100.00
		ISR, %	**100.00 ± 0.00**	100.00 ± 0.00
	40×40	CSR, %	**99.50**	99.50
		ISR, %	**99.94 ± 0.88**	99.94 ± 0.88

Continued on next page

Table B.1 continued from previous page

Map configuration		Metrics	Density 0.1	
# Agents	Size		DHC-R	DHC-paper
16	10×10	CSR, %	**64.00**	36.00
		ISR, %	**96.62 ± 5.37**	92.97 ± 7.34
	20×20	CSR, %	**93.00**	84.00
		ISR, %	**99.56 ± 1.59**	98.84 ± 2.80
	30×30	CSR, %	**97.50**	93.50
		ISR, %	**99.81 ± 1.24**	99.56 ± 1.71
	40×40	CSR, %	**99.50**	94.50
		ISR, %	**99.97 ± 0.44**	99.59 ± 1.78
32	10×10	CSR, %	**0.00**	0.00
		ISR, %	**74.23 ± 12.01**	55.73 ± 14.78
	20×20	CSR, %	**75.50**	13.00
		ISR, %	**98.98 ± 2.07**	92.64 ± 5.73
	30×30	CSR, %	**92.00**	60.00
		ISR, %	**99.72 ± 1.00**	98.11 ± 2.85
	40×40	CSR, %	**100.00**	100.00
		ISR, %	**100.00 ± 0.00**	100.00 ± 0.00

Table B.2: Communication turned off. DHC-R vs. DHC-paper. Density 0.3.

Map configuration		Metrics	Density 0.3	
# Agents	Size		DHC-R	DHC-paper
4	10×10	CSR, %	**83.50**	80.50
		ISR, %	**93.88 ± 14.68**	93.38 ± 14.46
	20×20	CSR, %	**99.00**	93.50
		ISR, %	**99.75 ± 2.49**	98.38 ± 6.16
	30×30	CSR, %	**96.00**	95.00
		ISR, %	**99.00 ± 4.90**	98.75 ± 5.45
	40×40	CSR, %	97.00	**98.50**
		ISR, %	99.25 ± 4.26	**99.62 ± 3.04**
8	10×10	CSR, %	**40.00**	36.50
		ISR, %	**86.88 ± 14.51**	84.94 ± 16.07
	20×20	CSR, %	**100.00**	0.00
		ISR, %	**100.00 ± 0.00**	87.50 ± 0.00
	30×30	CSR, %	**88.00**	83.50
		ISR, %	**98.19 ± 5.21**	97.62 ± 5.64
	40×40	CSR, %	**100.00**	100.00
		ISR, %	**100.00 ± 0.00**	100.00 ± 0.00
16	10×10	CSR, %	**3.00**	2.50
		ISR, %	**70.34 ± 15.65**	68.66 ± 17.33
	20×20	CSR, %	**42.00**	19.50
		ISR, %	**91.72 ± 10.06**	85.72 ± 12.75
	30×30	CSR, %	**56.50**	48.00
		ISR, %	**95.84 ± 5.74**	94.03 ± 7.30
	40×40	CSR, %	**66.50**	51.00
		ISR, %	**97.22 ± 4.75**	94.78 ± 6.93

Continued on next page

Table B.2 continued from previous page

Map configuration		Metrics	Density 0.3	
# Agents	Size		DHC-R	DHC-paper
32	10×10	CSR, %	-	-
		ISR, %	-	-
	20×20	CSR, %	**2.50**	0.50
		ISR, %	**81.06 ± 12.25**	69.91 ± 14.53
	30×30	CSR, %	**15.00**	1.50
		ISR, %	**91.22 ± 7.66**	81.72 ± 11.45
	40×40	CSR, %	**100.00**	0.00
		ISR, %	**100.00 ± 0.00**	84.38 ± 0.00