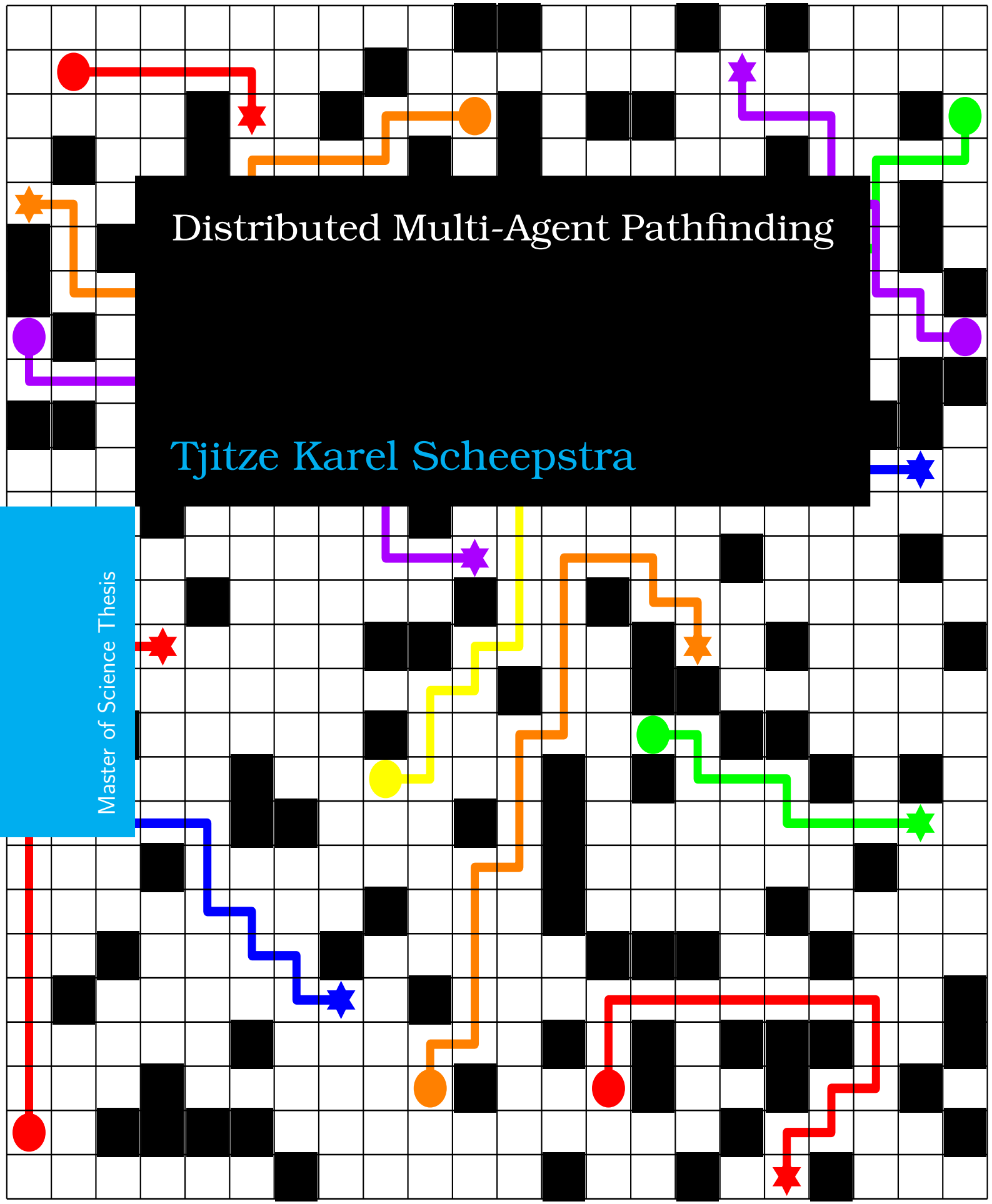


Distributed Multi-Agent Pathfinding

Tjitze Karel Scheepstra

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Distributed Multi-Agent Pathfinding

Master of Science Thesis

For the degrees of Master of Science

in **Systems and Control** and **Mechanical Engineering**
at the departments of **Delft Center for Systems and Control**
and **Maritime and Transport Technology**

at Delft University of Technology

Tjitze Karel Scheepstra

April 3, 2024

Student number	4593162
Supervisors	Dr.ir. Bilge Atasoy
	Dr.ir. Azita Dabiri
ME MSc track	MME
Report number	2024.MME.8911

Faculty of Mechanical Engineering (ME) · Delft University of Technology

Delft University of Technology
Departments of
Delft Center for Systems and Control (dcsc)
+ Maritime and Transport Technology (mtt)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical Engineering (ME) for acceptance a thesis entitled

Distributed Multi-Agent Pathfinding

by

Tjitze Karel Scheepstra

in partial fulfillment of the requirements for the degrees of

Master of Science in **Systems and Control** and **Mechanical Engineering**

Dated: April 3, 2024

Supervisor(s):

Dr.ir. Bilge Atasoy

Dr.ir. Azita Dabiri

Reader(s):

Dr.ing. Mohammad Khosravi

Abstract

Multi-agent path finding (MAPF) is the task of finding non-conflicting paths for multiple agents that operate in a environment with shared resources. Finding an optimal solution quickly becomes intractable for many applications and consequently suboptimal methods are also explored extensively in literature. This work presents the Decentralized Optimization (DECOP) algorithm: a novel receding horizon control algorithm that exploits insights from MAPF research as well as decentralized control. In the proposed framework, each travelling agent communicates with agents in its proximity to solve a local MAPF problem that considers only a selected tractable number of agents. Inter-agent cooperation and conflict free operation are induced through applying a common local optimization policy during parallel local optimization and through a subsequent path reservation scheme based on random priorities. Inter-agent communication consists of sharing respective route alternatives from which additional information with regard to an agents' entanglement can be inferred which can also be included in the local optimization cost function.
Comparative results with other decentralized algorithms show that the DECOP algorithm yields competitive results while guaranteeing conflict free operations, with limited required communication and without the need of any training time. Among many degrees of freedom to be explored further, including information about the entanglements of an agent's route alternatives in the common policy for local optimization yields an increase in performance and suggests an increased extent of induced cooperation.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the transition towards automating transport operations that take many shapes in modern day societies, many challenges remain. Advancements in communication, sensing, and computation technologies enable the replacement of human decision making by automated intelligent decision making. The main drivers behind this transition process are the increase of human safety, reduced energy consumption and monetary profits. Challenges range from including vulnerable, and in general, non-automated road users [8, 2] to coordinating hundreds of robots in automated warehouses [39]. This work aims to investigate how local coordination through limited communication can improve global network performance.
In a general sense, self-interested users utilizing a network of transportation links pose a prisoners dilemma over shared resources wherein all users will naturally execute their personal best trajectories while neglecting the demands of other users in the network. All users are ultimately forced to cooperate by the network operator to achieve conflict free network utilization, usually in a manner that can be described as passive. The (average) travel time performance of a passively controlled network will likely be longer compared to a situation where users actively cooperate as in order to reduce all users' travelling efforts, often personally non-optimal trajectories are to be executed by all users. From an individual interest perspective, such as for road users on a public road, the achieved reduction in travelling time should be distributed equally over the participating network users. To the contrary, in a network with common-interest users the network utilization is optimized according to the common cost function and the fairness of distributing travel time delay can be neglected. At the root of both problems lie the centralized optimal solutions that guide all users through a network with respect to the appropriate cost function. Due to scalability issues, a centralized approach to this problem will predictably not be tractable.

1-1 Research scope and contribution

This research aims to answer the following research question:

> *"Can a decentralized approach that applies local optimization combined with local negotiation be designed to competently solve a highly scalable multi-agent path finding (MAPF) problem?"*

The scope of this research is limited to the evaluation of a conceptional algorithm design that will be verified through simulation. Furthermore, this work aims to apply the widespread theoretical insights on solving the MAPF problem attuned to the boundary conditions of real world systems, using concepts of decentralized control theory. Especially computational and communicative hardware capabilities that are currently widely and cheaply available, the embodiment of internet of things (IoT), form an important basis of this research. Elaboration on the extent of these capabilities will be provided when required by the context. It is at least assumed that the users of a transportation network are equipped with some sort of computational and communicative hardware. This could be considered as a real world system boundary condition and will be exploited in this work.
The contribution of this work is threefold: 1) a novel framework called the Decentralized Optimization algorithm (DECOP) for decentralized MAPF that applies parallel receding horizon control is presented. 2) This work reports the first receding horizon control algorithm designed for the MAPF problem which was achieved by designing a terminal cost function that can avoid deadlocks. 3) The promising concepts that follow from route alternatives sharing that are developed in this work open up a new research area into exploiting dependency information to achieve induced cooperation through a common decentralized policy.

1-2 Outline

The remainder of this Masters Thesis is structured as follows: background information on the MAPF problem and its properties will be provided in chapter 2 together with relevant literature on MAPF algorithms. The methodology of the proposed framework will be presented in chapter 3. The empirical simulation results and algorithm evaluation are reported in chapter 4. Finally, the conclusions are drawn in chapter 5, accompanied by a discussion on the obtained results and the suggested directions for future work.

Chapter 2

Multi-Agent Path finding background and literature research

Multi-agent path finding (MAPF) is the task of finding non-conflicting paths for multiple agents that operate in a environment with shared resources. The MAPF problem arises in many real world applications such as routing of robots in warehouses [39], baggage handling [40], or even micro droplet manipulation [11]. Since the joint optimization of agent's continuous trajectories poses a very complex problem, planning is usually done on a graph based abstraction of the environment. A graph consists of nodes and edges that connect the nodes with each other. Additional features can be embedded in the graph in order to better represent the real world case which it aims to describe. For example, weights and directions can be considered for the edges of the graph. Weights on the edges could for example represent the physical distance between two nodes. Even if agents could be less restrictive about their movements in the environment, a graph abstraction can still be a useful mathematical tool. In [27] for example, a graph in the form of a grid is used to describe the 3D space in which the quadrotors can move. A MAPF algorithm comes up with an initial (feasible) solution for a continuous trajectory optimization problem in which agents are no longer bound to the edges and nodes of the 3D grid. Another example can be found in [20] where an initial solution for the coordination of non-holonomic robots is obtained by a MAPF algorithm on a discrete graph and consequently the trajectories are refined into smooth trajectories through nonlinear optimization.

2-1 Classical MAPF

The notion of *classical MAPF* as defined in [34] is a planning problem in which a node can be occupied by only one agent and all agents can either move or wait (in parallel) to a connected node in each time step. The concept of an "agent" in MAPF literature refers to the entities or users which operate in a shared environment by traversing through space over an assigned path. In control systems type of literature on the other hand, an agent usually refers to a

decision-making entity that operates within a larger system by deciding on control inputs for part of that system. As most of this work and the terminology used have their roots in MAPF literature, the term agent will be in line with the concept of an agent as used in MAPF literature. If the term agent is indeed used to designate a decision-making entity, this will be described explicitly by the context. Furthermore, all terminology of the algorithm developed in this work will be formally defined in the methodology in chapter 3.
For a system consisting of a number of k agents where $\mathcal{A} = \{a_1, \ldots, a_k\}, |\mathcal{A}| = k$ describes the set of agents, the ith agent is referred to as a_i. The input to a classical MAPF problem is the 3-tuple $\langle \mathcal{G}, s, g \rangle$. The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph with nodes $v \in \mathcal{V}$ connected through edges $e \in \mathcal{E}$. The k-tuple s defines a list of start nodes for all agents where $s_i \in \mathcal{V}$ denotes the start node of the ith agent. Similarly, the k-tuple g defines a list of goal nodes where $g_i \in \mathcal{V}$ denotes the goal node of the ith agent.
The initial simulation time is denoted as t_0 and marks the beginning of the plan execution of each agent starting at its start node. Time is furthermore discretized such that $t = t_0 + n \cdot h$ with $n \in \mathbb{N}^0$. The time step h is normally set to 1 and traversing an edge usually takes exactly one time step (for unweighted graphs). A single agent path for the ith agent is defined by π_i as a sequence of $v \in \mathcal{V}$ nodes. The single agent path π_i takes a time step n as an argument to denote the particular node v that the ith agent occupies in time step n: $\pi_i(n) = v$. The number of nodes that an agent plans to occupy (including wait actions, the start node and the goal node) is denoted as $|\pi_i|$, therefore the total number of actions in plan π_i is equal to $N_i = |\pi_i| - 1$. Agents can only traverse over edges of the graph $\mathcal{G}$ therefore $\{\pi_i(n), \pi_i(n+1)\} \in \mathcal{E}$ or alternatively an agent can wait at a node such that $\pi_i(n+1) = \pi_i(n)$.
The solution to a MAPF problem is the set of k single agent plans $\mathcal{P} = \{\pi_1, \ldots, \pi_k\}$ such that the respective paths are non-conflicting. In order to conclude that the solution of a MAPF problem is valid due to the absence of conflicts, the following conflicts are defined:

- a **node conflict**: $\pi_i(n) = \pi_j(n) \quad \forall i, j | i \neq j \in \mathcal{A}$, occurs when two agents occupy the same node in the same time step
- an **edge conflict** or **swap conflict**: $\pi_i(n) = \pi_j(n+1) \wedge \pi_i(n+1) = \pi_j(n) \quad \forall i, j | i \neq j \in \mathcal{A}$, occurs when two agents start from connected nodes and traverse over the same edge in the same time step
- a **following conflict**: $\pi_i(n+1) = \pi_j(n) \quad \forall i, j | i \neq j \in \mathcal{A}$, occurs when an agent traverses to a node that was occupied by another agent in the previous time step
- a **cycle conflict** occurs when every agent moves to a node that was occupied by another agent in the previous time step, therefore a **cycle conflict** requires a number of c of only following conflicts between a number of c agents with $c \geq 3$.

Different types of MAPF problems in practice may allow or disallow different types conflicts to occur in their solution and an algorithm is designed accordingly. Another distinction in the solving approach can be made with respect to the terminal state: whether an agent stays in the target node or disappears after a certain amount of time.
The MAPF problem is sometimes referred to as a pebble motion on graph (PMG) problem, which is an early identified variant of MAPF where only one pebble (agent) per time step is allowed to be moved to an unoccupied node. It has been proven that the PMG problem is solvable in polynomial time in [18] and it has also been proven that solution feasibility can be

checked in linear time by [14], predictably this also translates to solving MAPF problems with parallel moves [42]. Both of these algorithms were also derived by [43] for MAPF problems that allow the following, cycle, and swapping conflicts that were previously described. Furthermore, a MAPF problem is well-formed if there exists a set of plans $\mathcal{P}$ such that none of these paths cross each other [33]. Finally, a grid-based MAPF graph refers to the translation of a grid to a graph where the grid tiles are nodes and each node is connected to the adjacent up, down, right and left tile through an edge.

2-1-1 Objectives for optimal MAPF

Besides a (possibly non-unique) set of conflict free paths, a MAPF application might consider some sort of global objective or cost function. The MAPF objective is considered on a network level, as it refers to the utilization effectiveness of a composition of shared resources: the network itself. Two main objective variants are distinguished in MAPF research, namely:

- the **sum of costs (SOC) objective**: $\sum_{i=1}^{k} \sum_{n=0}^{N_i} C(\pi_i(n), \pi_i(n+1))$, which is the sum of each agent's cost of travelling the edges along the determined path. $C(\,\cdot\,,\,\cdot\,)$ denotes the cost of traversing a particular edge.
- the **makespan objective** $\max_{i \leq i \leq k} N_i$ that minimizes the amount of time (steps) required before all agents have reached their target node

The SOC objective can be regarded as the optimal solution purely from the operator perspective in terms of profit whereas the makespan objective ensures an equality objective among agents such that no agent is (significantly) worse off compared to the other agents. The SOC objective was shown to be NP-hard by proving that the distance optimal PMG problem is NP-hard [13],[28]. The PMG problem with parallel moves is also proven to be NP-hard when considering the makespan objective [35]. These objectives are also shown to have a Pareto optimal structure in [42], moreover a solution cannot (always) be optimal with respect to both objectives at the same time. It is furthermore proven that the optimal MAPF problem is NP-hard for both the SOC and makespan objective when considering at least the node and edge conflict types [42]. This makes the optimal MAPF problem intractable when a large number of agents is considered.

2-2 MAPF algorithm design strategies

Many solving strategies exists for the (optimal) MAPF problem. Finding an optimal solution with respect to one of the two introduced objectives quickly becomes intractable for many applications and consequently suboptimal solving strategies are also explored extensively in literature. Since there doesn't exist a "perfect" strategy (yet), a MAPF algorithm is usually tailored to fit its application. In general, MAPF algorithms will feature one or more of the following properties:

- **optimality**, a number of strategies is introduced to quickly obtain an optimal solution without exploring the entire search space (e.g. [30],[31]). Many algorithms also disregard the strict optimality requirement and instead look for bounded suboptimal solutions (e.g. [1],[12],[19]).

- **completeness**, a complete MAPF algorithm will always yield a feasible solution. Several algorithms exist that yield feasible solutions for any MAPF problem with at least two unoccupied nodes. Furthermore, [41] points out that feasible solutions often exist for grid-based MAPF problems that allow the cycle conflict with as many agents as nodes. A linear time algorithm to check feasibility is introduced in [44].

- **fast solving time (tractability)**, a MAPF algorithm is considered "fast" if its worst-case solving time is polynomial in the graph size and is not exponential in the number of agents [33]. This property is usually combined with the completeness property as it would otherwise be of little practical added value.

- **robustness** relates to the agent's plan execution under uncertainty. A real world application of MAPF might suffer from delays in the network as well as general uncertainties in arrival times. An MAPF planning solution can be considered robust if it is still executable under delays and uncertainties up to some extent, for example by incorporating time buffers around agents at intersections.

Another important distinction in MAPF algorithm design strategies is between compilation-based and search-based solvers. For compilation-based algorithms, the MAPF problem is reduced to a well established formalism for which an efficient solver exists [36]. Examples of such formalisms are boolean satisfiability (SAT), mixed integer linear programming (MILP), answer set programming (ASP) and constraint satisfaction (CSP). Alternatively, search-based algorithms solve the problem directly and define the solving process, treating it less like a black box [36]. Most compilation-based algorithms use makespan as their cost function, while most search-based use SOC [34].
Intuitively, one might try to solve the MAPF problem by generating single-agent shortest paths while neglecting all other agents, this will be regarded as a solution to the single-agent path finding (SAPF) problem. SAPF can be efficiently and optimally solved by graph expansion algorithms such as A* [15]. An A* graph expansion algorithm can also be used to jointly solve the (optimal) MAPF problem when considering the combined agents' search space[33]. This coupled approach is guaranteed to be feasible and optimal, it however becomes intractable with an increasing number of agents very quickly. Alternatively, in decoupled approaches, rather than expanding all possible moves, a single path is selected for each agent and conflicts are discovered through simulation. The task is now to disentangle the SAPF solutions such that no conflicts are resolved according to the MAPF objective. The most prominent example of a decoupled search-based algorithm, and considered as state-of-the-art (especially in enhanced forms), is conflict-based search (CBS) [30]. The term "coupled" is used somewhat ambiguously in MAPF literature. In this work a search-based MAPF algorithm is said to be coupled if it searches paths for all involved agents simultaneously and consequently all intermediate MAPF solutions are feasible. A decoupled search-based MAPF algorithm on the other hand usually has infeasible intermediate solutions. An intermediate solution is infeasible if the solution contains conflicts over the shared variables (the nodes and edges) and these conflicts in intermediate solutions are referred to as couplings.

2-3 Decentralized MAPF algorithms

A centralized approach to optimal multi-agent path planning suffers from tractability issues as the maximum time it could take to find an optimal solution grows exponentially with the number of agents. This motivates the search for suboptimal yet very usable algorithms for the MAPF problem. Alongside the development of (potentially bounded) centralized suboptimal algorithms (e.g. [1],[3],[7],[19]), there is a growing interest in decentralized MAPF algorithms. These algorithms apply decentralized decision-making where different decision-making agents have access to only a limited part of the global MAPF problem. Among other things, practical applicability presents a valuable benefit of this approach.
An ongoing trend in control of large scale systems is the implementation of decentralized control in order to reduce controller complexity. An encouraging preliminary for the application of decentralized control strategies in a system is if certain subsystems can be identified that have a relatively weaker coupling with other parts of the system. In decentralized control, a control policy is developed for each subsystem while treating any coupling with the rest of the system as a disturbance. A number of search-based MAPF algorithms divide agents into subsets of agents that are based on limited or no dynamic coupling and jointly search for solutions of these subsets. The remaining path entanglements or couplings can be regarded as disturbances between subsystems (subsets of agents) which are to be resolved in some manner. A particular challenge in MAPF is the fact that resolving couplings between any subsystems might create new couplings with other subsystems. For example in Independence detection (ID), the algorithm aims to find decoupled subsets of agents in a given MAPF problem and finds joint solutions for these subsets. In worst-case no dynamic independence can be identified and the ID algorithm adds all agents to the same subset such that the MAPF problem becomes a regular centralized MAPF problem. In suboptimal extensions of ID, the search is heuristically guided towards finding solutions that are decoupled for either the same or a slightly higher total solution cost. For example optimal anytime algorithm (OAt) introduces a maximum group size number and favours finding paths that are conflict free between groups. The algorithm then iteratively improves the MAPF solution quality by increasing the group size, thereby finding jointly optimal solutions for agents in a subset. Prioritized planning is another example of a decentralized MAPF search algorithm: the agents are placed within a hierarchy by a higher level control entity and subsequently plan their paths while treating the other (previously planned) agents paths as disturbances. Effectively, the size of the subsets in prioritized planning is one and couplings between agents are avoided by introducing subsequent planning on a random priority basis.
All aforementioned examples use some sort of higher level controller that is able to (successfully) identify agent subsets based on their start and goal locations or is able to determine a hierarchy among the agents. Moreover, a higher level control entity is used to enforce cooperation between different types of subsystems. The full information requirement of this higher level control entity is likely to be intractable or infeasible in practical applications. A decentralized scheme for MAPF that doesn't have a higher level control entity can be considered a distributed MAPF algorithm if path decision-making is allocated to the (travelling) agents which additionally utilize inter-agent communication links. In distributed control, there is usually no higher level control entity and cooperation is achieved solely through communication between agents. The advantages of distributed cooperation are robustness and tractability which also includes more convenient addition of new agents without being

required to update the whole system. Whereas in many distributed systems communicative coordination links might be fixed between particular decision-making agents, in MAPF these links only make sense if there are unresolved couplings between agents which greatly varies in time. Alternatively, a new type of decision-making agent could be introduced that is fixed in space and controls the movements of all agents in its proximity. An example of such an decision-making agent would be an intersection-management agent. In [9] for example, distributed cooperation is achieved between intersection decision-making agents that control on-ramp traffic flow (the travelling agents). The traffic flow can be interpreted as agents executing their SAPF optimal solutions while consequent couplings are resolved by the intersection agents through imposing wait actions and by conventional traffic rules. Although this system achieves distributed cooperation, cooperative replanning of agents paths based to further improve the network wide performance cannot be considered.
Three main challenges can be recognised in realizing a distributed MAPF algorithm: 1) identifying (possibly suboptimal) subsets of agents and thereby establishing communicative links between agents based on decentralized information, 2) the communication protocol or information to be communicated between agents and 3) the (re)planning strategy for individual agents' paths within a subset of agents to resolve or avoid conflicts.

2-3-1 Decentralized derivations from baseline algorithms

There are a number of algorithms that adapt an existing MAPF algorithm to work in a distributed setting. For example the DMAPP [5] algorithm defines a communication protocol such that the hierarchical cooperative A* (HCA*) algorithm is carried out in a distributed manner. The extended version called DiMPP [6] enables alteration of the initial hierarchy until a feasible hierarchy is obtained making this prioritized planning approach feasible only for well-formed MAPF problems as was proven by [46]. Using local communication graphs, an algorithm called DisCoF is developed in [17] which readily applies an extension of the (complete and fast) Parallel Push-and-Swap (PPS) algorithm in a distributed setting. In follow-up work, asynchronous communication is enabled together with improved efficiency to form DisCoF+ [45]. The priority inheritance with backtracking (PIBT) [24] algorithm is based on conflict oriented windowed hierarchical cooperative A* (CO-WHCA*) and applies a one time step reservation scheme according to a random hierarchy where each agent chooses the next node as the one with the shortest distance to its goal node. Agents (temporarily) inherent priority form a higher priority agent that requests its current node and this process is continued until an agent cannot make any feasible moves. When such a deadlock occurs, the invalidity of the planned moves is backtracked along the chain of agents until one agent finds an alternative move which makes the remaining moves up until the source agent valid. The authors supply a powerful proof of completeness for any MAPF problem on an open grid map which states that there exists a time step $t \leq diam(\mathcal{G}) \cdot |\mathcal{A}|$ in which all agents are at goal. The PIBT algorithm exhibits good performance on well connected graphs such that agents can easily evade each other and the distance optimal path is approximately equal to a greedy path planning strategy. The authors observe planning ineffectiveness of the algorithm on graphs that are less connected and introduce the windowed priority inheritance with backtracking (winPIBT) [25] algorithm which extends PIBT with planning windows of more than one time step. Both the PIBT and winPIBT algorithms can be readily implemented in a distributed fashion for which the authors recognize that the sensing and communication

capabilities of agents should be two times the number of implementation steps.

2-3-2 Decentralized reinforcement learning algorithms

Reinforcement learning (RL) techniques proved to be very useful solutions in many different research areas. When applied to decentralized MAPF, the type of RL can be characterized as multi-agent reinforcement learning (MARL). In MARL, the algorithm learns an individual agent policy using learning inputs from multiple agents simultaneously. When all agents in the system execute the resulting policy, a forthcoming global objective is achieved [26]. A general overview and taxonomy of recent MARL strategies is provided in [26].
The first application of MARL to MAPF is the algorithm called pathfinding via reinforcement and imitation multi-agent learning (PRIMAL) [29]. In PRIMAL, agents learn a decentralized policy from an expert centralized (optimal) MAPF algorithm. PRIMAL combines learning optimal SAPF with imitation of an expert (centralized) MAPF planner. PRIMAL assumes that the graph is partially observable by each agent wherein the positions of other agents, goals of other agents and the agent's own goal are known. In other words, each agent has a field of view (FOV) and receives information about other agents' positions and other agents' goals that lie within its FOV. An agent furthermore maintains a vector in the direction of the agent's goal with magnitude dependent on the remaining absolute distance to this goal. During execution of the obtained decentralized policy, there is no explicit communication between agents. In the extended version called pathfinding via reinforcement and imitation multi-agent learning - lifelong (PRIMAL_2) [10], an effort is made to deal with challenging graph properties such as corridors. Additionally, the ability to obtain new goals when a goal is reached is added which is called lifelong multi-agent path finding (LMAPF). The number of input channels is considerably increased to include both optimal SAPF moves and a number of graph properties. The added channels are SAPF optimal paths for a fixed horizon of all agents in the observable part of the graph and graph specific maps that show start and end-points of corridors in the observable part of the graph. Additionally, a blocking channel shows whether there are agents inside a certain corridor. In both algorithms, the authors recognize the potential to train the policy on a MAPF problem that is tractable for an optimal centralized MAPF algorithm while the learned policy can also be applied to a multi-agent system that would no longer render tractable for a centralized optimal MAPF algorithm. Both of these algorithms implicitly assume communication between agents since an agent knows the goal locations of the other agents within its FOV, however no other explicit communication is used.
In general, centralized search-based (optimal) MAPF algorithms suggest that communication between agents enables some sort of path reservation or path announcement scheme. A coupling in an intermediate MAPF solution occurs between at least two agents and an agent is likely to be coupled with multiple agents throughout its planning horizon. It is to be expected that multi-agent transportation systems would benefit from inter agent communication. The authors in [38] conclude from ablation results that inter-agent communication is essential in their decentralized RL algorithm called scalable communication for reinforcement- and imitation-learning-based multi-agent pathfinding (SCRIMP). In order to include communication, the remaining challenges are the choices of what information to communicate and which communication links to realize. In centralized optimal search-based algorithms basically all communicative links that emerge from inter-agent couplings are evaluated, guided by

a cost tree search. The information communicated between agents is usually their paths as rendered up to that point. As decentralized algorithms specifically deal with the tractability issue and aim to improve scalability, an all to all communication scheme would inherently be undesirable. Many distributed (RL) MAPF algorithms therefore assume exclusive (spacial) neighbour communication. One of the first decentralized RL algorithms that explores reinforcement communication learning is the distributed, heuristic and communication (DHC) [22] algorithm which reduces the FOV to a 3x3 grid and executes a two-round communication between connected agents. Communication is represented by graph convolutional communication through a graph convolutional network (GCN). In the GCN, agents are represented by nodes and communicative links between agents by edges. Feature vectors of neighbouring agents are aggregated during a communication round and all of these aggregations are passed through the same neural network. Additionally, more communication rounds can be executed which will extend the communicative reach of the agents in the network. DHC demonstrates the potential of inter-agent communication by outperforming the non-explicitly communicative PRIMAL algorithm while having a smaller FOV. The prioritized communication learning (PICO) algorithm [21] first imitates the priority assignment from a centralized optimal MAPF algorithm and subsequently learns inter-agent communication based on these priorities, in turn outperforming the DHC algorithm. Alternatively, the decision causal communication (DCC) algorithm [23] only slightly reduces the FOV compared to PRIMAL and selects which specific agent to communicate with within its FOV. The resulting DCC algorithm outperforms both PRIMAL and DHC.

2-4 Meta-analysis for scalable MAPF

It can be concluded from the presented MAPF literature that different types of algorithms enjoy mutual interest from the research community. As the fundamental complexity of the optimal MAPF problem has been studied thoroughly, most recent work explores methods that can provide practical solutions. Centralized suboptimal, decentralized, and reinforcement learning approaches are jointly explored to achieve this goal, which underlines the versatility of the problem. Although the aim of these approaches might differ (section 2-2), much of these works contributes insights to the general problem as well.
This work aims to exploit the advantages of distributed control apropos of the challenges that arise in the MAPF problem. The main advantages of controlling systems in a distributed manner are scalability and robustness as a result of distributing computing and decision-making power. A system that is controlled in a distributed manner would fail to benefit from these advantages if the underlying MAPF problem would require (induced) all-to-all communication. Established MAPF literature does suggest the use of at least some sort of inter-agent communication. All recent RL MAPF algorithms attempt to learn inter-agent communication in a MARL setting. In an ideal world, MARL algorithms would indeed be able to learn the "optimum" communication protocol with the corresponding type of information that is to be shared or agreed upon between agents. However, as a well performing decentralized MARL policy is hard to obtain in general, inter-agent communication is usually learned based on manually designed feature channels to increase performance. Manually designed feature channels make the MAPF algorithmic performance likely depend heavily on the specific MAPF problem assumptions. Additionally, many of the RL MAPF algorithms seem to learn concepts that have already been developed in existing search-based MAPF algorithms

such as priority assignment and path information sharing.
A general observation among MAPF algorithms of all different approaches is the consideration of mainly local evasion manoeuvres, which might be explained by the fact that significant deviations from a SAPF solution are unlikely to yield a MAPF optimal solution (at least for SOC). Additionally, two agents likely have to negotiate shared variables just once for two reasons: 1) their general directions cross only once (two straight lines have a single intersection point) or 2) if part of their paths overlap (in this case the general directions are equal or the environment forces an overlapping path for example through a corridor), priority has to be negotiated once after which the joint solution has a leader-follower structure that doesn't have to be negotiated again. Overall, this shows a strong indication in favour of decentralized MAPF algorithms.

2-4-1 Receding horizon MAPF

Considering local evasion manoeuvres as the driving mechanism behind solving a MAPF problem implies that the solution is a sequence of local evasion manoeuvres. Computing this sequence of control inputs in a receding horizon fashion instead of over the full problem horizon, offers many analogies to receding horizon control (RHC) and model predictive control (MPC). In these type of control strategies a (optimal) control input is computed based on the current state, deviation from the terminal state and acting (or predicted) disturbances. To the authors best knowledge, these strategies have not been applied directly to conceptual MAPF problems. For example the MAPF algorithms in [4] and [32] do implement windowed strategies by computing a number of control inputs over a certain window for the global MAPF problem and repeat this process once the planning window has elapsed. These two algorithms have better success rates compared to the baseline HCA* algorithm as the windowed planning allows for some more flexibility, however both methods are still not complete. Additionally, both methods will suffer from horizon effects by not being able to take the system state after the planning window into account. The PIBT algorithm [24] essentially makes the windowed hierarchical cooperative A* (WHCA*) algorithm with a planning window of one step complete by letting agents inherent priority if an intermediate move would be infeasible. As the PIBT algorithm renders inefficient solutions because of the short planning window, this is improved by the winPIBT algorithm [25] that applies the same strategy with a longer planning window.

2-5 Literature gap

Considering the presented research on MAPF algorithms, it can be concluded that there is a literature gap regarding decentralized MAPF algorithms that apply windowed planning. Whereas many decentralized algorithms aim to realize an established centralized MAPF algorithm in a decentralized setting, other work demonstrates the potential of developing an algorithm from a fundamentally decentralized perspective. The proposed RL MAPF algorithms essentially try to learn optimal local evasion manoeuvres that are also exhibited by optimal (centralized) MAPF algorithms. The drawback of this black-box approach is that the fundamental complexity of learning a decentralized policy forces the designer to make problem specific assumptions resulting in loss of generality.
Decentralized search-based algorithms on the other hand mostly use random prioritization

to realize fast algorithms and considering dynamic priorities can considerably increase completeness of the algorithm. The drawback of a decentralized random prioritization scheme is the inevitability of planning ineffectiveness as the algorithms cannot accommodate specific situations. For example, inefficient solutions will be obtained by these algorithms in any case where a higher priority agent has to incur delay in favour of a lower priority agent in order to achieve a combined delay that would much more lower in comparison to the contrary.
In this work an algorithm will be developed that addresses the challenges of decentralized MAPF control as raised in section 2-3. A novel receding horizon framework will be developed from a fundamentally decentralized perspective that exploits insights from MAPF research as well as distributed control theory.

Chapter 3

Distributed control framework

The previous chapter introduced the notion of distributed control and motivated the exploration of its application to the multi-agent path finding (MAPF) problem. This chapter will elaborate on the design of the distributed control algorithm and discuss its numerous degrees of design freedom. The design starting point of the algorithm follows from two assumptions that are inherent to eventual practical application of the algorithm. Firstly, any entity travelling through any network will require the capability to compute and actuate its physical movements. It is therefore necessary that the travelling entity has some computational capability in order to compute inputs for its actuators to follow a desired path. Secondly, it is assumed that entities travelling through the network can communicate based on their proximity. The extent to which these two requirements are met in practice will depend on the application.

Decision-making power is distributed in the following manner: each entity travelling through the network plans its own path and controls its actuators in order follow this desired path, these type of decision-making agents are referred to as "travelling agents". Alternatively, paths could be assigned simultaneously to a subset consisting of travelling agents. However, selecting these subsets in a decentralized manner would require either a large overhead of communication or require a centralized communication entity. Additionally, the very process of selecting subsets of travelling agents can pose problematic as described in subsection 2-3-2. Thus, as there will be no other types of decision-making agents considered in this work, the travelling agents are simply called "agents". The only type of controller that is considered is the path planning controller, it is therefore assumed that agents implement exactly those paths as computed by the path planning controller.

Since the (global) optimal MAPF problem is of NP complexity, the application of a distributed optimization technique to solve the global MAPF problem (e.g. [37]) will render intractable as the computational capability scales only linearly (with the number of agents). The algorithm that will be described in this chapter therefore follows a decentralized local optimization approach that is inspired by the continuous time receding horizon control algorithm described in [16]. In this work agents optimize their own path over certain time window while considering (predicted) costs of neighbouring agents. This approach is in line with the meta-analysis of MAPF algorithms described in section 2-4, which concludes that effective MAPF algorithms

compute mostly local evasion manoeuvres. Since the number of agents that is considered (to be) in the proximity of one agent is finite (and can be fixed), the optimal solution to the local MAPF problem is guaranteed to be found within a certain time limit. On the other hand, by considering a planning horizon that is shorter than the minimum (optimal) time required for all agents to reach their goals, it is no longer guaranteed that the optimal solution or even an existing feasible solution can be attained. The framework described in [16] considers continuous time linear time-invariant (LTI) dynamical systems and as the MAPF problem does not comply with these system characteristics, only the concept of windowed local optimization will be considered. Also, in order to turn the described decentralized control approach into a distributed control approach, the agents will consequently have to communicate to reach consensus over the locally optimized paths.
The goal of this research is to design a highly scalable distributed MAPF algorithm in which agents optimize their paths in parallel while considering a common objective of an agent's own goal together with the goals of a tractable number of other agents. After local optimization, agents have to reach consensus over the shared variables which are the nodes and edges of the network over the planning horizon. A supplementary element of this research is to investigate the type of information that is shared among agents and how information about path alternatives can be used for better parallel decision making.

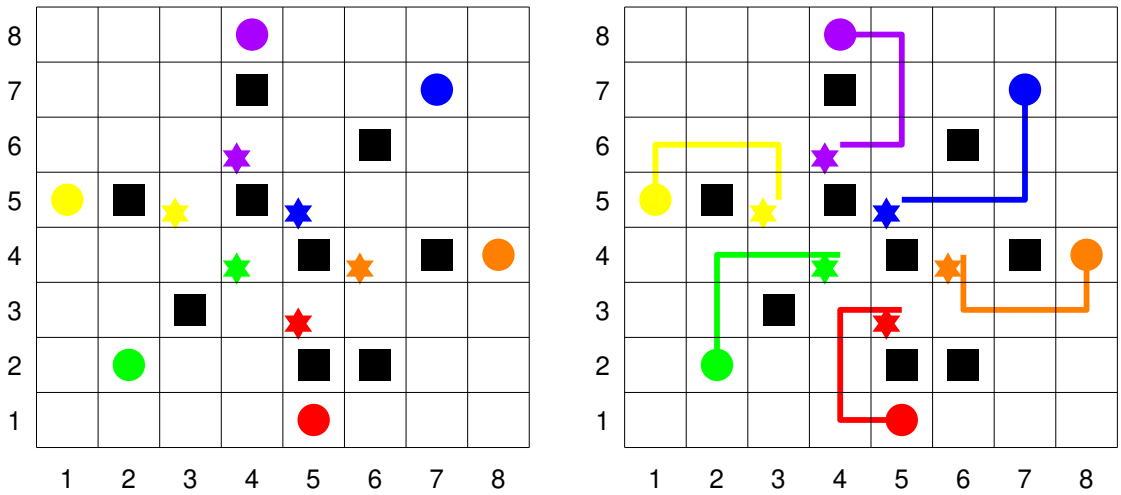
3-1 Problem setup

The working principle of the distributed control algorithm is shown in Algorithm 1. The agents participating in the MAPF problem by executing Algorithm 1 are indistinguishable, that is: they execute the same algorithm in parallel. A distinction between agents does have to be made when considering Algorithm 1 from the perspective of one particular agent. If one particular agent is considered, this agent will be referred to as the "searching agent" (denoted $a_i \subseteq \mathcal{A}$). The searching agent can communicate with other agents within its proximity, these agents are called (the time-varying set of) "connected agents" (denoted $\Gamma_i(n) \subseteq \mathcal{A}$). The searching agent considers the alternatives and goals of a set of "selected agents" (denoted $\mathcal{R}_i(n) \subseteq \Gamma_i(n)$) in its local path optimization in line 5 of Algorithm 1. Agents are added to the set of connected agents based on their proximity to the searching agent in line 3 of Algorithm 1. A planning horizon of w steps is considered and each planning iteration a number of $m \leq w$ steps is implemented which yields a local MAPF problem that is smaller than or equal to the global MAPF problem.

```
1 Each agent in parallel, do
2 while current position ≠ goal do
3 |   infer the set of connected agents based on proximity
4 |   exchange information regarding path alternatives with connected agents
5 |   optimize desired path over window w while considering selected agents
6 |   negotiate shared variables with connected agents
7 |   implement m ≤ w steps
8 end
```

Algorithm 1: Basic path negotiation algorithm with planning horizon w and implementation window m

The type of MAPF problem that will be considered in this research is the widely used abstracted grid form in which start coordinates, end coordinates, and blocked coordinates are assigned randomly. Each agent is assumed to have full static map knowledge, while information about other agents has to be obtained through communication. An example of a MAPF problem is shown in Figure 3-1 with the unsolved initial situation in Figure 3-1a and with a feasible solution in Figure 3-1b. Blocked coordinates, or obstacles, are represented by black squares. Start coordinates are indicated by circles and goal coordinates by hexagrams. Furthermore, paths are represented by coloured lines where each agents has a dedicated colour. A map such as shown in Figure 3-1 has an "invisible" border of blocked coordinates such that agents cannot "leave" the map.



(a) Initial problem **(b)** Solved problem

Figure 3-1: Example of a problem setup

Agents can execute one of the following five actions in each time step: move up, down, left or right or wait at current location. Apart from not using blocked coordinates, the following inter-agent conflicts that were described in section 2-1 will not be tolerated: node conflicts and edge conflicts.

An important distinction in MAPF algorithms is the assumed consequence of agents reaching their goal. In this research, agents are assumed to stay at their goal node if they have reached this node and any other agent has not yet reached its goal node. The MAPF problem is considered as solved once all agents have reached their goal node, in literature this is referred to as "one-shot" MAPF. Although the assumption of "one-shot" MAPF might seem of less practical value, this assumption best enables the analysis of the performance of the MAPF algorithm itself. The alternative assumption of letting agents disappear upon arrival at their goal node certainly has even less practical value. The most realistic alternative assumption of agents having an infinite number of goal nodes over a finite time horizon would require additional assumptions on the priority of specific task completions and would alter the purely MAPF problem to a multi-agent package and delivery (MAPD) or multi-robot task allocation (MRTA) problem, depending on the particular application.

Communication links are established between agents based on proximity. In practice, the most appropriate approach in establishing communication links between agents will depend on the

type of communication technology that is utilized. In theoretical algorithms for abstracted MAPF problems a fixed observation range within a Euclidean norm is usually assumed. The value of this Euclidean norm will be denoted by v.

3-1-1 Summary of assumptions

The collection of assumptions that were posed can be summarized as follows:

- **actuator dynamics** the effect of actuator dynamics is neglected and agents are therefore assumed to perfectly follow the result of the path planning algorithm
- **initial locations** the initial locations for start, goal, and obstacle coordinates do not overlap and the initial problem is therefore conflict free
- **map knowledge** agents posses full map knowledge: agents are informed with the location of all blocked coordinates
- **feasible actions** each time step, agents can perform one of the following five actions: up, down, right, left or wait
- **inter-agent conflicts** the notions of node conflict and edge conflict as defined in section 2-1 are not allowed to be included in the solution for a MAPF problem instance.
- **inter-agent communication** communication links between agents are established based on Euclidean norm proximity
- **one-shot MAPF** an agent stays at its goal location. The instance of a MAPF problem is considered solved as soon as all participating agents have reached their goal. An agent can move "out of the way" from its goal for another agent to pass, the time step in which consequently the goal is reached for the last time counts as the task completion time for this agent

3-2 Shared information

A concept that was introduced (and consequently also widely accepted as a result of proven effectiveness) in the research area of decentralized MAPF algorithms, is multiple step inter-agent communication. In the case of decentralized multi-agent reinforcement learning (MARL) algorithms (e.g. [38],[22],[21]), it is left to a reinforcement learning mechanism to infer what information is passed between agents in either a single or multiple round communication scheme. To the authors best knowledge, there is no work on decentralized parallel decision-making MAPF that investigates the type of information that is to be shared among agents.

3-2-1 Multi-value Decision Diagrams

First off, in order to be able to consider a set of selected agents during local optimization, the searching agent requires information about the goals of other agents. Rather than simply sharing goal locations with connected agents, the agents will share the complete set of alternative paths with their respective connected agents. This type of information is called a multi-value decision diagram (MDD) which was introduced in [31]. The concept of an MDD will be explained before motivating the choice for the specific choice of MDDs as the shared type of information. An MDD represents the set of alternative paths that an agent can traverse in order to reach a goal node or nodes such that each possible alternative path is of equal cost. The layers of an MDD diagram, such as shown in Figure 3-2a, correspond to time steps where the top layer is the current time step. Each layers consists of a number of nodes that are connected with arcs to other nodes on both the previous layer and the successive layer. The arcs represent actions that an agent has to perform to move from a certain node in a particular time step to another node in the next time step, hence different actions result in different paths.

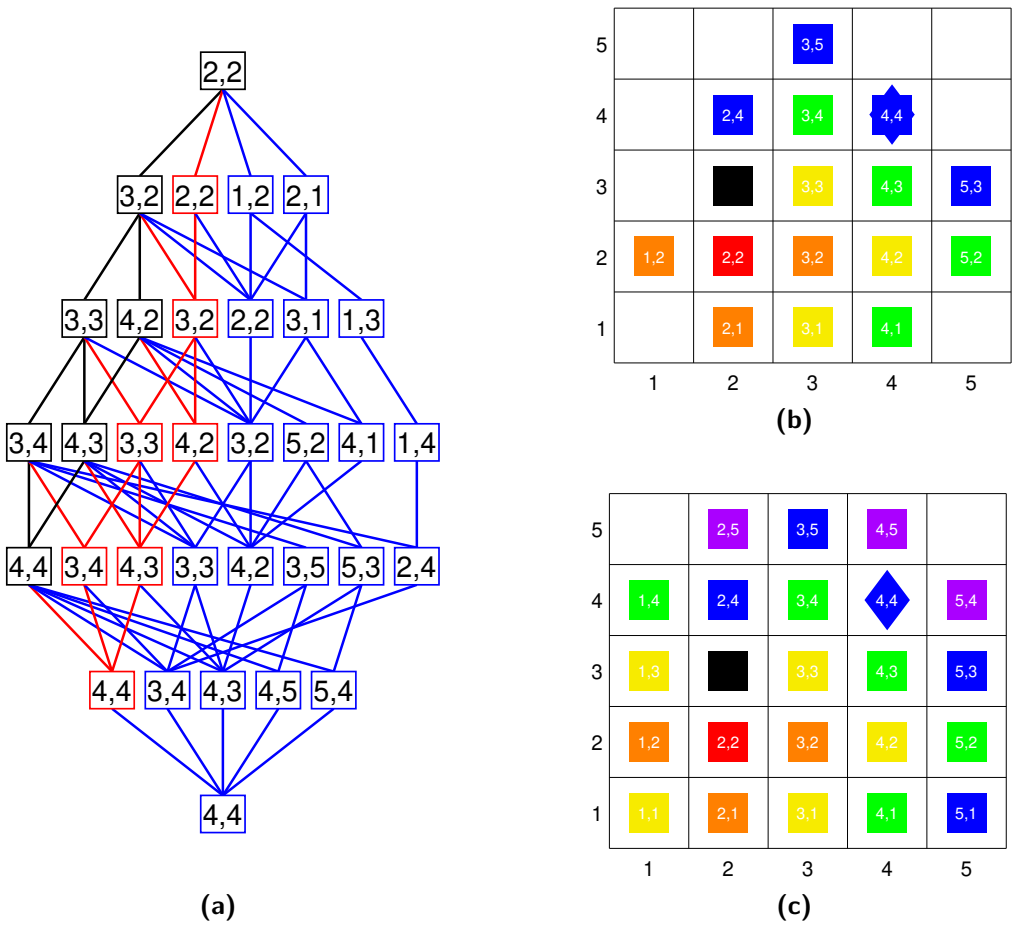


Figure 3-2: Construction of a two step delay MDD (a) from expanding the A* graph expansion base solution (b) with two extra steps (c)

The construction of a MDD harmonizes the result of a graph search single-agent path find-

ing (SAPF) solution. Figure 3-2b shows the result of an A* graph search with Euclidean distance as the heuristic cost. The different colors represent the amount of steps that were taken from the start location, note that the goal location is reached in 4 time steps (blue), this number of steps is called the minimum depth. If a delay of d time steps is considered, the graph search is continued by expanding nodes that have a heuristic cost lower than $minimum\ depth + delay$. An extended search of delay $d = 2$ is shown in Figure 3-2c. To construct a MDD from Figure 3-2c, one fills the layers of the MDD with expanded nodes. The depth t of a layer determines which expanded nodes are included in this layer: $MDD_{layer}(t) = nodes|_{(t-d)\leq depth(nodes)\leq t}$. Consequently, arcs are established between MDD layers if a particular node in one layer can reach a particular node in the next layer by applying one of the five feasible actions. Finally, nodes that are not connected to both a node in the next as well as the previous layer are filtered from the MDD. The different colours of the nodes and arcs of the MDD in Figure 3-2a represent the incurred amount of delay as a result of traversing a particular arc where black is zero steps delay, red is one step and blue is two steps. Note that the included MDD with one step delay (red) is just a shifted version of the MDD with zero delay as it takes at least two steps to make a detour in a grid based graph. Also note that a delay is always incurred irreversibly, which makes it relatively easy to extend an existing MDD with an extra step of delay by expanding the corresponding SAPF solution and appending the new nodes to the layers of the corresponding MDD.
From now on, the following notation will be used to indicate the amount of incurred delay that is included in a MDD for a particular agent: $\text{MDD}^{\text{up to } \delta \text{ steps delay}}_{\text{agent number:} a_i \in \mathcal{A}} \rightarrow \text{MDD}^{\delta}_{i}$.

3-2-2 Additional information in a MDD

The MDD concept offers several advantages with regard to decentralized communication. Firstly, the number of different path alternatives in the diagram grows exponentially while the MDD itself grows linearly in size [1]. Secondly, multiple MDDs can be compared in polynomial time as only the nodes of each corresponding layer have to be compared. If a particular node occurs in two MDDs in the same layer, these two agents are dependent on one another for not both choosing this particular node in this specific time step and the notion of a "node dependency" can be appended to the corresponding MDDs of both agents. Similarly, an "edge dependency" can be appended to particular edges in two MDDs if correspondingly these two agents will participate in an edge conflict by simultaneously traversing the respective edge. In general, a dependency represents a coupling between two agents where either one or none of the agents can opt to occupy a particular node or traverse a particular edge in a specific time step.
Each path selection round, an agent will construct its MDD and share its MDD up to layer $w+1$ with connected agents as described in line 4 of Algorithm 1. Sharing a windowed part of the MDD offers a number of benefits over sharing a goal location. Firstly, if a searching agent would receive goal locations of all of its connected agents, the searching agent would have search path alternatives for all of these agents, thus creating a lot of overhead computation. Secondly, privacy concerns might be raised in future practical implementation and by sharing path alternatives exclusively within the planning horizon, the goal location of other agents can only be inferred if it falls well within the planning horizon.
An exemplary MAPF problem was designed to exhibit the dependency scheme applied to a decentralized setting which is shown in Figure 3-3. This decentralized problem is demon-

strated from the perspective of agent a_2(orange) and the assumed Euclidean norm for the communication range is $v = 6$. The planning horizon is equal to the minimum depth of all agents: $w = 4$. Corresponding to line 3 of Algorithm 1, communication links are established to form Γ_2 containing agents a_1(red), a_3(blue), and a_4(purple). The MAPF problem map in Figure 3-3a only shows the alternatives of the set of connected agents, with delay $d = 0$. After having shared their MDD to connected agents, all agents agent can infer their respective set of dependencies and append this information to their MDD. The result of appending MDDs is shown in Figure 3-3b to 3-3e. The dashed boxes illustrate nodes with (node) dependencies and the colour corresponds to the respective agent with which the dependency is shared. Note that a dependency over the same resource could occur between more than two agents but that this is not the case in the example. Also note that edge dependencies do not occur in this example.

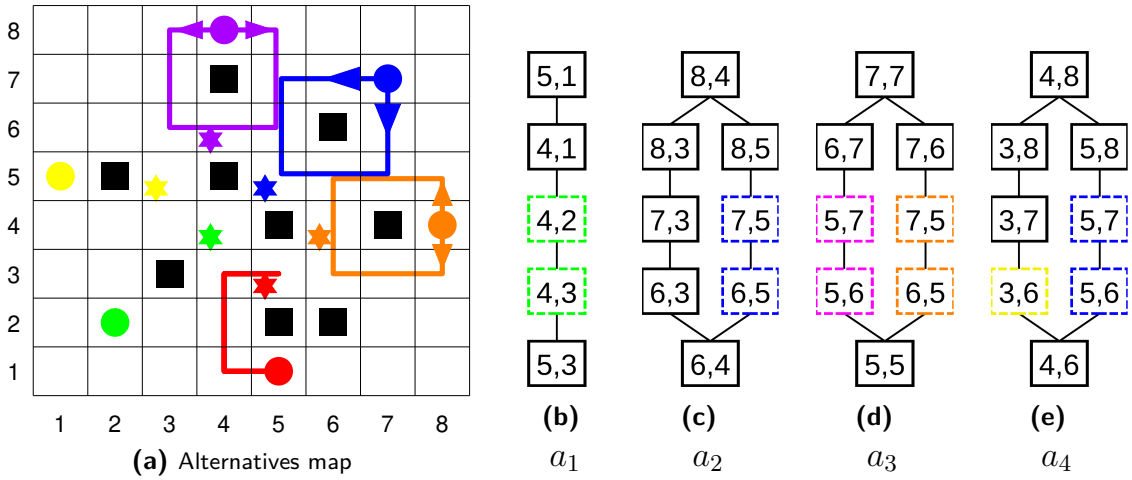


Figure 3-3: MAPF problem example to illustrate dependencies in a decentralized setting

In an important next step, all agents resend their appended MDDs to their respective set of connected agents. This last step enables the searching agent to obtain information about agents that are outside of its own communicative reach and concludes the actions described in line 4 of Algorithm 1. To illustrate the result of these two steps of sharing information, a MDD product between all connected agents of agent a_2 is shown in Figure 3-4. A joint node in the MDD product (black boxes) contains a node for each agent participating in the MDD product while an edge comprises an action for each participating agent. The layers of a MDD product are constructed by taking all possible combinations of nodes of participating MDDs in the corresponding layer. Note that layers 2-4 of the MDD product actually have $1 \cdot 2 \cdot 2 \cdot 2 \cdot = 16$ joint nodes, however most of these nodes have been filtered by considering only feasible joint nodes. The filtered result of the MDD product shown in Figure 3-4 denotes joint nodes by black boxes while nodes of individual agents are colour coded by the agent's respective colour. The dependencies obtained in Figure 3-3 are also included in the MDD product but now only the type of dependency is displayed through the border type. The general notion of a dependency can no be split into three: 1) an external dependency with an agent that does not belong to the set of connected agents ("ext"), 2) an internal dependency between agents that are included in the set of connected agents ("int"), 3) an internal depen-

dency that includes the searching agent ("sa").

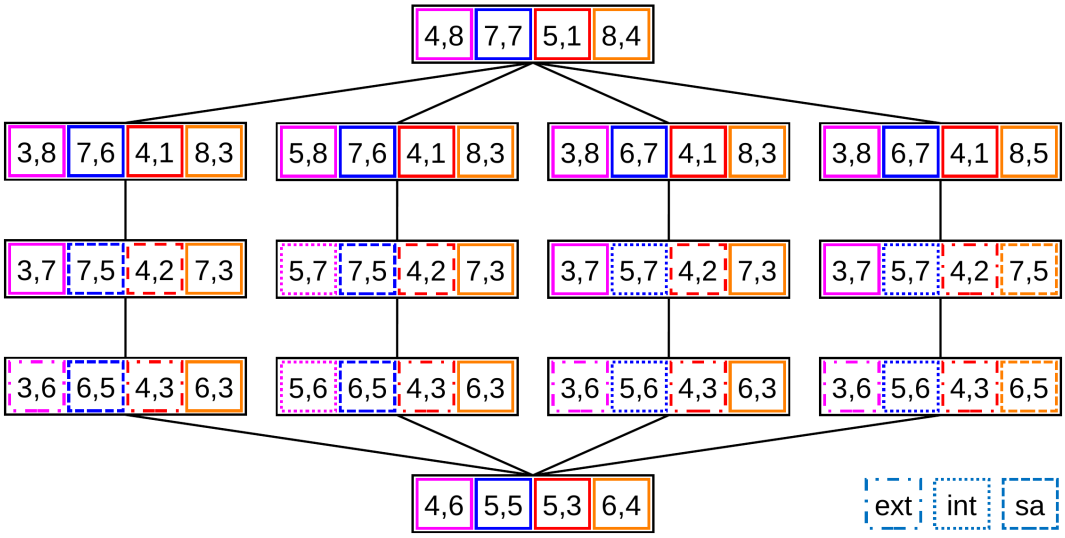


Figure 3-4: From the perspective of agent a_2, the filtered MDD product between the MDDs of the set Γ_2 where dependencies corresponding to the nodes of individual agents are included as "ext" (external), "int" (internal), and "sa" (internal with searching agent)

Ultimately, the appended information about individual agents dependencies could help distinguish between different joint solutions. Although the example of Figure 3-4 shows four feasible solutions for the local MAPF problem, the only local solution that would yield a global feasible solution with zero delay for all agents is the second column. Of the four feasible solutions, this solution can be differentiated from the other three solutions by the fact that is has a total of two external dependencies over all layers while the other solutions all have a total of three external dependencies over all layers.

3-3 Local MAPF optimization

The input of the local MAPF optimization problem is established to be the joint set of alternatives with additional node and edge properties as shown in Figure 3-4. The next step in executing the distributed control algorithm corresponds to line 5 of Algorithm 1, which executes the local windowed MAPF optimization. The local optimization problem will consider a tractable amount of agents to ensure that a local solution is found within a feasible amount of time. The set of agents that a searching agent will consider during optimization was defined as the set of selected agents ($\mathcal{R}_i \subseteq \Gamma_i$) in section 3-1. The reason for defining two different sets ($\mathcal{R}_i$ and Γ_i) has to do with the computational tractability relative to the communicative reach. For example, if every node in a grid MAPF problem instance is occupied by a different agent and the Euclidean norm for setting up communication links is 6, there are 113 agents in the set of connected agents of a searching agent. The amount of agents that will be considered tractable for solving an optimal MAPF problem in reasonable time will be called r. This number r is commonly considered to be at least less than 100 (strongly dependend on problem size and type), while for coupled MAPF solving approaches

this amount is significantly lower. It is for this reason that an additional step was added before local MAPF optimization which is the selection of a set of agents to consider during optimization. The selection of agents from Γ_i follows two basic steps:

1. at least every agent that can be reached within the implementation steps is added to $\mathcal{R}_i(n)$ such that $\mathcal{R}_i(n) = \Gamma_i(n)|\{\forall a_j \in \Gamma_i(n),\ \text{Manhtd}(\pi_j(n),\ \pi_i(n)) \ \leq \ 2 \cdot m\}$, consequently the number of implementation steps m is bounded by the considered tractable number of r selected agents.

2. if $|\mathcal{R}_i(n)| < r$ as a result of the preceding step, additional agents $a_j \in \Gamma_i(n)$ are added based on their Manhattan distance ($\text{Manhtd}(\pi_j(n),\ \pi_i(n))$) to the searching agent until $|\mathcal{R}_i(n)| = r$.

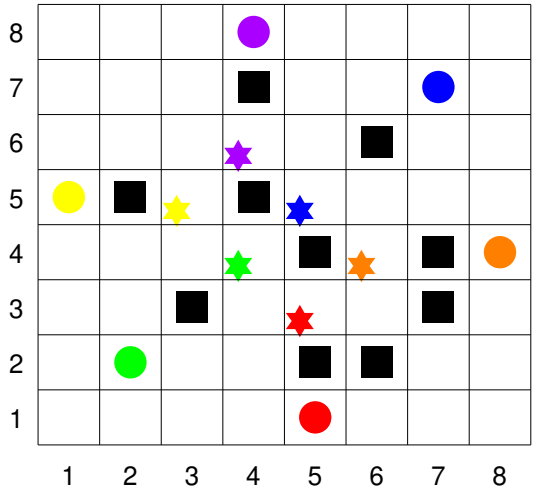
3-3-1 Cost function

Two types of objectives were introduced in MAPF literature as described in subsection 2-1-1, namely the sum of costs (SOC) objective and the makespan objective. These objectives refer to the global state of the system and would require solving the global MAPF problem in a distributed manner which was advocated to be undesirable as it would still yield an intractable problem. Moreover, due to the nature of a receding horizon controller that optimizes a windowed MAPF problem, agents are likely not to reach their goal within the planning horizon. Intuitively, intermediate locations could be considered that are classified with respect to their conformity with the final goal of an agent. To sketch an example of this approach one can consider layer four (from the top) of the MDD shown in Figure 3-2a. The colour of the boxes characterizes the conformity of a particular node with the final goal of the agent in terms of incurred delay. Node (4,3) is more favourable to choose in layer four compared to node (5,2) for example, as node (4,3) allows the agent to reach its goal node (4,4) in the next time step while node (5,2) would require two more time steps (hence two steps delay). The objective to consider in a windowed MAPF problem therefore becomes the combined intermediate delay which is basically a confined interpretation of the introduced global objectives. Analogously to the two mentioned global objectives, one can consider the confined interpretation of the makespan objective as the minimal maximum intermediate delay and the confined interpretation of the SOC objective as the total intermediate delay. The notion of "intermediate" delay refers to the fact that a MDD will be constructed in each path planning round, starting from the node that the agent currently occupies. If for example the agent in Figure 3-2a opts to implement two steps [(1,2),(1,3)], the path planning is re-initiated from the current node (1,3) and the zero-delay part of the new MDD will be [(1,4),(2,4),(3,4),(4,4)] as this would be the only option to get to the goal node without incurring any new intermediate delay. As a result, the final delay of an agent for a particular MAPF problem is equal to the sum of intermediate delays that this agent has incurred during the combination of all planning rounds.

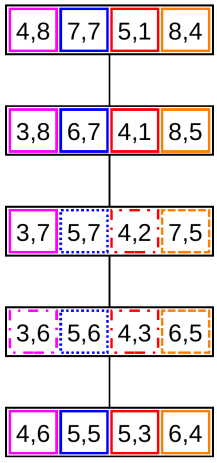
3-3-2 Baseline feasible solution

Agents participating in a MAPF problem controlled in a distributed manner inherently have access to a limited amount of information. As a consequence, a situation might occur where

a solution exists from a decentralized perspective but not on from the global perspective. An example of such a situation is shown in Figure 3-5 which is identical to the problem shown in Figure 3-3 except for the additional blocked coordinate (7,3). All agents in this problem will find a feasible local solution for **zero delay** that is somewhat similar to that of agent a_2 in Figure 3-5b and therefore all agents will select a path with zero delay. Using figure Figure 3-5a, one can easily infer that there doesn't exists a globally feasible solution where all agents have zero delay. Increasing the communicative range such that all agents are connected, and provided that all agents can decide in parallel on one particular agent that should incur delay (for example the most left agent in the chain), would allow the global problem to be solved. All agents now essentially solve the centralized problem, which undermines the decentralized approach and is certainly not realisable for a larger problem where this "dependency chain" is longer.



(a) problem setup



(b) Corresponding solution set for zero delay from the perspective of agent a_2

Figure 3-5: Exemplary MAPF problem where not all agents globally can reach their goal with zero delay but cannot infer this from a decentralized perspective

To resolve the described issue, a conflict resolution strategy of random priorities will be implemented. Random priorities are the driving mechanism behind most fast algorithms and basically assign a shared variable to the agent with the highest priority whenever a conflicts over shared variables occur. The main problem concerning algorithms that utilize random priorities as a conflict resolution method is that they are not complete if the priority order simply cannot accommodate the set of feasible solutions (a simple example is shown in Figure 2 of [33]). Dynamic priorities are required in such special cases, for example through backtracking as introduced in [24]. Priority backtracking is however likely to render inefficient solutions and requires relatively more communicative overhead and, especially with longer planning horizons. In order to reduce the communicative overhead and increase solution efficiency, this work will apply the following approach: agents optimizing locally while considering the local objectives of other agents in their proximity will induce a mechanism similar to dynamic priorities. That is, an agent will incur delay if doing so follows from the local optimization problem. Since the agents considered in a local optimization problem solve a very similar local optimization problem themselves, all agents will come to complying solutions. The hypothesis

is that the decentralized mechanism of selecting the optimum joint local solution based on the supplied information (appended MDDs) will enable agents to dynamically adjust their implicit priorities (can be interpreted as relatively incurred delay) based on their surroundings. As all agents will select their local optimum solution according to the same decentralized policy, it is expected that they will conclude on the same paths for each other. Furthermore, solving an optimization problem allows for more flexibility compared to (dynamically) prioritizing agents. Adding more information to the local MAPF problem by appending MDDs is expected to enable agents to differentiate between different local solutions with equal cost to ameliorate the parallel decision-making by coming to the same joint solutions without the need for additional communication. The "appended" properties embedded in the solutions of a local MAPF problem could also be used in a heuristic manner by letting agents actively choose paths that are less dependent on other agents to potentially favour the effectiveness of decentralized decision-making.
The effectiveness of parallel decentralized decision-making is still to be investigated and it is therefore uncertain if this decentralized control approach will indeed yield local MAPF solutions that have converged to induced consensus between agents, that is: in which agents have selected the same paths for each other. A restrictive conflict resolution strategy has been designed to ensure that the implemented steps m of all agents are guaranteed to be conflict free. This enables studying the decentralized optimization approach and also allows for comparison to other decentralized algorithms. The strategy entails the following; an agent will be allowed to refer to a baseline solution which is waiting at its current location for the duration of m implementation steps. Consequently, follower conflicts as described in section 2-1 are not allowed in the first m time steps. Although the imposition of prohibiting follower conflicts in the first m time steps is restrictive, this approach requires much less communication compared to the priority backtracking approach. Agents simply compare their local solutions and can revert to the baseline solution at any point during this negotiation. Additionally, if the parallel decentralized decision-making approach always yields local solutions that have converged, the baseline solution never has to be activated.

3-3-3 MDD product path selection strategy (local optimization)

To find a local optimal MAPF solution, graph expansion of the MDD product (MDD product) is applied which corresponds to the coupled centralized increasing cost tree search (ICTS) algorithm [31]. Note that "graph" in this context refers to the combined set of considered MDDs as shown in Figure 3-4 and not to the spacial graph that describes the MAPF problem. The graph that describes a MDD product will from now on be referred to as the MDD product with nodes describing a location for each of the participating agents and edges describing an action for each of the participating agents.
Constructing the MDD product quickly becomes intractable as the number of nodes in each layer of the MDD product is equal to the product of the number of nodes on the respective layers of the MDDs of participating agents. The number of nodes in the MDD product therefore scales exponentially with a linearly growing number of agents. The MDD product is therefore expanded like a graph search in order to construct only the part of the MDD product that is required to find the optimal solution. In each iteration, a node of the MDD product is expanded by adding all of the connected nodes (on the next layer) to the open list of nodes. Consequently, each newly added node will be associated with a depth and a cost as a result

of the defined cost function. The next iteration expands the node with the lowest cost of the list of open nodes. The node cost is a function of the cost of the originating node, delay, internal dependencies and external dependencies:

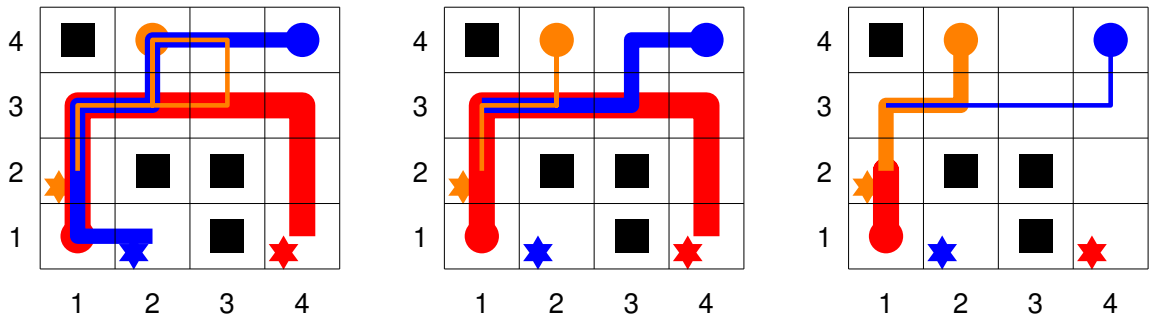
$$c(node_{new}) = c(node_{old}) + w_1 \sum_{n=1}^{|\mathcal{R}_i|} \Delta\delta_n + w_2 \sum_{n=1}^{|\mathcal{R}_i|} int.dep.(a_n) + w_3 \sum_{n=1}^{|\mathcal{R}_i|-1} ext.dep.(a_n) \quad (3\text{-}1)$$

Where $\Delta\delta_n$ is the increase of incurred delay from the previous layer to the current layer of agent n, $int.dep.(\cdot)$ denotes the number of internal dependencies that an agent has as a result of its individual node and similarly $ext.dep.(\cdot)$ denotes the number of external dependencies. Note that this cost function doesn't distinguish between internal dependencies within the selected set of selected agents and internal dependencies specifically with the searching agent. Also note that the sum of delay increases together with the cost of the previous node amounts to the confined interpretation of the SOC objective as the minimum total intermediate delay.

3-3-4 Mitigation of horizon effects

A consequence of considering a planning horizon that is shorter than the problem horizon in receding horizon control are implications as a result of the "horizon effect". The horizon effect concerns information about changes in system state or acting disturbance(s) that aren't considered in planning as they lie outside the planning horizon. To capture the (predicted) effect of states and disturbances on the trajectory cost that will occur after the planning horizon, receding horizon control (RHC) and model predictive control (MPC) add a terminal cost that is only depended on the final state.
For windowed MAPF, two different implications of the horizon effect have to be considered. Firstly, agents that will interfere with a searching agent but are outside of communication range in a certain time step can be interpreted as disturbances that will act outside of the planning horizon. Agents that were within communication range but didn't belong to the set of selected agents for local optimization can be considered as an equivalent disturbance. The implications of this type of disturbance might be alleviated through some form of extended communication, for example induced through connected agents. It is however assumed that setting the communication range and planning horizon sufficiently long will make these effects negligible as the fundamental basis of the algorithm builds upon local evasion manoeuvres (section 2-4). Secondly, the windowed planning mechanism can result in deadlocks. Planning over the full problem horizon can never result in a deadlock as the corresponding cost would be infinite (agents will not reach their goal). In windowed planning the cost corresponding to a deadlock can be finite as the cost evaluated is the incurred delay with respect to the zero delay path. A deadlock will occur in windowed planning if the joint delay cost of avoiding a deadlock is higher compared to the cost of being stuck (equally) in the deadlock.

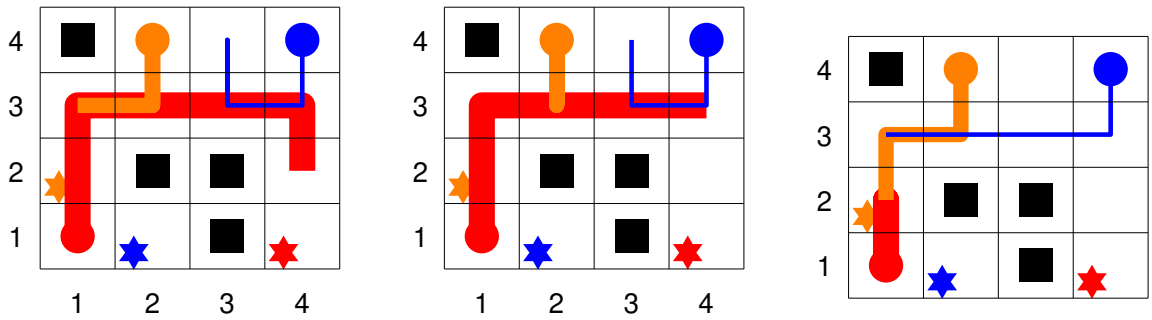
The illustrative example shown in Figure 3-6 will be used to explain the implications of applying the confined cost functions described in subsection 3-3-1 in windowed optimization for a challenging MAPF problem. Figures 3-6(a)-(c) show the results of the first step of windowed planning corresponding to different planning horizon lengths w. The three agents a_1 (red), a_2 (orange), and a_3 (blue) each have a planned number of w actions represented by lines with their respective colours. Note that the planned paths have different lengths because



(a) $c_0,\ w = 9$ **(b)** $c_0,\ w = 8$ **(c)** $c_0,\ w = 6$

Figure 3-6: MAPF problem that will result in a deadlock if the planning horizon w is insufficiently long for the baseline of the cost function of Equation 3-1: $c_0(w_1, w_2, w_3) = c_0(1, 0, 0)$

wait actions cannot be displayed explicitly in this setting, though all agents do execute the same number of w actions. The cost function c_0 implements the node cost function described in Equation 3-1 as $c_0(w_1, w_2, w_3) = c_0(1, 0, 0)$ which can be interpreted as the baseline cost function that only considers the sum of (intermediate) delays. The scenario displayed in Figure 3-6 is challenging for a windowed MAPF optimization algorithm because multiple agents (a_2 and a_3) have to incur delay for agent a_1 to pass. For a relatively shorter planning horizon, the actions of agents a_2 and a_3 would have to be exclusively wait actions, hereby incurring maximum delay. As in this case two agents have to incur this maximum delay versus one agent incurring no delay, a SOC intermediate delay path optimization will revert to the solution shown in Figure 3-6c where only one agent (a_1) incurs maximum delay. The scenario of Figure 3-6c will result in a deadlock in which only agent (a_2) will reach its goal. If the window length w is equal to the minimum problem depth, the agents are able to find the optimum solution as shown in Figure 3-6a. If the planning horizon is shorter, the agents are not able to see the optimum solution as shown in Figure 3-6b. However, as the path allocation process will be carried out in a receding horizon fashion, the scenario in Figure 3-6b will still be able to let all agents reach their goal in the minimum amount of steps for an implementation length of $m = 1$. For $w = 7$ (no figure) only agents a_1 and a_2 will reach their goal and for $w = 6$ (Figure 3-6c) only agent a_2 will reach its goal. These last two scenarios result in a deadlock where consequently the algorithm will not yield a feasible global solution.
One can infer from the scenarios displayed in Figure 3-6 that it becomes challenging for a windowed MAPF optimization approach to obtain intermediate solutions that will yield a feasible solution for the global MAPF problem if the majority of agents has to perform wait actions for most of the planning horizon. This is also the reason that the confined interpretation of the makespan objective (subsection 2-1-1) is less suitable for windowed MAPF planning since a scenario where at least one agent has a delay equal to the full planning horizon, different solutions with respect to other agents cannot be distinguished using this function. From a global perspective, a deadlock is easily recognized if in any time step there is **not a single agent** that decreases its Manhattan distance to its goal location. In a decentralized setting it is challenging to distinguish if an agent is either stuck in a deadlock or has to incur delay to let another agent past. Consequently, it is also challenging to infer which agents participate in the same deadlock and if this is recognized, to come up with a decentralized policy that can resolve the deadlock situation.



(a) $c_0 + c_T,\ w = 6,\ T = 3$ **(b)** $c_0 + c_T,\ w = 5,\ T = 3$ **(c)** $c_0,\ w = 5$

Figure 3-7: MAPF problem instance first step planning results for cost function c_0 with supplemented terminal cost penalty for $w = 6$ **(a)** and $w = 5$ **(b)** and comparative result without terminal cost penalty for $w = 5$ **(c)**

To alleviate the described implications of the horizon effect that can be interpreted as finite cost deadlock scenarios, a terminal cost function was designed analogously to RHC and MPC. The terminal cost function is implemented as a penalty function which penalizes local MAPF solutions that do not have at least one agent (that is not at goal) decreasing its distance to goal in the final time step of the planning horizon. In order to avoid scenarios where one agent in a deadlock takes a step away from goal in time step $w - 1$ to consequently takes a step towards goal in time step w (which would obviously not resolve the deadlock), a "decreasing threshold" (T) is introduced. This threshold demands an agent to execute a number of T consecutive actions (including the final action in time step w) that all result in a decrease of its distance to goal to be considered a "decreasing agent". Introducing the threshold T avoids agents taking one step back and decrease their distance to goal only in the final time step to avoid the deadlock penalty. Inevitably, in some cases a "truly" decreasing agent might have less consecutive decreasing steps than the threshold, T is therefore left as a hyperparameter to be obtained through tuning. Additionally, cases in which agents deliberately have to take detours will also be wrongly penalized by the penalty function c_T which underlines the challenge of inferring a deadlock scenario in a decentralized manner. The terminal cost penalty function is given in Equation 3-2 and is dependent on both the set of agents that are not at goal in the final time step (η) and the set of decreasing agents (θ) in the final time step. A penalty of large number value F is given whenever the set $\eta \subseteq \mathcal{R}_i$ is not empty while the set $\theta \subseteq \eta$ is empty, which is the case when there is no decreasing agent in the final time step w among the agents that are not yet at goal.

$$\begin{aligned} \eta(node) &= \forall a_j \in \mathcal{R}_i \mid \pi_j(w) \neq g_j \\ \theta(node) &= \forall a_j \in \eta \mid \text{decrease count}(a_j) \geq T \\ c_T(node) &= \begin{cases} F, & \text{if } \eta \neq \emptyset \wedge \theta = \emptyset \\ 0, & \text{if } \eta = \emptyset \vee \theta \neq \emptyset \end{cases} \end{aligned} \tag{3-2}$$

Note that the terminal cost penalty function described in Equation 3-2 requires only a single agent to be decreasing within the set of selected agents $\mathcal{R}_i$. In a scenario where a number of agents are involved in a deadlock while at least one selected agent is not involved (and

decreasing), the penalty will not be activated to resolve the deadlock. It is however assumed that eventually all of the agents hindering the decentralized recognition of a deadlock will either reach their goal (and no longer belong to η) or leave the proximity of the stuck agents and therefore will no longer be considered in the set of selected agents $\mathcal{R}_i$.
Figure 3-7a and Figure 3-7b show the first step planning result of enhancing the cost function c_0 with the penalty function described in Equation 3-2. Figures 3-6c and 3-7a show comparative results in both cases with $w = 6$ and Figures 3-7b and 3-7c correspondingly for $w = 5$. Finally, Table 3-1 reports the receding horizon implementation results for the same MAPF problem instance as considered in Figures 3-6 and 3-7 for an action implementation horizon of length $m = 1$. Clearly, using the supplemented terminal cost penalty function ameliorates solving this scenario for planning horizon lengths that are relatively shorter compared to the minimum amount of steps required to solve the global problem. If supplemented with the terminal cost penalty function, the algorithm is able to solve the problem for a planning horizon length as short as $w = 5$. Slightly different values for the decreasing agent threshold T didn't have an impact on the final result. This threshold value is expected to be mostly dependent on the selected planning horizon length w and should be tuned accordingly as a hyperparameter.

Table 3-1: Receding horizon implementation ($m = 1$) results for different lengths planning horizon w for baseline cost function c_0 without **(a)** and with **(b)** terminal cost penalty. The agent columns indicate if an agent has reached its goal and the final column shows total number of steps after which the solution no longer changes

(a) c_0 without terminal cost penalty

w	a_1 (red)	a_2 (orange)	a_3 (blue)	steps
9	✓	✓	✓	9
8	✓	✓	✓	9
7	✓	✓	✗	8
6	✗	✓	✗	4
5	✗	✓	✗	4
4	✗	✓	✗	4
3	✗	✗	✗	3

(b) c_0 with terminal cost penalty

w	T	a_1 (red)	a_2 (orange)	a_3 (blue)	steps
9	3	✓	✓	✓	9
8	3	✓	✓	✓	9
7	3	✓	✓	✓	9
6	3	✓	✓	✓	9
5	3	✓	✓	✓	11
4	2	✓	✓	✗	7
3	1	✗	✗	✗	4

3-4 Path communication and conflict resolution

This final section will aggregate all previous parts into the final Algorithm 2. The baseline solution scheme was already introduced in subsection 3-3-2 as well as the application of a random prioritization scheme. The prioritization scheme together with the baseline solution essentially serve as the backbone of the algorithm to ensure that the path allocation step in the receding horizon planning will always be conflict free. Before the solving process through running the algorithm in parallel is commenced, all agents draw a random priority value from a uniform distribution as shown in line 4. This priority value will be used to resolve a conflict over shared variables. The baseline solutions (wait actions for m time steps) of connected agents are filtered from an agent's MDD in line 9. During each planning round, lines 6-25,

all agents will secure a number of m actions to perform before starting the next planning round until the total simulation time has passed. Securing a path within a planning round can take multiple path securing iterations where all agents start with an invalid solution and execute lines 14-22 to secure a desired path over the planning horizon w. A desired path can be secured if it has no conflicts with higher priority agents (lines 19-21). If an agent was not able to secure a desired path, it will repeat the path search (lines 14-22). As a result of the prioritization scheme (combined with the baseline solutions), each negotiation round (lines 19-21), at least one agent will be able to secure a path and therefore the number of negotiation rounds in finite. Note that if the shared decentralized path selection policy (line 17) can successfully induce cooperation, all agents can secure a path simultaneously during the first negotiation round as all desired paths will comply. If an agent has to repeat its path search because it wasn't able to secure a path in the negotiation round, it filters the paths of the set of agents with secured paths within its set of connected agents $\Lambda_i \subseteq \Gamma_i$ in line 16.
Ideally, line 17 of Algorithm 2 consists of a tractable local MAPF problem optimization as

Result: parallel conflict free movement action selection
1 **Input** agent set $\mathcal{A}$ with set of start coordinates s and set of goal coordinates g
2 initialize $n = t_{sim} = 0$
3 Each agent in parallel, **do**
4 draw ε_i from a uniform distribution $(0, 1)$
5 set start location as current location $\pi_i(0) \leftarrow s_i$
6 **while** $n < t_{final}$ **do**
7 infer the set of connected agents $\Gamma_i(n)$ by Euclidean distance:
8 construct MDD_i^{δ}, $\delta = w$
9 filter the baseline solutions from $a_j \in \Gamma_i(n)$: $\pi_{j\in\Gamma_i}(n, \ldots, n+m)$ from MDD_i^{δ}
10 select first w levels of MDD_i^{δ} and send to connected agents
11 infer internal dependencies and append to MDD_i^{δ}
12 communicate new MDD to connected agents to infer external dependencies
13 sort $\Gamma_i(n)$ according to the Manhattan distance to the searching agent
14 **while** *solution* $_i$ = *INVALID* **do**
15 collect the set of agents with valid solutions $\Lambda_i(n) \subseteq \Gamma_i(n)$
16 filter $\pi_{j\in\Lambda_i}(n, \ldots, n+w) \backslash \pi_i(n)$ from MDD_i^{δ}
17 select a desired path according to the shared decentralized policy
18 communicate selected path to connected agents $a_j \in \Gamma_i(n)$
19 **if** *no conflicts with received paths of higher priority agents* **then**
20 *solution* $_i$ = VALID
21 **end**
22 **end**
23 perform selected action: $\pi_i(n+1, \ldots, n+m) \leftarrow \text{action}_i$
24 $n = n + m$
25 **end**

Algorithm 2: Receding horizon path negotiation algorithm with conservative baseline solution

described in section 3-3 that results in a desired path with w actions for agent a_i. However, for

some scenarios the local optimization problem still might not render tractable and therefore a common decentralized policy was designed to resolve these scenarios. Algorithm 5 shows the decentralized desired path selection policy which is used in line 17 of Algorithm 2. Line 6 of Algorithm 3 is activated if the local MAPF optimization is no longer considered tractable. Consequently, a desired path is chosen greedily (according to zero delay in the MDD) in line 7. It was pointed out in section 3-3 that the number of r agents to consider in a local optimization problem is likely less than the total number of connected agents and therefore a set of selected agents for optimization $\mathcal{R}_i \subseteq \Gamma_i$ is constructed in line 4. Additionally, a feature was added to "artificially" increase the number r: one can safely assume that agents which are already at goal will only move if required to by agents that are not at goal. Therefore, when constructing the set of selected agents, all connected agents that are already at goal are added to the initial set of ignored agents Σ_i. If it turns out that any of the resulting paths of the selected agents crosses a location of an agent in Σ_i, this agent is removed from Σ_i and the set of selected agents $\mathcal{R}_i$ is reconstructed. This scheme was implemented by considering an intermediate solution ($int.solution_i$) for agent a_i that is obtained in lines 3-14 which can only be valid if either the local MAPF problem is considered intractable or if all paths of connected agents ($\mathcal{R}_i$) do not interfere with the agents in Σ_i.

Result: desired path of agent a_i over planning horizon w
1 **Input** $\Gamma_i(n)$, $\text{MDD}_j^\delta \ \forall a_j \in \Gamma_i(n)$
2 initialize the set of ignored agents $\Sigma_i(n) \subseteq \Gamma_i(n)$ with agents that were at goal in time step $n-1$
3 **while** $int.solution_i$ = *INVALID* **do**
4 construct the set of selected agents $\mathcal{R}_i(n) = \Gamma_i(n) \setminus \Sigma_i(n)$, such that $|\mathcal{R}_i(n)| \leq r$
5 select path as a result of optimal MDD product graph search according to cost function f with terminal cost c_T
6 **if** *# expanded nodes has exceeded the maximum # of search iterations* **then**
7 choose greedy path
8 $int.solution_i$ = VALID
9 **else if** *any of the obtained paths contains a node of an agent in* $\Sigma_i(n)$ **then**
10 remove this agent from $\Sigma_i(n)$
11 **else**
12 $int.solution_i$ = VALID
13 **end**
14 **end**

Algorithm 3: Decentralized desired path selection policy

Chapter 4

Empirical performance evaluation

This chapter will report the simulation results that have been collected by running the designed algorithm over a number of multi-agent path finding (MAPF) instances. The designed algorithm is named the Decentralized Optimization (DECOP) algorithm for easier reference. After describing the experiment setup and selection of hyperparameters, the performance evaluation is presented. The empirical performance evaluation based on simulation results consists of three parts. Firstly, three different variants of DECOP are compared to other decentralized MAPF algorithms conform the performance evaluation structure as presented by reinforcement learning (RL) algorithms. Secondly, DECOP is compared to one specific fast decentralized MAPF algorithm, namely the priority inheritance with backtracking (PIBT) algorithm. Finally, the algorithmic behaviour for MAPF instances where DECOP is unable to attain a global solution is analysed.

4-1 Simulation experiment setup

In order to evaluate the performance of the distributed algorithm, the MAPF solution characteristics with respect to a number of MAPF instance types will be analyzed. Each instance type consists of a grid MAPF problem with the following initial properties:

- **map size** the considered grid maps are square
- **agent number** the total number of agents participating in a (one-shot) MAPF problem.
- **obstacle density** a percentage of random grid locations are selected as obstacle and cannot be used by the agents.
- **start locations** each of the agents has a dedicated start location which does not overlap with start locations of other agents or with locations of obstacles.
- **goal locations** each of the agents has a dedicated goal location which does not overlap with goal locations of other agents or with locations of obstacles (start and goal locations can overlap).

The MAPF instance types are analogous to those used by the considered RL algorithms for straightforward comparison. The following instance types of agent number - map size (corresponding agent density) will be evaluated: 8 agents on a 10x10 map (8%), 16 agents on a 20x20 map (4%), 32 agents on a 30x30 map (3.6%). Additionally, for the in depth analysis an instance type with a higher agent density is also considered: 20 agents on a 10x10 map (20%). Each of these combinations will be considered for a obstacle density percentage of 0%, 15% and 30%, resulting in a total of nine different instance types. For each MAPF instance type, one hundred random instances are created and the experiments are run on a server with an Intel Xeon E5-2637 (v3) CPU (4 cores). Although the DECOP algorithm is a fundamentally decentralized algorithm that can be executed in parallel by multiple agents, in this simulation setup the execution is sequential rather than parallel such that it is performed by a single computational entity. That is, the single CPU computes the outcomes of a path secure iteration for all agents sequentially. The total simulation could be readily adapted such that the computations of each individual agent is performed by a single computational unit. A MAPF instance is considered to be solved if a set of conflict free paths has been obtained such that all agents reach their goal location (see section 2-1 for formal definition). The total number of time steps required before all agents have reached their goal is referred to as the "problem depth". The problem depth is actually equal to the path length of the agent that is last one to arrive at its goal. An agent is considered to be at goal if it has to perform merely wait actions, at its goal location, up to the problem depth. Note that an agent that reaches its goal location in some time step is therefore not considered to be at goal if it has to leave its goal location again in a later time step. The time step in which an agent is considered to be at goal is referred to as the "individual agent problem depth" or "agent depth" for short.

4-2 Hyperparameter selection

The DECOP algorithm uses a number of hyperparameters during execution. The hyperparameters are shown in Table 4-1 with a corresponding description and assigned value. All of the parameters described in Table 4-1 have already been introduced in chapter 3 except the hyperparameter P. P designates the maximum number of nodes to be expanded when performing a graph search in the MDD product and was set to 20000 to limit the amount of memory required by the graph search. Furthermore, the other hyperparameters are already fixed are δ, m and u. The delay parameter δ was set equal to the planning horizon length w such that the most delayed route alternative of staying at the same node during the full planning horizon is included in the set of route alternatives. The parameter m was set to the minimum value of 1 to allow for maximal flexibility as local optimal planning is performed in each time step. Additionally, as the DECOP algorithm uses the baseline solution mechanism as described in subsection 3-3-2, a longer implementation window would lead to more restrictive planning as no other agents are allowed to use another agents current location over the whole implementation window m. The parameter u was set to a value of 5 which is equal to the communicative assumption made by the SCRIMP algorithm. The remaining hyperparameters (w,T,r) will be selected from Monte-Carlo simulation results on a number of MAPF instance types.

Table 4-1: Overview of hyperparameters used by the DECOP algorithm

parameter	description	value
w	length of the planning horizon	tuning
δ	maximum delay included in all multi-value decision diagram (MDD)	w
m	length of the implementation horizon	1
T	agent decreasing threshold	tuning
P	maximum number of nodes to expand from the MDD product	20000
r	desired size of the set of selected agents ($\lvert\mathcal{R}_i\rvert$)	tuning
F	terminal cost penalty for zero decreasing not at goal agents	100
v	Euclidean norm for two agents to be within communication range	5

4-2-1 Hyperparameter tuning

A number of hyperparameters shown in Table 4-1 have a value corresponding to "tuning". The values of these hyperparameters will be selected according to the results that follow from running a Monte-Carlo simulation with different value combinations of these parameters. The MAPF instance types used for Monte-Carlo simulation are the middle size MAPF instances with 16 agents on a 20x20 map with 0%, 15%, and 30% obstacle rates. Since the hyperparameter r relatively has the largest impact on the computational complexity and therefore the computation time, only two cases will be considered: $r = 4$ and $r = 5$. The exemplary MAPF instances used to design the terminal cost function in subsection 3-3-4 suggested that a longer planning horizon will better enable the local planning to find local solutions with a longer problem depth. Two planning horizon lengths $w = 8$ and $w = 12$ will be tested to evaluate the expected effect of incorporating a longer planning horizon. Finally, the decreasing threshold value T is tested for three different values with respect to the planning horizon length.

Table 4-2 shows the result of the Monte-Carlo simulation for $r = 4$ and Table 4-3 shows the result of the Monte-Carlo simulation for $r = 5$. The success rate (SR) key performance indicator (KPI) indicates the amount of MAPF instances for which a global solution was obtained. The "common maps" value displays the number of MAPF instances for which all combinations of hyperparameters have obtained global solution. The remaining values in the table are averaged only over those commonly solved MAPF instances to allow for fair comparison. The "δ" KPI represents the delay of an agent as a result of the obtained solution compared to its path length in a scenario where no other agents would have be present in the MAPF instance. Correspondingly, the "$\delta/_{depth}$" KPI denotes this delay relative to the path length in a scenario where no other agents would have be present in the MAPF instance. The relative delay is limited to 100%. The "expansion limit$_1$" KPI indicates the number of times that the node expansion limit was reached by at least one agent in each first path securing iteration compared to the total number of planning rounds. Similarly the "expansion limit$_2$" KPI shows the total number of times that the node expansion limit was reached in an agent's first path securing iteration compared to the total number of first path securing iterations (# planning rounds × # agents). Reaching the node expansion limit forces an agent to choose a greedy path instead of applying the common decentralized policy.
It can be concluded from these results that increasing the parameter r form 4 to 5 yields more complex local MAPF problems that exceed the node expansion limit more often and

Table 4-2: MAPF planning results on a 20x20 map with 16 agents with different obstacle rates. The desired selected agent set size (r) is 4. The results are given for different values of hyperparameters w (planning horizon) and T (decreasing threshold). The action implementation length m is equal to 1. A value in brackets $(\cdot)$ denotes a standard deviation from the corresponding mean value

obst. rate	com. maps	w	T	δ	δ/depth	exp. limit_1	exp. limit_2	SR	average runtime	total time
0%	100	8	2	0.19(0.60)	1.7%(6.3)	0.00%	0.00%	100%	3.4m(2.0)	5.7h
		8	4	0.19(0.60)	1.7%(6.3)	0.00%	0.00%	100%	3.3m(1.9)	5.6h
		8	6	0.20(0.65)	1.7%(6.5)	0.07%	0.00%	100%	3.6m(2.1)	6.0h
		12	3	0.19(0.59)	1.6%(6.2)	0.71%	0.09%	100%	9.3m(6.7)	15.5h
		12	4	0.19(0.59)	1.6%(6.2)	0.78%	0.09%	100%	9.1m(6.6)	15.1h
		12	6	0.19(0.58)	1.6%(6.2)	0.78%	0.09%	100%	9.4m(6.8)	15.7h
15%	91	8	2	0.54(1.5)	4.1%(12)	0.00%	0.00%	93%	1.2m(1.2)	2.0h
		8	4	0.59(1.7)	4.5%(13)	0.04%	0.00%	92%	1.3m(1.1)	2.1h
		8	6	0.62(1.7)	4.9%(14)	0.40%	0.04%	94%	1.5m(1.4)	2.5h
		12	3	0.55(1.5)	4.2%(12)	0.12%	0.01%	92%	2.6m(2.4)	4.3h
		12	4	0.55(1.5)	4.2%(12)	0.24%	0.02%	92%	2.5m(2.5)	4.2h
		12	6	0.56(1.6)	4.2%(12)	0.28%	0.03%	94%	2.7m(2.7)	4.6h
30%	82	8	2	2.1(4.1)	12%(23)	0.20%	0.02%	86%	1.7m(2.6)	2.9h
		8	4	2.2(4.3)	12%(23)	0.10%	0.01%	89%	1.9m(2.9)	3.2h
		8	6	2.2(4.3)	13%(24)	0.55%	0.05%	87%	2.2m(3.1)	3.7h
		12	3	2.0(4.1)	12%(23)	1.03%	0.09%	94%	3.4m(6.0)	5.7h
		12	4	2.0(4.1)	12%(23)	1.13%	0.10%	94%	3.4m(5.7)	5.6h
		12	6	2.1(4.4)	12%(23)	1.30%	0.11%	92%	3.9m(6.4)	6.4h

it takes significantly more time to attain a global solution. As the success rates with $r = 5$ are not higher compared to $r = 4$, r was selected to be 4 to reduce computation time. The remaining hyperparameters w and T will therefore be selected from the results in Table 4-2. For the 0% obstacle rate MAPF instance type, different combinations of w and T do not yield notably different results apart from runtime, likely because agents on average incur very little delay ($\sim$1.6%) in all cases. For a longer planning horizon $w = 12$, the node expansion limit is reached slightly more often and the run times increase probably just due to the fact that the local problems are larger in size. An analogous conclusion can be drawn for the MAPF instance type with a 15% obstacle rate. Finally, the results for the MAPF instance type with a 30% obstacle rate show more significant differences with respect to the success rate and incurred delay for different combinations of w and T. The longer planning horizon $w = 12$ yields a higher success rate compared to $w = 8$ for all cases of T. Furthermore, the success rate and performance of $w = 12$ with $T = 3$ and $T = 4$ are better compared to $T = 6$ but $T = 3$ has a slightly lower amount of times that the node expansion limit is reached by at least one agent in the first path securing iteration, compared to $T = 4$. The remaining hyperparameters w and T are therefore selected to be equal to $w = 12$ and $T = 3$.

Table 4-3: MAPF planning results on a 20x20 map with 16 agents with different obstacle rates. The desired selected agent set size (r) is 5. The results are given for different values of hyperparameters w (planning horizon) and T (decreasing threshold). The action implementation length m is equal to 1. A value in brackets $(\cdot)$ denotes the standard deviation from the corresponding mean value

obst. rate	com. maps	w	T	δ	δ/depth	exp. limit_1	exp. limit_2	SR	average runtime	total time
0%	100	8	2	0.18(0.56)	1.6%(5.9)	1.53%	0.13%	100%	13m(10)	21h
		8	4	0.18(0.56)	1.6%(5.9)	1.53%	0.13%	100%	13m(10)	21h
		8	6	0.19(0.63)	1.7%(6.4)	1.60%	0.14%	100%	12m(10)	20h
		12	3	0.18(0.58)	1.7%(6.9)	6.44%	0.71%	100%	26m(22)	43.3h
		12	4	0.18(0.55)	1.6%(6.1)	5.76%	0.66%	100%	28m(25)	47h
		12	6	0.18(0.55)	1.6%(6.1)	5.71%	0.66%	100%	27m(23)	45h
15%	91	8	2	0.51(1.6)	3.8%(12)	0.28%	0.03%	93%	2.9m(5.1)	4.9h
		8	4	0.54(1.7)	4.0%(12)	0.32%	0.03%	93%	3.1m(4.4)	5.1h
		8	6	0.56(1.7)	4.4%(14)	0.53%	0.05%	93%	3.0m(2.9)	5.1h
		12	3	0.49(1.5)	3.8%(11)	0.69%	0.07%	94%	5.3m(6.5)	8.8h
		12	4	0.49(1.5)	3.8%(11)	0.81%	0.09%	95%	5.7m(7.1)	9.4h
		12	6	0.52(1.6)	4.0%(12)	0.85%	0.10%	95%	5.9m(7.0)	9.8h
30%	74	8	2	1.6(3.6)	10%(21)	1.25%	0.15%	91%	4.2m(6.3)	7.0h
		8	4	1.6(3.7)	10%(21)	1.37%	0.14%	92%	4.5m(7.0)	7.4h
		8	6	1.6(3.7)	10%(21)	1.73%	0.20%	84%	4.8m(8.2)	8.0h
		12	3	1.5(3.2)	9%(21)	2.77%	0.36%	92%	6.6m(9.1)	11.1h
		12	4	1.6(3.3)	9%(21)	2.92%	0.37%	91%	7.4m(10)	12.3h
		12	6	1.4(3.1)	9%(20)	3.02%	0.38%	93%	8.0m(14)	13.3h

4-3 Comparison with other decentralized algorithms

The DECOP algorithm will be compared to state-of-the-art decentralized MAPF algorithms. To the authors best knowledge, there is no other decentralized algorithm that applies parallel local receding horizon optimization and therefore a number of other types of decentralized algorithms was selected for comparison. The SCRIMP algorithm [38] is considered to be the state-of-the-art decentralized RL MAPF algorithm and the algorithms reported in the comparison results of SCRIMP will be included as well which are the distributed, heuristic and communication (DHC) algorithm [22] and the prioritized communication learning (PICO) algorithm [21]. From other types decentralized algorithms the PIBT algorithm [24] is considered state-of-the-art for decentralized fast MAPF algorithms. The extended version (windowed priority inheritance with backtracking (winPIBT) [25]) is still under development and doesn't offer a fundamental advantage yet.
The KPIs are initially set equal to those considered in the results of SCRIMP to allow for straightforward comparison. In order to be able to properly compare the results according to these KPIs, also the types of MAPF problem instances are equal to the ones considered in SCRIMP. The first KPI that will be considered is the SR which is the percentage of MAPF instances of a certain MAPF instance type that has been solved successfully. The second KPI that is considered by SCRIMP is the problem depth (the amount of time steps it took to solve a MAPF instance) from which the average of solved MAPF instances is taken. Two

additional KPIs used in SCRIMP are left out because, 1) the first additional KPI refers to the amount of occurring conflicts which is relatively low in SCRIMP and doesn't occur in DECOP, 2) the KPI showing to what extent an unsolved MAPF instance was successfully solved has been replaced by a more focused analysis in section 4-5.

Table 4-4: selected weights corresponding to the designed cost function in Equation 3-1, resulting in three variants of the DECOP algorithm

Variant	w_1	w_2	w_3
DECOP_0	1	0	0
DECOP_1	$3 \cdot w \cdot (\lvert\mathcal{R}_i\rvert - 1)$	0	1
DECOP_2	$3 \cdot w \cdot \lvert\mathcal{R}_i\rvert$	1	0

Three different variants of the DECOP algorithm have been included which refer to different cost functions with the same hyperparameters as selected in section 4-2.The weights assigned corresponding to the cost function in Equation 3-1 to create different variants of the DECOP algorithm and the selected values are shown in Table 4-4. The DECOP_0 variant deploys a vanilla variant of the node cost function that assigns a cost to each node corresponding only to the combined incurred delay with respect to a node in the MDD product. The DECOP_2 variant assigns a value of $3 \cdot (\text{problem depth}) \cdot |\mathcal{R}_i|$ to w_1 which is set such that any joint path over the full planning horizon having the maximum amount of dependencies (3 types of dependencies times the total depth times the total amount of selected agents) is still favoured over a joint path having a single step of delay with no dependencies. In the DECOP_2 variant $|\mathcal{R}_i| - 1$ is used instead of $|\mathcal{R}_i|$, since the searching agent cannot have external dependencies. DECOP_1 and DECOP_2 are therefore still able to find the local optimum solution (with minimum delay) but can also distinguish between equal cost solutions by considering the amount of external and internal dependencies respectively. Distinguishing between equal cost solutions by favouring a solution with fewer dependencies is expected to improve induced cooperation through parallel decision making and ameliorate the solvability of future local problems regarding the receding horizon implementation.

Table 4-5 shows the results as reported by SCRIMP [38]. The displayed SCRIMP variant is reported in [38] as "SCRIMP-local". The results of a suboptimal centralized algorithm (OdrM* with inflation factor $\varepsilon = 2.0$ and a runtime limit of five minutes) as reported in [38] are included as well as a reference. Overall, the variants of the DECOP algorithm show very similar performance to the suboptimal centralized (ODrM*) reference algorithm. In all cases except for 30% 16 and 32 agents, the DECOP algorithm shows lower average problem depths compared to PIBT and in all cases the DECOP variants report lower average problem depths than any of the RL algorithms. For the 30% obstacle rate on all MAPF instance sizes, both PIBT and the DECOP algorithm do report a lower success rate compared to SCRIMP. Interestingly, the success rates on the 32 agent 30 by 30 map size differ significantly between the different DECOP variants. To investigate this phenomenon, an in depth analysis will be performed on for the 30 by 30 MAPF instance types.

Table 4-5: Performance results with three variants of the DECOP algorithm compared to other decentralized (RL) MAPF algorithms for different MAPF instance types. A value between brackets $(\cdot)$ denotes the standard deviation to the corresponding mean value.

algorithm	**average problem depth**			**success rate (SR)**		
8 agents in 10×10 world with $0\%, 15\%, 30\%$ obst. rate (columns)						
ODrM*	12.64(2.13)	13.72(2.46)	16.16(4.03)	100%	100%	100%
DHC	14.34(5.08)	17.23(5.96)	29.82(20.93)	100%	98%	92%
PICO	17.35(12.39)	29.23(28.71)	35.04(26.93)	100%	63%	29%
SCRIMP	14.40(2.57)	18.30(18.84)	20.95(10.23)	100%	100%	97%
PIBT	13.45(2.62)	14.84(3.00)	17.52(8.70)	100%	99%	86%
$DECOP_0$	12.30(2.35)	13.57(2.61)	15.78(4.77)	99%	96%	93%
$DECOP_1$	12.25(2.30)	13.75(3.28)	15.51(4.38)	100%	97%	90%
$DECOP_2$	12.26(2.42)	13.51(2.73)	15.54(4.57)	100%	99%	91%
16 agents in 20×20 world with l0%, 15%, 30% obst. rate (columns)						
ODrM*	27.38(3.63)	27.64(3.74)	34.61(6.55)	100%	100%	95%
DHC	29.15(4.66)	34.72(8.44)	55.71(26.75)	100%	100%	83%
PICO	29.83(5.80)	61.83(30.73)	-	100%	18%	0%
SCRIMP	26.47(4.09)	28.46(3.43)	41.30(20.07)	100%	100%	97%
PIBT	27.76(4.25)	27.80(4.21)	35.17(6.90)	100%	98%	92%
$DECOP_0$	26.80(4.16)	27.48(4.19)	36.00(8.82)	100%	92%	94%
$DECOP_1$	26.76(4.16)	27.41(4.36)	36.16(8.71)	99%	94%	92%
$DECOP_2$	26.84(4.04)	27.40(3.99)	36.26(8.15)	100%	99%	97%
32 agents in 30×30 world with l0%, 15%, 30% obst. rate (columns)						
ODrM*	42.97(4.62)	43.39(4.97)	53.95(10.43)	100%	100%	97%
DHC	48.64(17.54)	52.48(7.10)	91.29(29.25)	100%	98%	78%
PICO	65.14(10.30)	-	-	14%	0%	0%
SCRIMP	43.29(4.96)	46.53(7.00)	69.10(31.62)	100%	100%	97%
PIBT	44.19(4.78)	43.66(5.19)	52.37(7.42)	100%	100%	84%
$DECOP_0$	43.28(4.74)	43.41(5.31)	56.53(9.49)	100%	99%	66%
$DECOP_1$	43.20(4.69)	43.26(5.35)	55.52(9.56)	99%	95%	79%
$DECOP_2$	43.32(4.70)	43.36(4.84)	55.60(9.96)	100%	98%	88%

4-4 In depth analysis together with the PIBT algorithm

The performance differences of the DECOP algorithm variants on the MAPF instance type of 32 agents with a size of 30 by 30 presented in Table 4-5 are investigated using the KPIs introduced in section 4-2. The planning results of the PIBT algorithm are added as a reference as this algorithm is really fast and also complete for open maps[24]. The planning results according to the introduced KPIs are shown in Table 4-6. The KPIs used in Table 4-6 are mostly equal to the ones introduced in section 4-2 but the "expansion $limit_2$" KPI has been replaced by a "compliance" KPI. The compliance KPI denotes the amount of time steps in which all agents were simultaneously able to selected a complying solution during the first path securing iteration compared to the total number of planning rounds. The compliance KPI therefore gives an indication of the extend to which cooperation was successfully induced through the decentralized policy. The values with respect to all KPIs except the success rate

are shown only with respect to the set of commonly solved MAPF instances which allows for a better comparison. The planning results according to the introduced KPIs are shown in

Table 4-6: MAPF planning results for two different MAPF instance types with 0%, 15% and 30% obstacle rates. The hyperparameters of the DECOP algorithm are set as described in section 4-2. A value between brackets $(\cdot)$ denotes the standard deviation to the corresponding mean value.

obst. rate	com. maps	method	δ	δ/depth	exp. limit_1	compl.	success rate	average runtime
32 agents on 30 × 30 map - 3.6% agent density								
0%	99	PIBT	2.3(5.3)	14%(30)	-	-	100%	instant
		DECOP_0	0.25(0.65)	1.4%(4.1)	13%	37%	100%	83m(52)
		DECOP_1	0.23(0.73)	1.3%(4.5)	2.8%	35%	99%	42m(21)
		DECOP_2	0.21(0.59)	1.1%(3.5)	2.0%	45%	100%	34m(16)
15%	93	PIBT	2.8(5.8)	17%(31)	-	-	100%	instant
		DECOP_0	0.78(2.1)	4.0%(11)	0.20%	34%	99%	12m(7.5)
		DECOP_1	0.77(2.0)	3.9%(11)	0.20%	33%	95%	10m(5.7)
		DECOP_2	0.69(1.8)	3.5%(9.4)	0.30%	39%	98%	8.7m(4.4)
30%	49	PIBT	2.9(6.7)	13%(28)	-	-	84%	instant
		DECOP_0	2.1(5.5)	8.4%(20)	1.85%	29%	66%	9.8m(25)
		DECOP_1	2.1(5.2)	8.4%(20)	1.57%	29%	79%	8.8m(17)
		DECOP_2	2.0(5.0)	8.2%(20)	1.56%	32%	88%	8.4m(16)
20 agents on 10 × 10 map - 20% agent density								
0%	62	PIBT	0.93(2.1)	18%(36)	-	-	100%	instant
		DECOP_0	0.46(1.2)	9.9%(24)	0.00%	37%	74%	0.9m(0.9)
		DECOP_1	0.49(1.3	10%(24)	0.00%	32%	87%	1.8m(1.8)
		DECOP_2	0.52(1.4)	11%(25)	0.00%	31%	86%	1.0m(1.0)
15%	43	PIBT	1.2(2.8)	16%(34)	-	-	88%	instant
		DECOP_0	1.1(3.0)	14%(31)	2.10%	27%	71%	4.2m(11)
		DECOP_1	1.1(3.0)	14%(31)	2.23%	23%	74%	4.1m(8.6)
		DECOP_2	1.1(3.2)	15%(31)	1.94%	23%	72%	3.6m(9.4)
30%	3	PIBT	0.17(1.4)	1.6%(12)	-	-	44%	instant
		DECOP_0	0.17(1.5)	1.5%(11)	3.23%	21%	17%	0.4m(2.6)
		DECOP_1	0.14(1.2)	1.6%(12)	5.08%	17%	22%	0.8m(4.8)
		DECOP_2	0.20(1.8)	1.8%(13)	7.59%	15%	29%	1.1m(6.6)

Table 4-6. Whereas the values with respect to the problem depth KPI of Table 4-5 between the PIBT and DECOP did not seem to differ significantly, the delay and relative delay KPI in table Table 4-6 show more significant differences. The compliance values are higher for the variant of the DECOP algorithm that uses the amount of internal dependencies to distinguish equal cost solutions (DECOP_2) only for the 30 by 30 MAPF instance types which have a lower agent density. Interestingly, for the MAPF instance type of 32 agents on a 30 by 30 open map, the DECOP_2 variant the node expansion limit is reached significantly fewer times by any agent in the first path securing iteration compared to the other variants. This likely also explains the significantly lower computation time. For the MAPF instance with 20 agents on a 10 by 10 map, the patterns are less clear as the compliance value of the vanilla DECOP_0 variant is higher compared to the other variants. In all cases, the DECOP_0 variant reports

a lower success rate compared to the extended variants. Note that the MAPF instance type with 20 agents on a 10 by 10 map with a 30% obstacle density shows KPI values over just 3 commonly solved instances, therefore only the SR KPI is of statistical significance for this MAPF instance type.
The results for the remaining MAPF instance types that were tested for the introduced KPIs show similar but less explicit patterns in the respective values and can be found in the Appendix section B-1.

4-5 Algorithm behaviour analysis

The unsolved MAPF instances of the DECOP algorithm as reported in the previous sections can all be characterized according to (possibly a combination of) one of three scenarios. Examples of these three scenarios will be reported in this section. Each example is a MAPF problem instance from the 20x20 - 16 agents - 30% obstacle density problem set for which no global solution was obtained. The examples all show the final implementation step before either all agents would persist in recurring actions (stay at the same location or oscillate between the same locations).

Too many agents requesting the same variable

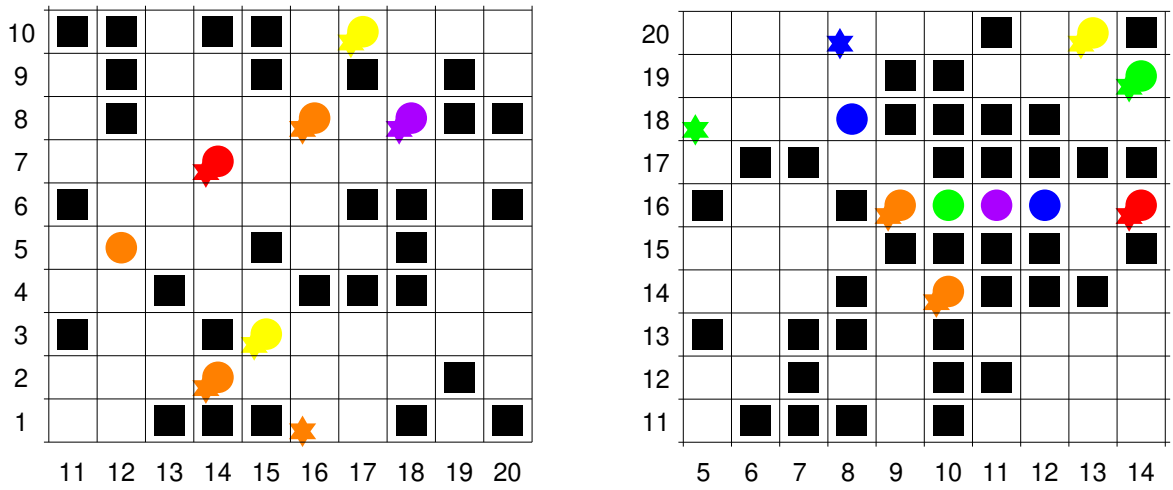
Secondly, if a scenario requires coordination with a number of agents more than r, the DECOP will likely fail to find a global solution. For example Figure 4-1b shows a scenario where five agents have to coordinate the utilization of the same corridor. The goal of a blue agent at (8,20) corresponds to the blue agent at (12,16). The goal location of the other blue agent at (8,18) is not shown in this figure but this blue agent does have to cross this corridor in opposite direction. Since the hyperparameter r was set to 4, the orange agent at (9,16) cannot incorporate both blue agents at (8,18) and (12,16).

Alternation of desired path

Finally, in some cases an agent alternates between two feasible solutions by selecting the implementation of a different solution each consecutive planning rounds. Figure 4-1a shows a MAPF configuration in which the orange agent at (12,5) can choose between two routes to evade the obstacles at (13,4) and (14,3). After selecting one of the two options and implementing the first step ($m = 1$), the agent opts to use the other route instead, resulting in persistent alternation of the same actions.

Multiple equal cost local solutions

Firstly, the local optimization problem might contain two solutions of identical cost and agents participating in this local problem do not select the same desired local solution. An example of this scenario is shown in Figure 4-2 where agent a_1 (red at (1,13)) selects a different

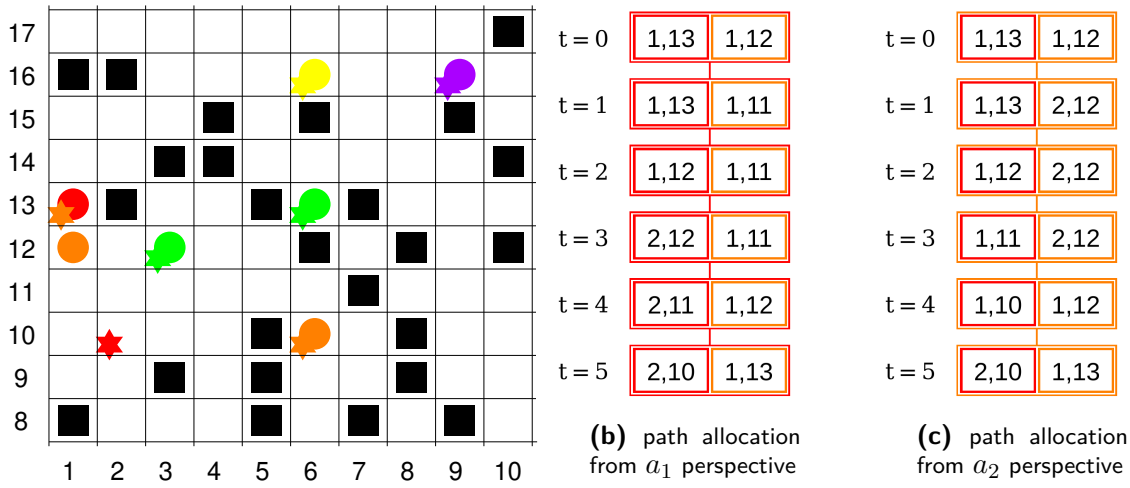


(a) MAPF problem instance that is not solved because the orange agent at (12,5) alternates between two different solutions in consecutive planning rounds.

(b) MAPF problem instance where more many agents than the selected agent set size $r = 4$ have to coordinate the use of the same shared variables (the corridor).

Figure 4-1: Exemplary scenarios in which the DECOP algorithm fails to attain a global solution

evasion manoeuvre than agent a_2 (orange at(1,12)). Figures 4-2b and 4-2c show the selected local solutions of agents a_1 and a_2 respectively. Clearly, the selected desired paths conflict at $t = 3$. The higher success rate of the two variants of the DECOP algorithm that incorporate the amount of dependencies in their cost function can probably be explained due to having fewer occurrences of this scenario.



(a) MAPF problem configuration

(b) path allocation from a_1 perspective

(c) path allocation from a_2 perspective

Figure 4-2: MAPF problem instance that is not solved due to agents choosing different equal cost local solutions

Chapter 5

Discussion

This chapter presents a reflection on the results reported in chapter 4. A number of conclusions is drawn with respect to the performance of the Decentralized Optimization (DECOP) algorithm both in comparison to other algorithms and regarding the algorithmic behaviour. Finally, some suggestions for further investigation of different aspects of this type of framework are provided.

5-1 Algorithm performance

It can be concluded from the presented results that the proposed DECOP algorithm yields very effective solutions that are likely close to the global sum of costs (SOC) optimal solution. For most considered multi-agent path finding (MAPF) instance types, the DECOP algorithm reports a lower average problem depth compared to reinforcement learning (RL) algorithms while it is guaranteed to be conflict free and requires no training. It is however questionable if it can be concluded that the DECOP algorithm indeed consistently provides solutions with a shorter problem depth as the average problem depth values in Table 4-5 only consider the MAPF instances that have been solved by the respective algorithm. Therefore the conclusion whether the DECOP algorithm indeed consistently provides solutions with a shorter problem depth is highly dependent on the assumption that those MAPF instances that the algorithm didn't solve do not have a problem depth that is significantly longer compared to the average problem depth. For example, an algorithm that solves only one MAPF instance out of a hundred instances but does solve this instance with a problem depth of 1 will report a deceptively good average problem depth performance. Overall, the results in Table 4-5 depend on the assumption that the randomly created MAPF instances have comparable problem depths.
To overcome this issue, the in-depth results in Table 4-6 report all values, except for the success rate, only for the set of commonly solved MAPF instances. Another problem regarding the key performance indicator (KPI) of average problem depth is that it provides no information with respect to the paths of agents that arrive at their goal location before the last agent

does. If one were to consider the makespan objective for the optimal MAPF problem as described in subsection 2-1-1, this information would indeed not be relevant. For reasons described in subsection 3-3-4, the makespan objective is not suitable for a receding horizon MAPF algorithm and therefore the SOC objective function is used for optimization of the local MAPF problems. Additionally, using the SOC objective function likely is more relevant in practice as this objective can be adapted to accommodate a sequence of goal locations rather than a one-shot setting. In this regard, information about paths of all agents participating in a MAPF problem becomes relevant and therefore the KPIs of delay and relative delay have been introduced in section 4-2.
From the in-depth results shown in Table 4-6, it can be concluded that all variants of the DECOP algorithm consistently provide sets of paths that contain a lower combined delay over all agents compared to the priority inheritance with backtracking (PIBT) algorithm. The DECOP_2 variant, which considers the total amount of internal dependencies for equally delayed solutions, achieves a higher success rate and compliance value for the lower agent density MAPF types that were considered. For the higher agent density MAPF type (20 agents on 10 by 10 map), no clear conclusion can be drawn with respect to the induced cooperation based on the compliance values and success rates.
Similarly to the RL algorithms, the most challenging MAPF instances (and therefore unsolved instances) are instances with a higher obstacle density. This is likely due to the fact that these types of maps contain many corridors which pose a complex coordination task like the scenario described in section 4-5 where too many agents request the same variables. Although MAPF instances that require complex corridor coordination such as shown in Figure 4-1b can be solved by increasing r, w and u, this approach doesn't scale well. For example Figure 5-1 shows a similar case of an unsolved MAPF instance with more agents involved in the global problem which consequently increases the chance of more agents participating in a complex corridor situation. Since the (local) MAPF problem is proven to be of non polynomial (NP) time complexity, the approach of extending sensing and optimization range would eventually result in local MAPF problems that are too large to be solved in reasonable time. The detrimental effect of scaling is also evident in the success rates of the SCRIMP algorithm for MAPF instance types that were not reported in the results of Table 4-5. Whereas the SCRIMP algorithm is able to achieve a near perfect success rate for all MAPF instance types reported in this work, the success rates significantly drop for a MAPF instance type of 128 agents on a 40 by 40 map with 15% and 30% obstacle density. This is also likely due to cluttering of a large number of agents as a result of problematic scaling as displayed in Figure 5-1, which might just be impossible to solve using a decentralized MAPF algorithm.
Secondly, the unresolved MAPF instance scenario shown in Figure 4-1a can likely be resolved by increasing the number of implementation steps m. Increasing m to from $m = 1$ to $m = 2$ does implicate that the number of baseline selected agents (agents that can interfere with the searching agents within m steps) will be higher but this increase is likely to avoid agents alternating between two solutions.
The last scenario for which the DECOP algorithm was reported to fail in section 4-5 could be resolved by adding more properties to different equal cost solutions which would allow agents to distinguish equal cost solutions and subsequently follow the same policy. For the unresolved MAPF instance showcased in Figure 4-2 for example, the decentralized path searching policy could be extended such that in case of equal case solutions, a solution where agents pass each other on the right is always selected.

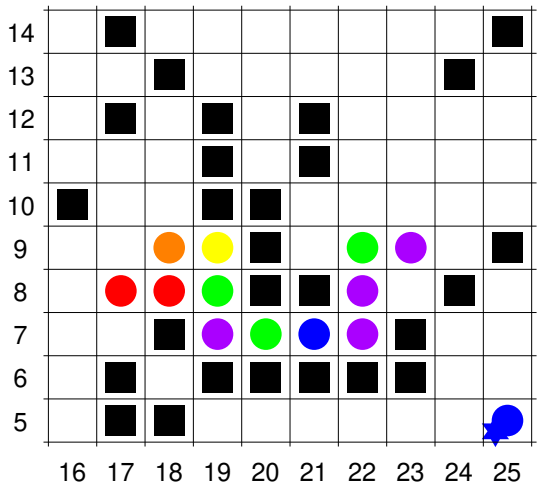


Figure 5-1: Example of an unsolved MAPF instance due to complex corridor coordination for a 20x20 size, 32 agents, 30% obstacle MAPF instance

5-1-1 Considered MAPF problems

From a general point of view, one might argue that the types of MAPF problems shown in Figure 4-1a and Figure 5-1 are less relevant to real world (logistical) applications as it would yield a very inefficient network, even in a centralized optimal setting. On the other hand, these types of MAPF problems (compared to open maps and maps with 15% obstacle density) might be the only of the considered low agent density MAPF problem instance types that incur coordination situations where a relatively large number of agents is located in near proximity such as shown in Figure 5-1. To investigate this hypothesis, a MAPF instance type was considered in Table 4-6 with a relatively higher agent density of 20% (compared to the considered low agent density rates of $\sim$4%). Although many agents being present close to each other on an open map is very different to a complex corridor scenario shown in Figure 5-1, the success rates for the DECOP algorithm variants reported in Table 4-6 for a 20 by 20 map with 20% agent density are indeed significantly lower compared to the same MAPF instance size with a lower agent density. It can therefore be concluded that the DECOP algorithm performs less well in scenarios where many agents are closely packed together. This is more or less to be expected from this decentralized framework because an agent will assume that at least one agent within its locally solved MAPF problem is able to move and can consequently not accommodate situations in which this is impossible from a global perspective. Strategies to overcome this issue will most certainly contain some form of induced communication like the backtracking strategy of the PIBT algorithm or using external dependencies introduced in subsection 3-2-2.

5-2 Conclusion

An overall conclusion is that in basic settings the DECOP algorithm works very well for MAPF instances with a relatively low agent density. Increasing the respective hyperparameters of the DECOP algorithm would not be an adequate approach to overcome the described

challenges as it would not scale well and eventually create intractable local MAPF problems. Although the DECOP algorithm is less effective for MAPF problems with a relatively high agent density compared to the PIBT algorithm, the cooperation that the DECOP algorithm is able to induce in complex corridor situations do yield a higher success rate for MAPF problems with a relatively high obstacle density.
Table 5-1 shows a comparison of algorithm features where "++" denotes good performance, "+" a slight advantage and "-" bad performance or not favourable for scalability. Since SCRIMP and PIBT apply a decentralized policy, both algorithms exhibit good performance with respect to the computational effort. The OrdM* algorithm requires centralized communication (and computation) and the PIBT algorithm demands induced sequential communication, both resulting in relatively bad performance with respect to the communicational effort. The DECOP algorithm requires no training and reports a high (individual agent) efficiency. The communicational effort in DECOP can be multi-step but the number of communication rounds is bounded by the (maximum) number of connected agents. Besides being much more efficient compared to the PIBT and SCRIMP algorithms, the DECOP algorithm can better cope with complex corridor scenarios using a dedicated deadlock detection mechanism. Although a higher computational effort is demanded by DECOP, this effort is distributed over all agents in the system.

Table 5-1: Algorithm feature comparison

method	training time	efficiency	communication	computation
ODRM*	++	++	-	-
SCRIMP	-	+	++	++
PIBT	++	-	-	++
DECOP	++	++	+	+

5-3 Future work

This research was mainly aimed to propose a new approach for solving the MAPF problem in a decentralized manner with high scalability. The reported results suggest that the proposed approach could be very effective as it requires no training and the hyperparameters can be tuned with respect to a particular MAPF problem type. To further investigate the characteristics of the parallel decentralized decision-making mechanism the following directions can be considered:

- the implementation of a decoupled MAPF optimization algorithm instead of the used graph search coupled increasing cost tree search (ICTS) algorithm. Although increasing the respective hyperparameters is argued to not scale well, a more efficient local optimal MAPF algorithm would enable increasing the size of the set of baseline selected agents which is expected to be beneficial to the performance in higher agent dense environments.

- in coherence with the previous point, the path search is currently stopped if the iterations exceed a certain limit. Ultimately there would be no iteration limit because the

local MAPF problems should be able to be readily solved within reasonable time. Discarding the iteration limit would enable further research into the induced cooperation through shared policy decentralized decision-making over path allocation.

- further investigation into the compliance of the first path securing iteration in each planning round. Instruments to consider are the weights of different dependencies in the cost function and for example the type of dependencies to consider.
- investigation of heuristic search that for example favours having fewer dependencies over delay, this mechanism could potentially further enable induced cooperation based on a common decentralized planning policy.
- incorporating policies for scenarios where agents can recognize that they are stuck or will get stuck, for example if agents have to fit through a single corridor.
- inclusion of some sort of priority values in the local optimization problem that are depended on the amount of incurred delay such that the delay can be balanced in an online manner.

Appendix A

A-1 Research paper

DECOP: a parallel local optimization framework for decentralized cooperative multi-agent pathfinding

Karel Scheepstra, Azita Dabiri, Bilge Atasoy

***Abstract*—Multi-agent path finding (MAPF) is the task of finding non-conflicting paths for multiple agents that operate in a environment with shared resources. Finding an optimal solution quickly becomes intractable for many applications and consequently suboptimal methods are also explored extensively in literature. This work presents the Decentralized Optimization (DECOP) algorithm: a novel receding horizon control algorithm that exploits insights from MAPF research as well as decentralized control. In the proposed framework, each travelling agent communicates with agents in its proximity to solve a local MAPF problem that considers only a selected tractable number of agents. Inter-agent cooperation and conflict free operation are induced through applying a common local optimization policy during parallel local optimization and through a subsequent path reservation scheme based on random priorities. Inter-agent communication consists of sharing respective route alternatives from which additional information with regard to an agents' entanglement can be inferred which can also be included in the local optimization cost function.**

Comparative results with other decentralized algorithms show that the DECOP algorithm yields competitive results while guaranteeing conflict free operations, with limited required communication and without the need of any training time. Among many degrees of freedom to be explored further, including information about the entanglements of an agent's route alternatives in the common policy for local optimization yields an increase in performance and suggests an increased extent of induced cooperation.

***Index Terms*—Multi-agent pathfinding, receding horizon control, decentralized control.**

I. Introduction

IN the transition towards automating transport operations that take many shapes in modern day societies, many challenges remain. Advancements in communication, sensing, and computation technologies enable the replacement of human decision making by automated intelligent decision making. The main drivers behind this transition process are the increase of human safety, reduced energy consumption and monetary profits. C hallenges r ange f rom i ncluding v ulnerable, a nd in general, non-automated road users [1], [2] to coordinating hundreds of robots in automated warehouses [3]. This work aims to investigate how local coordination through limited communication can improve global performance in the Multi-agent Pathfinding (MAPF) problem.

Multi-agent pathfinding i s t he t ask o f fi nding non-conflicting paths for multiple agents that operate in a environment with shared resources. The MAPF problem arises in many real world applications such as routing of robots in warehouses [3], baggage handling [4], or even micro droplet manipulation [5].

K. Scheepstra is with the Department of Maritime and Transport Technology at the Faculty of Mechanical Engineering, Delft University of Technology, Delft, NL,

A. Dabiri and B. Atasoy are with Delft University of Technology.

A. Related work

The MAPF problem is sometimes referred to as a pebble motion on graph (PMG) problem, which is an early identified variant of MAPF where only one pebble (agent) per time step is allowed to be moved to an unoccupied node. It has been proven that the PMG problem is solvable in polynomial time in [6] and that solution feasibility can be checked in linear time [7]. Predictably this also translates to solving MAPF problems with parallel moves [8]. A feasible solution solution for a MAPF problem is a set of non-conflicting paths. A MAPF problem is well-formed if there exists a set of paths such that none of these paths cross each other [9]. Besides a (possibly non-unique) set of conflict free paths, a MAPF application might consider some sort of global objective or cost function. The MAPF objective is considered on a network level, as it refers to the utilization effectiveness of a collection of shared resources: the network itself. Mainly two types of global objective are distinguished in MAPF research, firstly the **sum of cost (SOC) objective**: $\sum_{i=1}^{k}\sum_{n=0}^{N_i} C(\pi_i(n), \pi_i(n+1))$, which is the sum of each agent's cost of travelling the edges along the determined path where $C(\ \cdot\ ,\ \cdot\ )$ denotes the cost of traversing a particular edge. Secondly the **makespan objective**: $\max_{i\leq i\leq k} N_i$ minimizes the amount of time (steps) required before all agents have reached their goal node. It is has been proven that the optimal MAPF problem is of NP-hard complexity for both the SOC and makespan objective [8]. Finding an optimal solution with respect to one of the two introduced objectives therefore quickly becomes intractable for many applications and consequently suboptimal algorithms are also explored extensively in literature. In general, MAPF algorithms will feature one or more of the following properties:

- **optimality**, a number of algorithms is introduced to quickly obtain an optimal solution without exploring the entire search space (e.g. [10], [11]). In order to make algorithms more time efficient, (potentially bounded) suboptimal variants of optimal MAPF algorithms have also been developed (e.g. [12], [13], [14], [15]).
- **completeness / feasibility**, a complete MAPF algorithm will always yield a feasible solution. Several algorithms exist that yield feasible solutions for any MAPF problem with at least two unoccupied nodes. Furthermore, [16] points out that feasible solutions often exist for grid-based MAPF problems that allow the cycle conflict with

as many agents as nodes. A linear time algorithm to check feasibility is introduced in [17]. A prominent example of such an algorithm is the "Push and rotate" algorithm [18].
- **computational efficiency / tractability**, a MAPF algorithm is considered time efficient if its worst-case solving time is polynomial in the graph size and is not exponential in the number of agents [9]. For example the Hierarchical Cooperative A* (HCA*) algorithm [19] introduces a random hierarchy among agents and subsequently agents plan their paths sequentially.
- **compilation-based** Compilation-based MAPF algorithms reduce a MAPF problem to a formalism for which a solver already exists. Examples of such formalism's are SAT, MILP, ASP and CSP, an overview of compilation-based MAPF algorithms is provided in [20].

A centralized approach to optimal multi-agent path planning suffers from tractability issues. This motivates the search for suboptimal yet very usable algorithms. Alongside the development of centralized (potentially bounded) suboptimal algorithms, there is a growing interest in decentralized MAPF algorithms. Decentralized algorithms apply decentralized decision-making where different decision-making agents have access to only a limited part of the problem. Work on decentralized algorithms can roughly be divided into three different categories. Firstly, a number of algorithms adapt a centralized MAPF algorithm to work in a decentralized setting. For example the DMAPP algorithm [21] introduces a communication protocol to achieve a decentralized version of the HCA* algorithm and the DisCof algorithm [22] adapts the Push and rotate algorithm [18] to work in a decentralized setting through induced communication. The second category of decentralized algorithms applies multi-agent reinforcement learning (MARL) to the MAPF problem. While the first applications of MARL aimed to imitate an optimal centralized planner, the most recent contributions focus explicitly on learning (multi-step) decentralized communication (e.g. [23], [24], [25]). See [26] for an overview of decentralized reinforcement learning MAPF algorithms. The last category of decentralized MAPF algorithms considers designing algorithms which apply fundamentally (non RL) decentralized decision-making through (induced) communication. To the author's best knowledge, the only examples of algorithms that fall within this category are the priority inheritance with backtracking (PIBT) algorithm [27] and its extended version with longer planning windows: winPIBT [28].
Three main challenges can be recognised in realizing a decentralized MAPF algorithm: 1) identifying (possibly suboptimal) subsets of agents and thereby establishing communicative links between agents based on decentralized information, 2) the communication protocol or information to be communicated between agents and 3) the (re)planning strategy for individual agents' paths within a subset of agents to resolve or avoid conflicts.

B. Scope and contribution

The scope of this research is limited to the evaluation of a conceptional algorithm design that will be verified through simulation. Furthermore, this work aims to apply the widespread theoretical insights on solving the MAPF problem attuned to the boundary conditions of real world systems, using concepts of decentralized control theory.
The contribution of this work is threefold: 1) a novel framework called the Decentralized Optimization algorithm (DECOP) for decentralized MAPF that applies parallel receding horizon control is presented. 2) This work reports the first receding horizon control algorithm designed for the MAPF problem which was achieved by designing a terminal cost function that can avoid deadlocks. 3) The promising concepts that follow from route alternatives sharing that are developed in this work open up a new research area into exploiting dependency information to achieve induced cooperation through a common decentralized policy.

C. Outline

The remainder of this paper is structured as follows: after describing the formal problem definition in section II, the framework is introduced in section III where subsequently the DECOP algorithm is presented. The empirical simulation results and algorithm evaluation are reported in section IV. Finally, conclusions are drawn in section V, accompanied by a discussion on the obtained results and the suggested directions for future work.

II. Problem definition

The notion of *classical MAPF* as defined in [29] is a planning problem in which a node can be occupied by only one agent and all agents can either move or wait (in parallel) to a connected node in each time step. For a system consisting of a number of k agents where $\mathcal{A} = \{a_1, \ldots, a_k\}, |\mathcal{A}| = k$ describes the set of agents, the ith agent is referred to as a_i. The input to a classical MAPF problem is the 3-tuple $\langle \mathcal{G}, s, g \rangle$. The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph with nodes $v \in \mathcal{V}$ connected through edges $e \in \mathcal{E}$. The k-tuple s defines a list of start nodes for all agents where $s_i \in \mathcal{V}$ denotes the start node of the ith agent. Similarly, the k-tuple g defines a list of goal nodes where $g_i \in \mathcal{V}$ denotes the goal node of the ith agent.
The initial simulation time is denoted as t_0 and marks the beginning of the plan execution of each agent starting at its start node. Time is furthermore discretized such that $t = t_0 + n \cdot h$ with $n \in \mathbb{N}^0$. The time step h is normally set to 1 and traversing an edge usually takes exactly one time step (for unweighted graphs). A single agent path for the ith agent is defined by π_i as a sequence of $v \in \mathcal{V}$ nodes. The single agent path π_i takes a time step n as an argument to denote the particular node v that the ith agent occupies in time step n: $\pi_i(n) = v$. The number of nodes that an agent plans to occupy (including wait actions, the start node and the goal node) is denoted as $|\pi_i|$, therefore the total number of actions in plan π_i is equal to $N_i = |\pi_i| - 1$. Agents can only traverse over edges of the graph $\mathcal{G}$ therefore $\{\pi_i(n), \pi_i(n+1)\} \in \mathcal{E}$ or alternatively an agent can wait at a node such that $\pi_i(n+1) = \pi_i(n)$. The solution to a MAPF problem is the set of k single agent plans $\mathcal{P} = \{\pi_1, \ldots, \pi_k\}$ such that the respective paths are non-conflicting. In order to conclude that the solution of a MAPF

problem is valid due to the absence of conflicts, the following conflicts are defined:

- a **node conflict**: $\pi_i(n) = \pi_j(n) \quad \forall i, j | i \neq j \in \mathcal{A}$, occurs when two agents occupy the same node in the same time step
- an **edge conflict** or **swap conflict**: $\pi_i(n) = \pi_j(n+1) \wedge \pi_i(n+1) = \pi_j(n) \quad \forall i, j | i \neq j \in \mathcal{A}$, occurs when two agents start from connected nodes and traverse over the same edge in the same time step
- a **following conflict**: $\pi_i(n+1) = \pi_j(n) \quad \forall i, j | i \neq j \in \mathcal{A}$, occurs when an agent traverses to a node that was occupied by another agent in the previous time step
- a **cycle conflict** occurs when every agent moves to a node that was occupied by another agent in the previous time step, therefore a **cycle conflict** requires a number of c of only following conflicts between a number of c agents with $c \geq 3$.

The type of MAPF problem that will be considered in this research is the widely used grid as an abstracted graph, which is also referred to as the "map". Start coordinates, end coordinates, and blocked coordinates are assigned randomly with a set of unique start locations and a set of unique goal locations. Each agent is assumed to have full static map knowledge (obstacles), while information about other agents has to be obtained through communication. Communication links between agents are established based on Euclidean norm proximity. Agents can execute one of the following five actions in each time step: move up, down, left or right or wait at current location. Apart from not using blocked coordinates, the following inter-agent conflicts will not be tolerated: node conflicts and edge conflicts. This paper considers a "one-shot" MAPF problem in which each agents has a single goal location and agents will stay at their goal location. The path length of an agent is equal to the last time step in which the goal location is reached and the number of time steps it takes for the last agent to reach its goal location is the "problem depth". A MAPF instance is considered to be "solved" if all agents have successfully reached their goal in some finite time step.

III. DECOP FRAMEWORK

The framework proposed in this paper consists of a highly scalable distributed MAPF algorithm in which agent a_i can communicate with a set of connected agents $\Gamma_i(n) = \{a_j \in \mathcal{A} \big| \, \|\pi_j(n) - \pi_i(n)\| \leq v\}$ based on proximity with Euclidean norm v. Each agent in parallel optimizes its own path using local information. An agent considers a combined cost together with the paths of selected agents in the set $\mathcal{R}_i(n) \subseteq \Gamma_i(n)$, such that the number of selected agents less than or equal to a tractable number of r agents: $|\mathcal{R}_i(n)| \leq r$. The working principle of the algorithm is inspired by a framework presented in [30] which applies cooperative local optimization for continuous linear systems.

Considering a limited number of nearby other agents yields a tractable local MAPF problem. The local MAPF problem that one particular agent solves is likely similar but not identical to the local MAPF problems of its neighbours. It is expected that agents will conclude on the same local optimal evasion manoeuvres when applying the same local optimization objective. This approach is motivated by a meta-analysis of multiple search-based centralized optimal MAPF algorithms which suggest that efficient algorithms explore different combinations of local evasion manoeuvres in order to select the optimal set of local evasion manoeuvres.

Agents traversing over a graph will likely be nearby different agents in different time steps. To account for the alteration in nearby agents, receding horizon control (RHC) is well suited. This control strategy optimizes an agent's actions over a planning horizon w and implements a number of $m \leq w$ actions, where usually $m \ll w$.

After local optimization, agents have to reach consensus with other agents in their set of connected agents $\Gamma_i(n)$ over shared variables. The shared variables in this case are the nodes and edges of the network in specific time steps. A supplementary element of this research is to investigate the type of information that is shared among agents and how information about path alternatives can be used to ameliorate parallel decision making.

Before formally introducing the DECOP algorithm, its features and characteristics will be explained according to the structure of the decision-making process as shown in Figure 1.

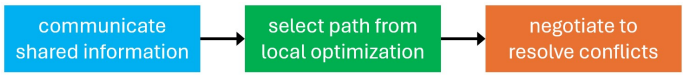


Fig. 1. DECOP decision-making process pipeline

A. Communicate shared information

A concept that was introduced (and consequently also widely accepted as a result of proven effectiveness) in the research area of decentralized MAPF algorithms, is multiple step inter-agent communication. In the case of decentralized MARL algorithms (e.g. [24], [25], [23]), it is left to a reinforcement learning mechanism to infer what information is passed between agents in either a single or multiple round communication scheme. To the authors best knowledge, this work presents the first research on the type of information that is to be shared among agents in decentralized parallel decision-making MAPF.

The selected foundation of the type of shared information is a multi-value decision diagram (MDD) for path finding that was introduced in [11]. A MDD contains the set of route alternatives that an agent can take to reach a desired node in the final (bottom) layer of the MDD, according to a specified delay. A MDD is a powerful concept because it scales linearly with the size of the map [12]. Furthermore, comparing two MDD's can also be done relatively quickly. Each agent constructs a MDD to reach its goal node and shares the first w layers with its set of connected agents. An MDD of agent a_i with a considered amount of δ delay is denoted as MDD_i^δ.

Figure 2 shows an exemplary MAPF instance where each colour represents an agent and the corresponding path alternatives with zero individual delay δ are indicated. Figure 3 shows the corresponding MDD's with $w = 4$ and $\delta = 0$,

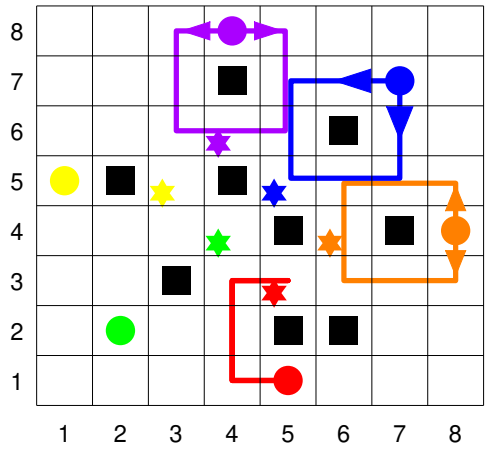


Fig. 2. Exemplary MAPF instance with $w = 4$ and $\delta = 0$ from the perspective of agent a_2(orange): agents a_1(red), a_3(blue) and a_4(purple) belong to the set of connected agents $\Gamma_2(n)$ and $\mathcal{R}_2 = \Gamma_2(n)$

agents traverse to a next layer of their MDD in each time step starting from the top layer. The coloured dashed boxes in Figure 3 indicate nodes that are also present in the same layer of the MDD of the agent with the respective colour of the box. For each shared node, agents are said to share a dependency which refers to the fact that both selecting the particular node in the particular time step as it would result in a node conflict. Equivalently edge dependencies are considered, however that type of dependency doesn't occur in this particular example.

Combining the individual MDD's of Figure 3 into feasible

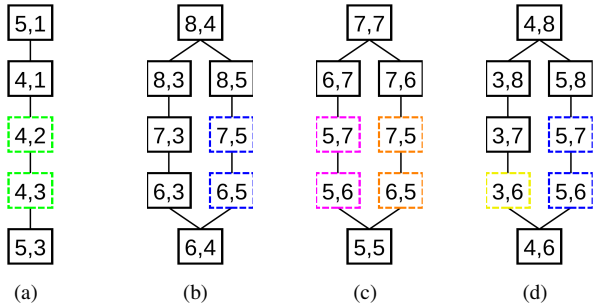


Fig. 3. Individual MDD's corresponding to the exemplary MAPF instance in Figure 2 for agents a_1**(a)**, a_2**(b)**, a_3**(c)**, a_4**(d)**

combinations results in a joint solution set called the MDD product, as shown in Figure 4. In this new type of data structure, the colour of the boxes represents the respective agent involved in the MDD product. Additionally, the dash type of a box represents a specific type of dependency (with an unspecified agent). The dash type distinguishes types of dependencies with respect to the set of agents connected ($\mathcal{R}_i$) to the "searching agent" which is agent a_2 in this case. An external dependency between agents (a_j, a_p) indicates a dependency of any agent $a_j \in \Gamma_i(n)$ with another agent $a_p \notin \Gamma_i(n)$. Similarly, for an internal dependency between (a_j, a_p), both agents $a_j, a_p \in \Gamma_i(n)$ belong to the set of selected agents. Finally, an internal dependency with the searching agent is also indicated.

B. Path selection from local optimization

The input of an agent's local MAPF optimization problem are the MDD's that were received from selected agents ($a_j \in \mathcal{R}_i$). The MDD's are appended with additional node and edge properties as shown in Figure 3. Subsequently, each agent in parallel solves a local MAPF problem by performing a graph search on the local MDD product (set of feasible joint solutions), as shown in Figure 4, according to a common objective.

Two types of global objectives were introduced in MAPF literature, namely the sum of cost (SOC) objective and the makespan objective. The SOC objective refers to the sum of all path lengths while the makespan objective considers only the length of the longest path (which is equal to the problem depth). These objectives refer to the global state of the system whereas for RHC, agents are likely not to reach their goal within the planning horizon. The sum of intermediate delays is therefore introduced as an objective to comply with the windowed planning mechanism. Intermediate delay is defined as the number of additional time steps that an agent has to travel within the planning horizon in a single planning round, compared to its shortest path to goal. Contrary to the generic MAPF objectives, the intermediate delay objective is more focused on individual agent objectives as it can take into account the minimum amount of time steps required to reach an agent's goal. With respect to the generic MAPF objectives, one can consider the minimum maximum intermediate delay as the confined interpretation of the makespan objective and the minimum total intermediate delay as the confined interpretation of the SOC objective. The confined interpretation of the makespan objective is less suitable for windowed MAPF planning, for example in a scenario where at least one agent has a delay equal to the full planning horizon as different solutions with respect to other agents cannot be distinguished using this objective.

The MDD product is explored through a graph search and the cost of a new node in the MDD product is equal to the cost of the originating node together with the sum of all agent's intermediate delay increase:

$$c(node_{new}) = c(node_{old}) + \sum_{n=1}^{|\mathcal{R}_i|} \Delta\delta_n \tag{1}$$

When applying the node cost function specified in Equation 1 to the exemplary MDD product in Figure 4, all nodes will be of cost zero since the considered delay in the individual MDD's was zero. To be able to differentiate between solutions in these special scenarios, an additional cost can be considered for nodes with external and internal dependencies resulting in a weighted node cost function:

$$\begin{aligned} c(node_{new}) =& c(node_{old}) + w_1 \sum_{n=1}^{|\mathcal{R}_i|} \Delta\delta_n + w_2 \sum_{n=1}^{|\mathcal{R}_i|} int.dep.(a_n) \\ &+ w_3 \sum_{n=1}^{|\mathcal{R}_i|-1} ext.dep.(a_n) \end{aligned} \tag{2}$$

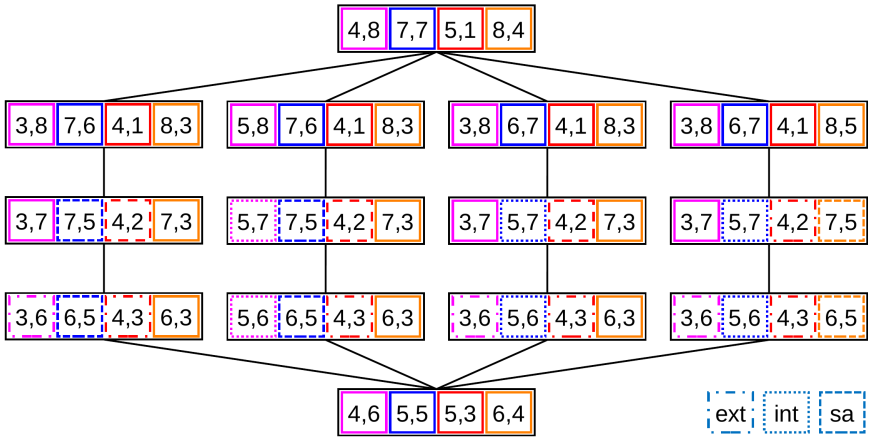


Fig. 4. Feasible combinations of individual agent MDD's $\in \mathcal{R}_2$ with $w = 4$ and $\delta = 0$, resulting in a joint solution set called the MDD product. The dash type distinguishes types of dependencies with respect to the set ($\mathcal{R}_2$) of agents connected to the searching agent (a_2)

Note that the number of agents over which the external dependencies are summed in Equation 2 is $|\mathcal{R}_i| - 1$ since the searching agent cannot have external dependencies.

A consequence of considering a planning horizon that is shorter than the (minimum) problem depth in RHC are implications as a result of the "horizon effect". The horizon effect concerns information about changes in system state or acting disturbance(s) that aren't considered in planning as they lie outside the planning horizon. To capture the effect of states and disturbances on the trajectory cost that will occur after the planning horizon, RHC and model predictive control (MPC) usually add a terminal cost that is only dependent on the final state.

The main implication of horizon effects in windowed MAPF is the fact that a local optimum solution with respect to the sum of intermediate delays might be a deadlock configuration. Planning over a horizon of sufficient length such that all considered agents can reach their goal location can never result in a deadlock as the corresponding cost would be infinite (agents will not reach their goal). In windowed planning the cost corresponding to a deadlock can be finite as the cost evaluated is the incurred intermediate delay over a finite planning horizon. A deadlock will occur in windowed planning if the joint delay cost of avoiding a deadlock is higher compared to the cost of being stuck (equally) in the deadlock. Figure 5 shows an example of a windowed MAPF optimization problem that will result in a deadlock. The cost function applied in Figure 5(a) is the baseline cost function c_0 that doesn't consider internal and external dependencies. The weights of the node cost function in Equation 2 are therefore set as $c_0(1, 0, 0) = c(w_1, w_2, w_3)$. Since the orange and blue agents both have to incur intermediate delay to let the red agent pass, the combined cost is lower when incurring delay in the deadlock configuration shown in Figure 5(a). Figure 5(b) shows the MAPF optimization result when applying the terminal cost function $c_T(node)$ which assigns a penalty of value F to a node in the final layer of the MDD product if there is no agent decreasing its distance to goal in the final time step:

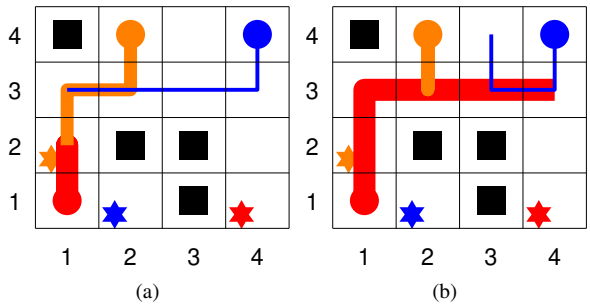


Fig. 5. Result of local MAPF optimization without **(a)** and with **(b)** terminal cost function c_T for $w = 5$, $T = 3$

$$\eta(node) = \forall a_j \in \mathcal{R}_i \mid \pi_j(w) \neq g_j$$
$$\theta(node) = \forall a_j \in \eta \mid \text{decrease count}(a_j) \geq T$$
$$c_T(node) = \begin{cases} F, & \text{if } \eta \neq \emptyset \wedge \theta = \emptyset \\ 0, & \text{if } \eta = \emptyset \vee \theta \neq \emptyset \end{cases} \tag{3}$$

The set η contains the agents considered in the local MAPF optimization problem that are not at goal and its subset θ contains only agents that are considered to be "decreasing". A threshold value T is introduced which specified the number of consecutive time steps an agent has to decrease its distance to goal to be considered as a decreasing agent.

Introducing the threshold T avoids agents taking one step back and decrease their distance to goal only in the final time step to avoid the deadlock penalty. Inevitably, in some cases a "truly" decreasing agent might have less consecutive decreasing steps than the threshold, T is therefore left as a hyperparameter to be obtained through tuning. Additionally, cases in which agents deliberately have to take detours will also be wrongly penalized by the penalty function c_T which underlines the challenge of inferring a deadlock scenario in a decentralized manner.

Table I shows the results of applying a RHC policy with implementation window length $m = 1$ and considers applying the baseline cost function c_0 compared to applying c_0

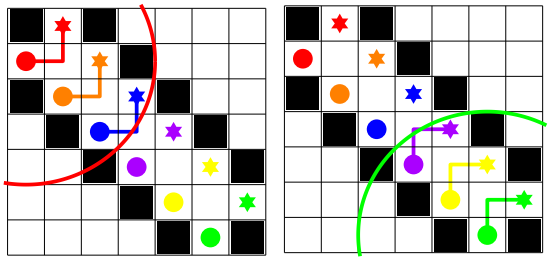
TABLE I
NUMBER OF AGENTS THAT REACH THEIR GOAL AS A RESULT OF RHC IMPLEMENTATION WITH $m = 1$ FOR THE MAPF PROBLEM INSTANCE OF FIGURE 5 FOR DIFFERENT PLANNING HORIZON LENGTHS w

w	9	8	7	6	5	4	3
c_0	3	3	2	1	1	1	0
$c_0 + c_T$	3	3	3	3	3	2	0
T	3	3	3	3	3	2	1

combined with the terminal cost penalty function c_T. The minimum problem depth in this scenario is 9 and evidently all agents can reach their goal for a planning horizon $w = 9$ when applying c_0. For a shorter planning horizon of $w = 7$, the red and orange agent will reach their goal and for an even shorter planning horizon only the orange agent or no agent will reach its goal. Supplementing the cost function c_0 with penalty function c_T allows all agents to reach their goal for a planning horizon as short as $w = 5$, where the decreasing threshold values T are tuned accordingly.

C. Path negotiation

In a decentralized setting, agents inherently have access to limited information about the global problem when solving a local optimization problem in parallel. Consequently, the desired paths obtained from solving local MAPF optimization problems might render infeasible from a global perspective. An illustrative example is shown in Figure 6 with communication range $v = 3.5$ and $r = 6$. The communication range for both the red and green agent are indicated with large circles of their respective colour. Local optimization results with a resulting delay of zero are shown from the perspectives of the red agent Figure 6(a) and the green agent Figure 6(b). Although the desired paths are locally feasible, one can easily see that the desired paths for the blue and purple agents from the perspective of respectively the red and green agent, are incompatible. Moreover, for this exemplary MAPF problem there exists no solution where all agents can reach their goal location with zero delay. However, all agents will select a desired path with zero delay because they can all find a feasible local solution with zero delay. To



(a) (b)

Fig. 6. Result of local MAPF optimization without **(a)** and with **(b)** terminal cost function c_T for $w = 5$, $T = 3$

resolve any remaining conflicts a random prioritized planning scheme is therefore introduced in which each agents draws a random priority value from a uniform distribution and the conflict over shared variables is awarded to the agent with the highest priority. Note that such a scheme, which is analogous to the HCA* MAPF algorithm [19], is not complete for MAPF problems that are not well-formed. A number of MAPF algorithms introduce schemes that employ dynamic priorities using communication to overcome this problem such as in the DMAPP algorithm [21] or the PIBT algorithm [27]. In order to keep the communication effort tractable, a fixed priority scheme combined with a conservative baseline solution will be implemented. The baseline solution of an agent entails a number of m wait actions such that $\pi_i(n) = \pi_i(n+1) = \ldots = \pi_i(n+m)$ and represents a solution that is always feasible. To ensure that the baseline solution is always feasible, each agent is deterred from selecting any other agent's baseline solution as a desired path within its planning horizon. Filtering other agent's baseline solutions introduces the restriction of follower conflicts during the first m steps, making this scheme more conservative. Since the "fixed priority - baseline feasible solution" mechanism is only activated in special scenarios such as described in Figure 6, it is expected that the combined approach of local optimization and receding horizon implementation will compensate for the drawbacks of a fixed priority mechanism.
The described decentralized MAPF framework is presented in

Result: parallel conflict free action selection
1 **Input** ¡$\mathcal{A}, s, g$
2 initialize $t_{sim} = 0$
3 Each agent in parallel, **do**
4 draw ε_i from a uniform distribution $(0, 1)$
5 $\pi_i(0) \leftarrow s_i$
6 **while** $t_{sim} < t_{final}$ **do**
7 construct Γ_i by Euclidean distance norm v
8 construct MDD$_i^\delta$, $\delta = w$
9 filter baseline solutions of $a_j \in \Gamma_i$ from MDD$_i^\delta$
10 select first w levels of MDD$_i^\delta$ and share with Γ_i
11 infer dependencies and append to MDD$_i^\delta$
12 share new MDD with Γ_i to infer external dependencies
13 sort Γ_i according to Manhattan distance
14 **while** *solution* $_i$ = INVALID **do**
15 collect agents with valid solutions: $\Lambda_i \subseteq \Gamma_i$
16 filter valid solutions of $a_j \in \Lambda_i$ from MDD$_i^\delta$
17 search MDD product to select desired path
18 communicate selected path to Γ_i
19 **if** *no conflicts with received paths of higher priority agents* **then**
20 *solution* $_i$ = VALID
21 **end**
22 **end**
23 $\pi_i(t_{sim} + 1, \ldots, t_{sim} + m) \leftarrow$ action$_i$
24 $t_{sim} = t_{sim} + m$
25 **end**

Algorithm 1: DECOP: receding horizon path negotiation algorithm with conservative baseline solution

pseudo-code in Algorithm 1. In each planning round, lines 6-

25 are executed where line 9 represents realising the baseline solution scheme. The allocation of shared variables according to the random priority scheme in lines 19-20 ensures that each path securing iteration (lines 14-22) at least one of the agents in a set of connected agents will be able to secure a path. Note that if induced cooperation is achieved successfully through the common local path optimization policy of line 17, all agents can simultaneously secure a path in the first iteration. Algorithm 2 shows the decentralized common path selection policy which is used in line 17 of Algorithm 1. The local MAPF problem is no longer considered tractable if a maximum number of P nodes has been expanded without obtaining a final solution. Consequently, a greedy path search is performed in line 7 of Algorithm 2 which doesn't consider any other agents. In order to potentially increase the size of the set $\mathcal{R}_i$, agents that are already at goal are initially added to the set of ignored agents Σ_i in line 2 of Algorithm 2 as it can be assumed that these agents will stay at their goal location. If it turns out that after local optimization a path of any agent $a_i, a_j \in \mathcal{R}_i$ crosses the goal location of an ignored agent, this ignored agent is removed from the set Σ_i and the path selection process is repeated.

Result: desired path of agent a_i with length w
1 **Input** Γ_i, $\{\text{MDD}_j^{\delta} \ \forall a_j \in \Gamma_i\}$
2 initialize the set of ignored agents $\Sigma_i \subseteq \Gamma_i$ with agents that were at goal in $t_{sim} - 1$
3 **while** *int.solution* $_i$ = INVALID **do**
4 construct $\mathcal{R}_i = \Gamma_i \setminus \Sigma_i$, such that $|\mathcal{R}_i| \leq r$
5 select path as a result of optimal MDD product graph search according to node cost function c (Eq. 2) with terminal cost c_T (Eq. 1)
6 **if** *# expanded nodes has exceeded* P **then**
7 choose greedy path
8 *int.solution* $_i$ = VALID
9 **else if** *any of the paths of agents in* $\mathcal{R}_i$ *contains a node of an agent in* Σ_i **then**
10 remove this agent from Σ_i
11 **else**
12 *int.solution* $_i$ = VALID
13 **end**
14 **end**

Algorithm 2: Decentralized path selection policy

IV. Empirical evaluation

After describing the experiment setup and selection of hyperparameters, the performance evaluation is presented. The designed algorithm is named the decentralized optimization (DECOP) algorithm for an easier reference. The empirical performance evaluation based on simulation results consists of three parts. Firstly, DECOP is compared to a number of state-of-the-art decentralized MAPF algorithms. Secondly, DECOP is compared to one specific fast decentralized MAPF algorithm, namely the PIBT algorithm [27], with an in-depth analysis on individual agent performance.

A. Experimental setup and hyperparameter selection

The following properties will be specified for the considered grid map MAPF problem instances: (square) map size, density of (random) obstacles, set of unique start locations, set of unique goal locations. A MAPF instance type specifies these properties and consequently one hundred instances are generated. A MAPF instance is considered to be solved if a set of conflict free paths has been obtained such that all agents reach their goal location.

The experiments are run on a server with an Intel Xeon E5-2637 (v3) CPU (4 cores) and although the DECOP algorithm is a fundamentally decentralized algorithm that can be executed in parallel by multiple CPU, in this simulation setup the path securing iteration is preformed in a sequential manner by a single CPU. The hyperparameters shown in Table II are selected from the performance results for MAPF instance type: 20 (square) map size, 16 agents (4% agent density), 0%, 15% and 30% obstacle density. The hyperparameter combinations of $r = 4, 5$, $w = 8, 12$ and $T = 2, 3, 4, 6$ were evaluated for this MAPF instance type (reported in Appendix section A). Whereas performance with respect to the delay KPI's was similar for all considered combinations, the selected combination of hyperparameters exhibited a better combined performance on success rate and times the node expansion limit was reached.

To investigate the consideration of internal and external dependencies, three different variants of the DECOP algorithm will be evaluated with node cost functions: $c_0 = c(1, 0, 0) + c_T$, $c_1 = c\,(3 \cdot w \cdot (|\mathcal{R}_i| - 1), 0, 1) + c_T$, $c_2 = c(3 \cdot w \cdot |\mathcal{R}_i|, 1, 0) + c_T$. Where c is the cost function presented in Equation 2 and c_T is the terminal cost function in Equation 3. The weights of cost functions c_1 and c_2 are set such that the internal and external dependencies are only used to differentiate between equal cost solutions. A single step of delay is equal to the maximum number of dependencies (three types times the number of agents) over the full planning horizon w. Note that c_1 uses $|\mathcal{R}_i| - 1$ as the number of agents since the searching agent cannot have external dependencies.

TABLE II
Overview of hyperparameters used by the DECOP algorithm

parameter	description	value
w	length of the planning horizon	12
δ	maximum delay included in all MDD	w
m	length of the implementation horizon	1
T	agent decreasing threshold	3
P	maximum number of nodes to expand	20000
r	desired size of the set of selected agents ($\lvert\mathcal{R}_i\rvert$)	4
F	terminal cost penalty value	100
v	Euclidean norm for communication range	5

B. Performance comparison

The DECOP algorithm will be compared to the state-of-the-art decentralized RL algorithm SCRIMP [25] and the state-of-the-art decentralized fast algorithm: PIBT [27]. In SCRIMP a policy is trained in a MARL setting, including a two step learned communication with agents that are within a range of 5. The PIBT algorithm applies dynamic priorities and

induced communication and it is proven to be complete for open maps (no obstacles). The two used KPI's by SCRIMP are the problem depth, which is the number of time steps it takes to attain a global solution for a MAPF instance, and the success rate of successfully solved MAPF instances. The planning results of SCRIMP and OdrM* reported in Table III were not run but instead taken directly from tables II and IV in [25]. OdrM* is a suboptimal centralized algorithm with an inflation factor $\varepsilon = 2.0$ and a runtime limit of five minutes. The PIBT and DECOP algorithms were both run on the same dataset of 100 MAPF instances and the DECOP_2 variant represents the best performing variant of DECOP with node cost function c_2. Note that both the DECOP and PIBT algorithms are guaranteed to be conflict-free while the SCRIMP algorithm usually contains a (small) number of conflicts in its solutions. Overall, the DECOP algorithm

TABLE III
PERFORMANCE COMPARISON WITH OTHER DECENTRALIZED (RL) MAPF ALGORITHMS FOR DIFFERENT MAPF INSTANCE TYPES. A VALUE BETWEEN BRACKETS (·) DENOTES THE STANDARD DEVIATION TO THE CORRESPONDING MEAN VALUE.

method	**average problem depth**			**success rate**		
8 agents in 10×10 world with $0\%, 15\%, 30\%$ obst. rate (columns)						
ODrM*	12.6(2.1)	13.7(2.5)	16.2(4.0)	100%	100%	100%
SCRIMP	14.4(2.6)	18.3(18.8)	21.0(10.2)	100%	100%	97%
PIBT	13.5(2.6)	14.8(3.0)	17.5(8.7)	100%	99%	86%
DECOP_2	12.3(2.4)	13.5(2.7)	15.5(4.6)	100%	99%	91%
16 agents in 20×20 world with l0%, 15%, 30% obst. rate (columns)						
ODrM*	27.4(3.6)	27.6(3.7)	34.6(6.6)	100%	100%	95%
SCRIMP	26.5(4.1)	28.46(3.4)	41.3(20.1)	100%	100%	97%
PIBT	27.8(4.3)	27.8(4.2)	35.2(6.9)	100%	98%	92%
DECOP_2	26.8(4.0)	27.4(4.0)	36.3(8.2)	100%	99%	97%
32 agents in 30×30 world with l0%, 15%, 30% obst. rate (columns)						
ODrM*	43.0(4.6)	43.4(5.0)	54.0(10.4)	100%	100%	97%
SCRIMP	43.3(5.0)	46.5(7.0)	69.1(31.6)	100%	100%	97%
PIBT	44.2(4.8)	43.7(5.2)	52.4(7.4)	100%	100%	84%
DECOP_2	43.3(4.7)	43.4(4.8)	55.6(10.0)	100%	98%	88%

shows very similar performance to the suboptimal centralized (ODrM*) reference algorithm. In all cases, except for 30% 16 and 32 agents, the DECOP algorithm reports lower average problem depths compared to PIBT and in all cases the DECOP has a lower average problem depth than SCRIMP. For the 30% obstacle rate on all MAPF instance sizes, both PIBT and the DECOP algorithm do report a lower success rate compared to SCRIMP.

Two additional KPI's are introduced to accommodate the (more individualistic) objective function of minimum intermediate delay, namely the total delay δ and the relative delay δ/depth where "depth" is the shortest path with zero delay. The relative delay is limited to 100%. Table IV shows the performance results according to the new KPI's for the different variants of the DECOP algorithm and the PIBT algorithm. Instead of taking an average over the MAPF instances that an algorithm solved successfully (as was done in Table III), the values reported in Table IV are averaged only with respect to the set of MAPF instances that all considered algorithms solved successfully to get a more representative performance comparison. Additionally, Table V shows the success rates of the PIBT algorithm and considered DECOP variants for the same 100 MAPF instances for each MAPF instance type.

To represent the extent to which cooperation was successfully induced, the EX and CP values are reported in Table IV. The "expansion limit" (EX) indicates the number of times that the node expansion limit P was reached (in line 17) by at least one agent in each first path-securing iteration compared to the total number of planning rounds. The "compliance" (CP) denotes the number of times all agents were simultaneously able to selected a complying solution during the first path securing iteration compared to the total number of planning rounds. Evidently, for the 30 by 30 MAPF instance types with a

TABLE IV
PERFORMANCE RESULTS FOR TWO DIFFERENT MAPF INSTANCE TYPES WITH 0%, 15% AND 30% OBSTACLE RATES OVER A SET OF COMMONLY SOLVED MAPS. A VALUE BETWEEN BRACKETS (·) DENOTES THE STANDARD DEVIATION TO THE CORRESPONDING MEAN VALUE.

obst rate	com. maps	method	δ	$\frac{\delta}{\text{depth}}$	EX	CP	average runtime
32 agents on 30×30 map - 3.6% agent density							
0%	99	PIBT	2.3(5.3)	14%(30)	-	-	instant
		DECOP_0	0.25(0.65)	1.4%(4.1)	13%	37%	83m(52)
		DECOP_1	0.23(0.73)	1.3%(4.5)	2.8%	35%	42m(21)
		DECOP_2	0.21(0.59)	1.1%(3.5)	2.0%	45%	34m(16)
15%	93	PIBT	2.8(5.8)	17%(31)	-	-	instant
		DECOP_0	0.78(2.1)	4.0%(11)	0.20%	34%	12m(7.5)
		DECOP_1	0.77(2.0)	3.9%(10)	0.20%	33%	10m(5.7)
		DECOP_2	0.69(1.8)	3.5%(9.4)	0.30%	39%	8.7m(4.4)
30%	49	PIBT	2.9(6.7)	13%(28)	-	-	instant
		DECOP_0	2.1(5.5)	8.4%(20)	1.85%	29%	9.8m(25)
		DECOP_1	2.1(5.1)	8.4%(20)	1.57%	29%	8.8m(17)
		DECOP_2	2.0(5.0)	8.2%(20)	1.56%	32%	8.4m(16)
20 agents on 10×10 map - 20% agent density							
0%	62	PIBT	0.93(2.1)	18%(36)	-	-	instant
		DECOP_0	0.46(1.2)	9.9%(24)	0.00%	37%	0.9m(0.9)
		DECOP_1	0.49(1.3)	10%(24)	0.00%	32%	1.8m(1.8)
		DECOP_2	0.52(1.4)	11%(25)	0.00%	31%	1.0m(1.0)
15%	43	PIBT	1.2(2.8)	16%(35)	-	-	instant
		DECOP_0	1.1(3.0)	14%(30)	2.10%	27%	4.2m(11)
		DECOP_1	1.1(3.0)	14%(31)	2.23%	23%	4.1m(8.6)
		DECOP_2	1.1(3.2)	15%(31)	1.94%	23%	3.6m(9.4)
30%	3	PIBT	0.17(1.4)	1.6%(12)	-	-	instant
		DECOP_0	0.17(1.5)	1.5%(11)	3.23%	21%	0.4m(2.6)
		DECOP_1	0.14(1.2)	1.6%(12)	5.08%	17%	0.8m(4.8)
		DECOP_2	0.20(1.8)	1.8%(13)	7.59%	15%	1.1m(6.6)

TABLE V
SUCCESS RATE COMPARISON BETWEEN DIFFERENT VARIANTS OF DECOP AND THE PIBT ALGORITHM FOR BOTH A LOWER AND A HIGHER AGENT DENSITY MAPF INSTANCE TYPE

method	**32 agents on 30×30**			**20 agents on 10×10**		
obstacle rate	0%	15%	30%	0%	15%	30%
PIBT	100%	100%	84%	100%	88%	44%
DECOP_0	100%	99%	66%	74%	71%	17%
DECOP_1	99%	95%	79%	87%	74%	22%
DECOP_2	100%	98%	88%	86%	72%	29%

relatively low agent density, the compliance values are consistently higher for the DECOP_2 variant which uses the amount of internal dependencies to distinguish equal cost solutions. Interestingly, for the DECOP_2 variant on the MAPF instance type of 32 agents on a 30 by 30 open map, the node expansion limit is reached significantly fewer times by any agent in the first path-securing iteration compared to the other variants which likely also explains the lower computation time. For the MAPF instance with 20 agents on a 10 by 10 map, the patterns are less clear as the compliance value of the vanilla DECOP_0 variant is higher compared to the other variants. In all cases, the DECOP_0 variant reports a lower success rate compared to the extended variants. Note that the MAPF instance type with 20 agents on a 10 by 10 map with a 30% obstacle density shows KPI values over just 3 commonly solved instances, therefore only the success rate KPI is of statistical significance for this MAPF instance type.

V. Discussion

It can be concluded from the presented results that the proposed DECOP algorithm yields very effective solutions that are likely close to the global SOC optimal solution. For most considered MAPF instance types, the DECOP algorithm reports a lower average problem depth compared to RL algorithms while it is guaranteed to be conflict free and requires no training.

Table VI shows a comparison of algorithm features where "++" denotes good performance, "+" a slight advantage and "-" bad performance or not favourable for scalability. Since SCRIMP and PIBT apply a decentralized policy, both algorithms exhibit good performance with respect to the computational effort. The OrdM* algorithm requires centralized communication (and computation) and the PIBT algorithm demands induced sequential communication, both resulting in relatively bad performance with respect to the communicational effort. The DECOP algorithm requires no training and reports a high (individual agent) efficiency. The communicational effort in DECOP can be multi-step but the number of communication rounds is bounded by the (maximum) number of connected agents. Besides being much more efficient compared to the PIBT and SCRIMP algorithms, the DECOP algorithm can better cope with complex corridor scenarios using a dedicated deadlock detection mechanism. Although a higher computational effort is demanded by DECOP, this effort is distributed over all agents in the system.

TABLE VI
Algorithm feature comparison

method	training time	efficiency	communication	computation
ODRM*	++	++	-	-
SCRIMP	-	+	++	++
PIBT	++	-	-	++
DECOP	++	++	+	+

From the in-depth results shown in Table IV, it can be concluded that all variants of the DECOP algorithm consistently provide sets of paths that contain a lower combined delay over all agents compared to the PIBT algorithm. Of the three variants, the DECOP_2 variant, which considers the total amount of internal dependencies for equally delayed solutions, achieves a higher success rate (Table V) and compliance value (Table IV) for the lower agent density MAPF types that were considered. For the higher agent density MAPF type (20 agents on 10 by 10 map), no clear conclusion can be drawn with respect to the induced cooperation based on the compliance values and success rates.

Similarly to the RL algorithms, the most challenging MAPF instances (and therefore unsolved instances) are instances with a higher obstacle density. This is likely due to the fact that these MAPF problems are usually not well-formed or contain corridors through which many agents have to pass (simultaneously). Although MAPF instances that require such complex corridor coordination can be solved by increasing r, w and u, this approach doesn't scale well as the number of agents to consider will eventually render intractable.

Many of the MAPF instances that were not solved by the DECOP algorithm failed as a result of agents choosing different solutions from a set of equal cost solutions. This shortcoming might be resolved by considering additional properties during path selection such as "always pass on the right".

VI. Conclusion and future work

This research introduced a novel approach for solving the MAPF problem in a decentralized manner with high scalability. In the proposed DECOP algorithm, agents achieve induced cooperation through solving a tractable local MAPF problem. Local optimal solutions are obtained in parallel and a communication protocol is executed to achieve global conflict-free operation using only local information.

The reported results suggest that the proposed approach could be very effective as it requires no training and the hyper-parameters can be tuned with respect to a particular MAPF problem type. To further investigate the characteristics of the parallel decentralized decision-making mechanism, many research directions can be considered, for example:

- the implementation of a more computationally efficient decoupled MAPF optimization algorithm instead of the used graph search coupled ICTS algorithm.
- further investigation into the compliance of the first path securing iteration in each planning round. Instruments to consider are for example the weights of different dependencies in the cost function and the type of dependencies to consider.
- investigation of heuristic search that for example favours having fewer dependencies over delay, this mechanism could potentially improve induced cooperation.
- design a policy to overcome complex corridor scenarios, or deadlocks in general, in a decentralized manner
- introduce priority values in the local optimization problem that are dependent on the amount of incurred delay such that the delay can be balanced in an online manner.

REFERENCES

[1] A. Dabiri, A. Hegyi, and S. Hoogendoorn, "Optimized Speed Trajectories for Cyclists, Based on Personal Preferences and Traffic Light Information-A Stochastic Dynamic Programming Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 777–793, Feb. 2022.

[2] S. Aoki and R. Rajkumar, "Safe Intersection Management With Cooperative Perception for Mixed Traffic of Human-Driven and Autonomous Vehicles," *IEEE Open Journal of Vehicular Technology*, vol. 3, pp. 251–265, 2022.

[3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–9, Mar. 2008, number: 1. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2082

[4] X. Yang, R. Feng, P. Xu, X. Wang, and M. Qi, "Internet-of-Things-augmented dynamic route planning approach to the airport baggage handling system," *Computers & Industrial Engineering*, vol. 175, p. 108802, Jan. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835222007902

[5] J. Ding, K. Chakrabarty, and R. Fair, "Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 12, pp. 1463–1468, Dec. 2001.

[6] D. M. Kornhauser, G. Miller, and P. Spirakis, *Coordinating pebble motion on graphs, the diameter of permutation groups and applications*, ser. Robotics report. New York: Courant Institute of Mathematical Sciences, New York University, May 1984, oCLC: 1042475119. [Online]. Available: https://archive.org/details/coordinatingpebb00korn

[7] G. Goraly and R. Hassin, "Multi-Color Pebble Motion on Graphs," *Algorithmica*, vol. 58, no. 3, pp. 610–636, Nov. 2010. [Online]. Available: https://doi.org/10.1007/s00453-009-9290-7

[8] J. Yu and S. LaValle, "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, no. 1, pp. 1443–1449, Jun. 2013, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/8541

[9] R. Stern, "Multi-Agent Path Finding – An Overview," in *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*, ser. Lecture Notes in Computer Science, G. S. Osipov, A. I. Panov, and K. S. Yakovlev, Eds. Cham: Springer International Publishing, 2019, pp. 96–115. [Online]. Available: https://doi.org/10.1007/978-3-030-33274-7_6

[10] 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, Feb. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001386

[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, Feb. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370212001543

[12] F. Aljalaud and N. Sturtevant, "Finding Bounded Suboptimal Multi-Agent Path Planning Solutions Using Increasing Cost Tree Search (Extended Abstract)," *Proceedings of the International Symposium on Combinatorial Search*, vol. 4, no. 1, pp. 203–204, 2013, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18303

[13] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem," *Proceedings of the International Symposium on Combinatorial Search*, vol. 5, no. 1, pp. 19–27, 2014, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18315

[14] L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding," *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, Jul. 2016.

[15] J. Li, W. Ruml, and S. Koenig, "EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, pp. 12 353–12 362, May 2021, number: 14. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17466

[16] J. Yu, "Diameters of Permutation Groups on Graphs and Linear Time Feasibility Test of Pebble Motion Problems," Oct. 2012, arXiv:1205.5263 [cs]. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5210f86a914f5db8674feec6a9a2fe7bc1268d01

[17] J. Yu and D. Rus, "Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms," Jul. 2014, arXiv:1205.5263 [cs]. [Online]. Available: http://arxiv.org/abs/1205.5263

[18] B. de Wilde, A. W. ter Mors, and C. Witteveen, "Push and rotate: cooperative multi-agent path planning," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, ser. AAMAS '13. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 87–94.

[19] D. Silver, "Cooperative Pathfinding," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, no. 1, pp. 117–122, 2005, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/18726

[20] P. Surynek, "Compilation-based Solvers for Multi-Agent Path Finding: a Survey, Discussion, and Future Opportunities," Apr. 2021, arXiv:2104.11809 [cs]. [Online]. Available: http://arxiv.org/abs/2104.11809

[21] S. S. Chouhan and R. Niyogi, "DMAPP: A Distributed Multi-agent Path Planning Algorithm," in *AI 2015: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, B. Pfahringer and J. Renz, Eds. Cham: Springer International Publishing, 2015, pp. 123–135.

[22] Y. Zhang, K. Kim, and G. Fainekos, "DisCoF: Cooperative Pathfinding in Distributed Systems with Limited Sensing and Communication Range," in *Distributed Autonomous Robotic Systems*, ser. Springer Tracts in Advanced Robotics, N.-Y. Chong and Y.-J. Cho, Eds. Tokyo: Springer Japan, 2016, pp. 325–340.

[23] Z. Ma, Y. Luo, and H. Ma, "Distributed Heuristic Multi-Agent Path Finding with Communication," Jun. 2021, arXiv:2106.11365 [cs]. [Online]. Available: http://arxiv.org/abs/2106.11365

[24] 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)*, May 2022, pp. 10 695–10 701.

[25] Y. Wang, B. Xiang, S. Huang, and G. Sartoretti, "SCRIMP: Scalable Communication for Reinforcement- and Imitation-Learning-Based Multi-Agent Pathfinding," Mar. 2023, arXiv:2303.00605 [cs]. [Online]. Available: http://arxiv.org/abs/2303.00605

[26] J. Chung, J. Fayyad, Y. A. Younes, and H. Najjaran, "Learning team-based navigation: a review of deep reinforcement learning techniques for multi-agent pathfinding," *Artificial Intelligence Review*, vol. 57, no. 2, p. 41, Feb. 2024. [Online]. Available: https://doi.org/10.1007/s10462-023-10670-6

[27] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, Sep. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370222000923

[28] K. Okumura, Y. Tamura, and X. Défago, "winPIBT: Extended Prioritized Algorithm for Iterative Multi-agent Path Finding," Dec. 2020, arXiv:1905.10149 [cs]. [Online]. Available: http://arxiv.org/abs/1905.10149

[29] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks," Jun. 2019, arXiv:1906.08291 [cs]. [Online]. Available: http://arxiv.org/abs/1906.08291

[30] T. Keviczky, F. Borrelli, and G. J. Balas, "Decentralized receding horizon control for large scale dynamically decoupled systems," *Automatica*, vol. 42, no. 12, pp. 2105–2115, Dec. 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109806003049

APPENDIX A
HYPERPARAMETER TESTING RESULTS

The "δ" KPI represents the delay of an agent as a result of the obtained solution compared to its path length when considering no other agents (minimum path length). Correspondingly, the "$\delta/_{depth}$" KPI denotes this delay relative to the minimum path length. The relative delay is limited to 100%. The "EX" KPI indicates the number of times that the node expansion limit was reached by at least one agent in each first path securing iteration compared to the total number of planning rounds. Similarly the "EX_2" KPI shows the total number of times that the node expansion limit was reached in an agent's first path securing iteration compared to

the total number of first path securing iterations (# planning rounds $\times$ # agents). Reaching the node expansion limit forces an agent to choose a greedy path instead of applying the common decentralized policy. Table VII shows a Monte-Carlo simulation for different combinations of hyperparameters w and T for an selected agent set size $r = 4$. The node expansion limit is $P = 20000$, the communication range is $v = 5$ and the length of the implementation horizon is $m = 1$.
Similarly Table VIII shows a Monte-Carlo simulation for a selected agent set size $r = 5$.

TABLE VII
MAPF PLANNING RESULTS ON A 20X20 MAP WITH 16 AGENTS WITH DIFFERENT OBSTACLE RATES. DIFFERENT VALUES OF HYPERPARAMETERS w (PLANNING HORIZON) AND T (DECREASING THRESHOLD) ARE GIVEN FOR $r = 4$. THE ACTION IMPLEMENTATION LENGTH m IS EQUAL TO 1. A VALUE IN BRACKETS $(\cdot)$ DENOTES A STANDARD DEVIATION FROM THE CORRESPONDING MEAN VALUE

type	w	T	δ	$\frac{\delta}{\text{depth}}$	EX	EX_2	SR	average runtime
0%/100	8	2	0.19(0.60)	1.7%(6.3)	0.00%	0.00%	100%	3.4m(2.0)
	8	4	0.19(0.60)	1.7%(6.3)	0.00%	0.00%	100%	3.3m(1.9)
	8	6	0.20(0.65)	1.7%(6.5)	0.07%	0.00%	100%	3.6m(2.1)
	12	3	0.19(0.59)	1.6%(6.2)	0.71%	0.09%	100%	9.3m(6.7)
	12	4	0.19(0.59)	1.6%(6.2)	0.78%	0.09%	100%	9.1m(6.6)
	12	6	0.19(0.58)	1.6%(6.2)	0.78%	0.09%	100%	9.4m(6.8)
15%/91	8	2	0.54(1.5)	4.1%(12)	0.00%	0.00%	93%	1.2m(1.2)
	8	4	0.59(1.7)	4.5%(13)	0.04%	0.00%	92%	1.3m(1.1)
	8	6	0.62(1.7)	4.9%(14)	0.40%	0.04%	94%	1.5m(1.4)
	12	3	0.55(1.5)	4.2%(12)	0.12%	0.01%	92%	2.6m(2.4)
	12	4	0.55(1.5)	4.2%(12)	0.24%	0.02%	92%	2.5m(2.5)
	12	6	0.56(1.6)	4.2%(12)	0.28%	0.03%	94%	2.7m(2.7)
30%/82	8	2	2.1(4.1)	12%(23)	0.20%	0.02%	86%	1.7m(2.6)
	8	4	2.2(4.3)	12%(23)	0.10%	0.01%	89%	1.9m(2.9)
	8	6	2.2(4.3)	13%(24)	0.55%	0.05%	87%	2.2m(3.1)
	12	3	2.0(4.1)	12%(23)	1.03%	0.09%	94%	3.4m(6.0)
	12	4	2.0(4.1)	12%(23)	1.13%	0.10%	94%	3.4m(5.7)
	12	6	2.1(4.4)	12%(23)	1.30%	0.11%	92%	3.9m(6.4)

TABLE VIII
MAPF PLANNING RESULTS ON A 20X20 MAP WITH 16 AGENTS WITH DIFFERENT OBSTACLE RATES. DIFFERENT VALUES OF HYPERPARAMETERS w (PLANNING HORIZON) AND T (DECREASING THRESHOLD) ARE GIVEN FOR $r = 5$. THE ACTION IMPLEMENTATION LENGTH m IS EQUAL TO 1. A VALUE IN BRACKETS $(\cdot)$ DENOTES A STANDARD DEVIATION FROM THE CORRESPONDING MEAN VALUE

type	w	T	δ	$\frac{\delta}{\text{depth}}$	EX	EX_2	SR	average runtime
0%/100	8	2	0.18(0.56)	1.6%(5.9)	1.53%	0.13%	100%	13m(10)
	8	4	0.18(0.56)	1.6%(5.9)	1.53%	0.13%	100%	13m(10)
	8	6	0.19(0.63)	1.7%(6.4)	1.60%	0.14%	100%	12m(10)
	12	3	0.18(0.58)	1.7%(6.9)	6.44%	0.71%	100%	26m(22)
	12	4	0.18(0.55)	1.6%(6.1)	5.76%	0.66%	100%	28m(25)
	12	6	0.18(0.55)	1.6%(6.1)	5.71%	0.66%	100%	27m(23)
15%/91	8	2	0.51(1.6)	3.8%(12)	0.28%	0.03%	93%	2.9m(5.1)
	8	4	0.54(1.7)	4.0%(12)	0.32%	0.03%	93%	3.1m(4.4)
	8	6	0.56(1.7)	4.4%(14)	0.53%	0.05%	93%	3.0m(2.9)
	12	3	0.49(1.5)	3.8%(11)	0.69%	0.07%	94%	5.3m(6.5)
	12	4	0.49(1.5)	3.8%(11)	0.81%	0.09%	95%	5.7m(7.1)
	12	6	0.52(1.6)	4.0%(12)	0.85%	0.10%	95%	5.9m(7.0)
30%/74	8	2	1.6(3.6)	10%(21)	1.25%	0.15%	91%	4.2m(6.3)
	8	4	1.6(3.7)	10%(21)	1.37%	0.14%	92%	4.5m(7.0)
	8	6	1.6(3.7)	10%(21)	1.73%	0.20%	84%	4.8m(8.2)
	12	3	1.5(3.2)	9%(21)	2.77%	0.36%	92%	6.6m(9.1)
	12	4	1.6(3.3)	9%(21)	2.92%	0.37%	91%	7.4m(10)
	12	6	1.4(3.1)	9%(20)	3.02%	0.38%	93%	8.0m(14)

Appendix B

B-1 Supplementary results

Table B-1: MAPF planning results on a 10x10 map with 8 agents with different obstacle rates. The hyperparameters of the DECOP algorithm are set as described in section 4-2. A value between brackets $(\cdot)$ denotes the standard deviation to the corresponding mean value.

obst. rate	com. maps	method	δ	δ/depth	exp. limit_1	compl.	success rate	average runtime
8 agents on 10×10 map								
0%	99	PIBT	1.0(2.2)	16%(32)	-	-	100%	instant
		DECOP_0	0.22(0.59)	3.6%(11)	0.00%	67%	99%	0.5m(0.3)
		DECOP_1	0.21(0.60)	3.6%(11)	0.00%	66%	100%	0.5m(0.3)
		DECOP_2	0.19(0.54)	3.4%(11)	0.00%	72%	100%	0.6m(0.4)
15%	93	PIBT	1.5(3.0)	21%(36)	-	-	99%	instant
		DECOP_0	0.79(1.7)	11%(23)	0.16%	57%	96%	0.6m(1.4)
		DECOP_1	0.81(1.8)	11%(23)	0.16%	59%	97%	0.7m(1.3)
		DECOP_2	0.76(1.6)	11%(23)	0.24%	63%	99%	0.8m(1.2)
30%	78	PIBT	2.1(4.3)	23%(38)	-	-	86%	instant
		DECOP_0	1.2(2.5)	14%(29)	1.24%	62%	93%	1.1m(2.1)
		DECOP_1	1.3(2.7)	15%(30)	1.16%	58%	90%	1.1m(1.9)
		DECOP_2	1.2(2.6)	15%(30)	1.41%	63%	91%	1.3m(2.2)

Table B-2: MAPF planning results on a 20x20 map with 16 agents with different obstacle rates. The hyperparameters of the DECOP algorithm are set as described in section 4-2. A value between brackets $(\cdot)$ denotes the standard deviation to the corresponding mean value.

obst. rate	com. maps	method	δ	δ/depth	exp. limit_1	compl.	success rate	average runtime
16 agents on 20×20 map								
0%	99	PIBT	1.4(3.6)	12%(29)	-	-	100%	instant
		DECOP_0	0.19(0.58)	1.6%(6.2)	0.72%	51%	100%	9.2m(6.8)
		DECOP_1	0.19(0.88)	1.6%(6.6)	0.38%	51%	99%	6.7m(4.5)
		DECOP_2	0.14(0.47)	1.2%(4.5)	0.38%	63%	100%	5.5m(3.8)
15%	89	PIBT	1.8(3.9)	15%(30)	-	-	98%	instant
		DECOP_0	0.53(1.5)	4.0%(12)	0.12%	51%	92%	2.5m(2.4)
		DECOP_1	0.54(1.5)	4.1%(12)	0.21%	52%	94%	2.4m(3.0)
		DECOP_2	0.52(1.5)	4.0%(12)	0.16%	58%	99%	2.0m(1.6)
30%	81	PIBT	3.0(5.9)	18%(33)	-	-	92%	instant
		DECOP_0	2.0(4.5)	11%(23)	0.83%	48%	94%	3.2m(5.3)
		DECOP_1	2.0(4.1)	11%(22)	0.84%	49%	92%	3.6m(8.0)
		DECOP_2	2.0(4.1)	12%(23)	0.63%	51%	97%	3.5m(4.9)

Bibliography

[1] Faten Aljalaud and Nathan Sturtevant. Finding Bounded Suboptimal Multi-Agent Path Planning Solutions Using Increasing Cost Tree Search (Extended Abstract). *Proceedings of the International Symposium on Combinatorial Search*, 4(1):203–204, 2013. Number: 1.

[2] Shunsuke Aoki and Ragunathan Rajkumar. Safe Intersection Management With Cooperative Perception for Mixed Traffic of Human-Driven and Autonomous Vehicles. *IEEE Open Journal of Vehicular Technology*, 3:251–265, 2022. Conference Name: IEEE Open Journal of Vehicular Technology.

[3] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. *Proceedings of the International Symposium on Combinatorial Search*, 5(1):19–27, 2014. Number: 1.

[4] Zahy Bnaya and Ariel Felner. Conflict-Oriented Windowed Hierarchical Cooperative A*. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3748, May 2014. ISSN: 1050-4729.

[5] Satyendra Singh Chouhan and Rajdeep Niyogi. DMAPP: A Distributed Multi-agent Path Planning Algorithm. In Bernhard Pfahringer and Jochen Renz, editors, *AI 2015: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 123–135, Cham, 2015. Springer International Publishing.

[6] Satyendra Singh Chouhan and Rajdeep Niyogi. DiMPP: a complete distributed algorithm for multi-agent path planning. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(6):1129–1148, November 2017. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/0952813X.2017.1310142.

[7] Liron Cohen, Tansel Uras, T K Satish Kumar, Hong Xu, Nora Ayanian, and Sven Koenig. Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, July 2016.

[8] Azita Dabiri, Andreas Hegyi, and Serge Hoogendoorn. Optimized Speed Trajectories for Cyclists, Based on Personal Preferences and Traffic Light Information-A Stochastic Dynamic Programming Approach. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):777–793, February 2022. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[9] Azita Dabiri and Balázs Kulcsár. Distributed Ramp Metering—A Constrained Discharge Flow Maximization Approach. *IEEE Transactions on Intelligent Transportation Systems*, 18(9):2525–2538, September 2017. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[10] 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, April 2021. Conference Name: IEEE Robotics and Automation Letters.

[11] Jie Ding, K. Chakrabarty, and R.B. Fair. Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(12):1463–1468, December 2001. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[12] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28:83–87, June 2018.

[13] Oded Goldreich. Finding the Shortest Move-Sequence in the Graph-Generalized 15-Puzzle Is NP-Hard. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, Lecture Notes in Computer Science, pages 1–5. Springer, Berlin, Heidelberg, 2011.

[14] Gilad Goraly and Refael Hassin. Multi-Color Pebble Motion on Graphs. *Algorithmica*, 58(3):610–636, November 2010.

[15] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. Conference Name: IEEE Transactions on Systems Science and Cybernetics.

[16] Tamás Keviczky, Francesco Borrelli, and Gary J. Balas. Decentralized receding horizon control for large scale dynamically decoupled systems. *Automatica*, 42(12):2105–2115, December 2006.

[17] Kangjin Kim, Joe Campbell, William Duong, Yu Zhang, and Georgios Fainekos. DisCoF+: Asynchronous DisCoF with flexible decoupling for cooperative pathfinding in distributed systems. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 369–376, August 2015. ISSN: 2161-8089.

[18] D. M. Kornhauser, G. Miller, and P. Spirakis. *Coordinating pebble motion on graphs, the diameter of permutation groups and applications*. Robotics report. Courant Institute of Mathematical Sciences, New York University, New York, May 1984. OCLC: 1042475119.

[19] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12353–12362, May 2021. Number: 14.

[20] Juncheng Li, Maopeng Ran, and Lihua Xie. Efficient Trajectory Planning for Multiple Non-Holonomic Mobile Robots via Prioritized Trajectory Optimization. *IEEE Robotics and Automation Letters*, 6(2):405–412, April 2021. Conference Name: IEEE Robotics and Automation Letters.

[21] Wenhao Li, Hongjun Chen, Bo Jin, Wenzhe Tan, Hongyuan Zha, and Xiangfeng Wang. Multi-Agent Path Finding with Prioritized Communication Learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10695–10701, May 2022.

[22] Ziyuan Ma, Yudong Luo, and Hang Ma. Distributed Heuristic Multi-Agent Path Finding with Communication, June 2021. arXiv:2106.11365 [cs].

[23] Ziyuan Ma, Yudong Luo, and Jia Pan. Learning Selective Communication for Multi-Agent Path Finding. *IEEE Robotics and Automation Letters*, 7(2):1455–1462, April 2022. Conference Name: IEEE Robotics and Automation Letters.

[24] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752, September 2022.

[25] Keisuke Okumura, Yasumasa Tamura, and Xavier Défago. winPIBT: Extended Prioritized Algorithm for Iterative Multi-agent Path Finding, December 2020. arXiv:1905.10149 [cs].

[26] Afshin OroojlooyJadid and Davood Hajinezhad. A Review of Cooperative Multi-Agent Deep Reinforcement Learning, April 2021. arXiv:1908.03963 [cs, math, stat].

[27] Jungwon Park, Junha Kim, Inkyu Jang, and H. Jin Kim. Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee using Relative Bernstein Polynomial, March 2020. arXiv:1909.10219 [cs, eess].

[28] Daniel Ratner and Manfred Warmuth. The (n2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, August 1990.

[29] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, July 2019. Conference Name: IEEE Robotics and Automation Letters.

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

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

[32] David Silver. Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 1(1):117–122, 2005. Number: 1.

[33] Roni Stern. Multi-Agent Path Finding – An Overview. In Gennady S. Osipov, Aleksandr I. Panov, and Konstantin S. Yakovlev, editors, *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*, Lecture Notes in Computer Science, pages 96–115. Springer International Publishing, Cham, 2019.

[34] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks, June 2019. arXiv:1906.08291 [cs].

[35] Pavel Surynek. An Optimization Variant of Multi-Robot Path Planning Is Intractable. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):1261–1263, July 2010. Number: 1.

[36] Pavel Surynek. Compilation-based Solvers for Multi-Agent Path Finding: a Survey, Discussion, and Future Opportunities, April 2021. arXiv:2104.11809 [cs].

[37] Andrea Testa, Alessandro Rucco, and Giuseppe Notarstefano. Distributed Mixed-Integer Linear Programming via Cut Generation and Constraint Exchange. *IEEE Transactions on Automatic Control*, 65(4):1456–1467, April 2020. Conference Name: IEEE Transactions on Automatic Control.

[38] Yutong Wang, Bairan Xiang, Shinan Huang, and Guillaume Sartoretti. SCRIMP: Scalable Communication for Reinforcement- and Imitation-Learning-Based Multi-Agent Pathfinding, March 2023. arXiv:2303.00605 [cs].

[39] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1):9–9, March 2008. Number: 1.

[40] Xiuqing Yang, Ruchen Feng, Pengcheng Xu, Xiaorui Wang, and Mingyao Qi. Internet-of-Things-augmented dynamic route planning approach to the airport baggage handling system. *Computers & Industrial Engineering*, 175:108802, January 2023.

[41] Jingjin Yu. Diameters of Permutation Groups on Graphs and Linear Time Feasibility Test of Pebble Motion Problems, October 2012. arXiv:1205.5263 [cs].

[42] Jingjin Yu and Steven LaValle. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27(1):1443–1449, June 2013. Number: 1.

[43] Jingjin Yu and Steven M. LaValle. Multi-agent Path Planning and Network Flow, January 2013. arXiv:1204.5717 [cs].

[44] Jingjin Yu and Daniela Rus. Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms, July 2014. arXiv:1205.5263 [cs].

[45] Yu Zhang, Kangjin Kim, and Georgios Fainekos. DisCoF: Cooperative Pathfinding in Distributed Systems with Limited Sensing and Communication Range. In Nak-Young Chong and Young-Jo Cho, editors, *Distributed Autonomous Robotic Systems*, Springer Tracts in Advanced Robotics, pages 325–340, Tokyo, 2016. Springer Japan.

[46] Michal Čáp, Jiří Vokřínek, and Alexander Kleiner. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-formed Infrastructures. *Proceedings of the International Conference on Automated Planning and Scheduling*, 25:324–332, April 2015.

Glossary

List of Acronyms

MAPF	multi-agent path finding
PMG	pebble motion on graph
SAT	boolean satisfiability
MILP	mixed integer linear programming
ASP	answer set programming
CSP	constraint satisfaction
CBS	conflict-based search
SOC	sum of costs
ICTS	increasing cost tree search
ID	independence detection
SAPF	single-agent path finding
HCA*	hierarchical cooperative A*
WHCA*	windowed hierarchical cooperative A*
PPS	Parallel Push-and-Swap
CO-WHCA*	conflict oriented windowed hierarchical cooperative A*
PIBT	priority inheritance with backtracking
OA*t*	optimal anytime algorithm
winPIBT	windowed priority inheritance with backtracking
MARL	multi-agent reinforcement learning
RL	reinforcement learning
LMAPF	lifelong multi-agent path finding
GCN	graph convolutional network
FOV	field of view

DHC	distributed, heuristic and communication
PICO	prioritized communication learning
DCC	decision causal communication
PRIMAL	pathfinding via reinforcement and imitation multi-agent learning
PRIMAL$_2$	pathfinding via reinforcement and imitation multi-agent learning - lifelong
MDD	multi-value decision diagram
LTI	linear time-invariant
MAPD	multi-agent package and delivery
MRTA	multi-robot task allocation
MDD product	MDD product
SCRIMP	scalable communication for reinforcement- and imitation-learning-based multi-agent pathfinding
KPI	key performance indicator
IoT	internet of things
RHC	receding horizon control
MPC	model predictive control
NP	non polynomial
SR	success rate
DECOP	Decentralized Optimization

List of Symbols

$\Gamma_i(n)$	The set of connected agents based on proximity to the searching agent
Λ_i	From the perspective of agent a_i, the set of agents within the set of connected agents ($\subseteq \Gamma_i$) that have already secured a path during a planning round
$\mathcal{A}$	Set of k agents
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Transportation network graph
$\mathcal{P} = \{\pi_1, \ldots, \pi_k\}$	Solution to a MAPF problem as a set of k single agent plans
$\mathcal{R}_i(n)$	The tractable set of selected agents which the searching agent considers for local optimization
π_i	Single agent path for the ith agent
a_i	The particular ith agent
a_i	The searching agent from whose perspective the algorithm is considered
$C(\cdot, \cdot)$	Cost of traversing a particular edge
$e \in \mathcal{E}$	Single edge in a set of edges that represent transportation links
g	List of goal nodes where $g_i \in \mathcal{V}$ denotes the goal node of the ith agent
P	Maximum number of nodes to expand from the MDD product in a single local optimization graph search

r	The number of agents considered tractable for finding the optimal solution in reasonable time for the respective multi-agent path finding (MAPF) problem
s	List of start nodes for all agents where $s_i \in \mathcal{V}$ denotes the start node of the ith agent
T	Threshold that specifies the number of consecutive actions in which an agent should decrease its distance to goal in order to be considered a "decreasing agent"
$t = t_0 + n \cdot h$ with $n \in \mathbb{N}^0$	Discretized time with time step h
t_0	Initial simulation time
v	Value of the Euclidean norm that defines the maximum distance that agents can be apart to allow for communication
$v \in \mathcal{V}$	Single node in a set of nodes that represent junctions of links
MDD_i^δ	$\text{MDD}^{\text{up to } \delta \text{ steps delay}}_{\text{agent number:} a_i \in \mathcal{A}} \rightarrow$ multi-value decision diagram (MDD) for agent i with δ steps of incurred delay