Simultaneous Computation with Multiple Prioritizations in Multi-Agent Motion Planning

PATRICK SCHEFFE*, KU Leuven, Belgium
JULIUS KAHLE, RWTH Aachen University, Germany
BASSAM ALRIFAEE, University of the Bundeswehr Munich, Germany

Multi-agent path finding (MAPF) in large networks is computationally challenging. An approach for MAPF is prioritized planning (PP), in which agents plan sequentially according to their priority. Albeit a computationally efficient approach for MAPF, the solution quality strongly depends on the prioritization. Most prioritizations rely either on heuristics, which do not generalize well, or iterate to find adequate priorities, which costs computational effort. In this work, we show how agents can compute with multiple prioritizations simultaneously. Our approach is general as it does not rely on domain-specific knowledge. The context of this work is multi-agent motion planning (MAMP) with a receding horizon subject to computation time constraints. MAMP considers the system dynamics in more detail compared to MAPF. In numerical experiments on MAMP, we demonstrate that our approach achieves near-optimal prioritization and outperforms state-of-the-art methods with only a minor increase in computation time. We show real-time capability in an experiment on a road network with ten vehicles in our Cyber-Physical Mobility Lab.

JAIR Track: Multi-Agent Path Finding

JAIR Associate Editor: Daniel Harabor

JAIR Reference Format:
Patrick Scheffe, Julius Kahle, and Bassam Alrifaee. 2026. Simultaneous Computation with Multiple Prioritizations in Multi-Agent Motion Planning. *Journal of Artificial Intelligence Research* 86, Article 56 (August 2026), 33 pages. DOI: 10.1613/jair.1.20870

1 Introduction

Multi-agent path finding (MAPF) is a multi-agent planning problem in which agents must find a path from a start position to a target position on a discrete grid represented by a graph. Multi-agent motion planning (MAMP) for connected and automated vehicles (CAVs), a variant of MAPF that is considered in this article, considers a dynamic environment by frequently updating control inputs and takes the kinodynamic constraints of vehicles into account during planning. MAPF problems can be solved in a centralized fashion, where all information required to solve every agent's planning problem, such as their objective, state, and constraints, is present in a central computation entity, and this entity computes the motion plan for each agent. Alternatively, they can be solved in a distributed fashion without a centralized computation entity, where each agent computes only its own motion plan and knows only about its own objective, but communicates with other agents. MAMP in environments of large extent, such as road networks, which typically lack the infrastructure of centralized computation entities, thus requires distributed formulations. This setting is illustrated in Figure 1 with three CAVs on an intersection. A more detailed comparison of MAPF and MAMP is given in Section 4.

*Corresponding Author. Also with Flanders Make@KU Leuven and RWTH Aachen University.

Authors' Contact Information: Patrick Scheffe, ORCID: 0000-0002-2707-198X, patrick.scheffe@kuleuven.be, KU Leuven, Leuven, Belgium; Julius Kahle, ORCID: 0000-0003-3986-1986, kahle@embedded.rwth-aachen.de, RWTH Aachen University, Aachen, Germany; Bassam Alrifaee, ORCID: 0000-0002-5982-021X, bassam.alrifaee@unibw.de, University of the Bundeswehr Munich, Neubiberg, Germany.

DOI: 10.1613/jair.1.20870

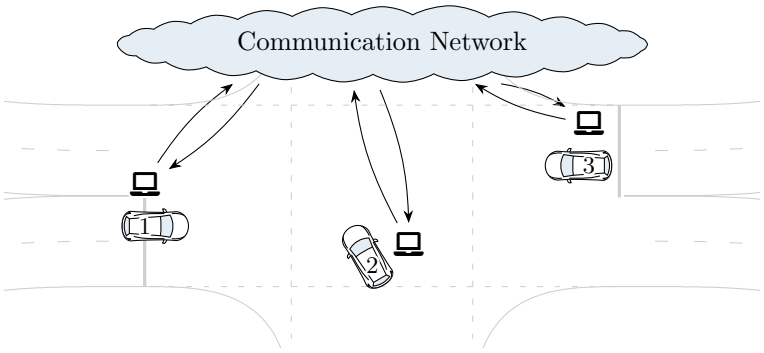


Fig. 1. The application of this work is distributed MAMP for CAVs without a central coordinator. Each agent knows only its own objective and computes only its own control input, but can communicate with other agents. Since the environment is dynamic and the planning horizon limited, agents plan frequently and should finish computation within a limited time duration.

When solving multi-agent planning problems, agents may have coupled objectives or be constrained by each other. Therefore, the decisions of agents are interdependent. Solving a MAPF problem optimally by choosing all decision variables simultaneously is NP-hard (Yu 2016; Yu and LaValle 2013). Prioritized planning (PP), introduced by Erdmann and Lozano-Pérez (1987), is a computationally efficient method for MAPF. Instead of solving a MAPF problem optimally, PP divides the problem into multiple subproblems, and each agent solves only its subproblem. Since the subproblems are smaller in size, the computation time is reduced. PP is particularly well suited to distributed planning problems because it both decomposes the MAPF problem as described above and follows a sequential, non-iterative scheme that requires only sparse communication.

1.1 Motivation

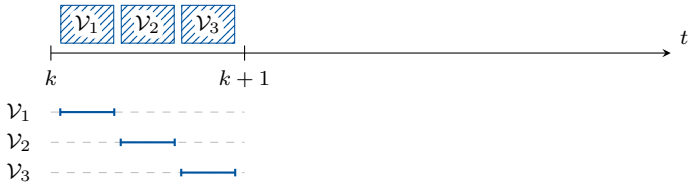
While efficient, PP often produces suboptimal solutions and is not guaranteed to find a solution at all. Let J_p denote the summed cost of all agents' optimal motion plans given a prioritization p. A motion plan is optimal given a prioritization if there is no other motion plan with equal cost that results in a motion plan of lower-priority agents with lower cost. In MAMP, there is typically a single motion plan with lowest cost due to the asymmetry of the environment; in MAPF, which often exhibits symmetry and therefore admits multiple plans with the same cost, there exist algorithms to find optimal plans given a prioritization (Morag et al. 2024).

Definition 1. The optimal prioritization p^* is the prioritization which yields the solution with highest quality, i.e., with lowest summed cost of all agents' optimal motion plans given the prioritization,

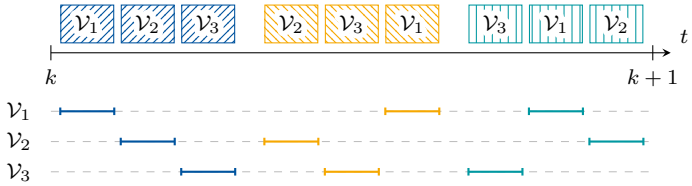
$$p^* = \arg\min_p J_p. \tag{1}$$

If we view prioritization and planning as a two-stage optimization, the solution given a prioritization can be seen as a local optimum. The solution given the optimal prioritization is denoted as the global optimum in PP, which notably can be different from the optimal solution to the multi-agent planning problem. A core challenge in PP is finding a prioritization that results in a solution of high quality.

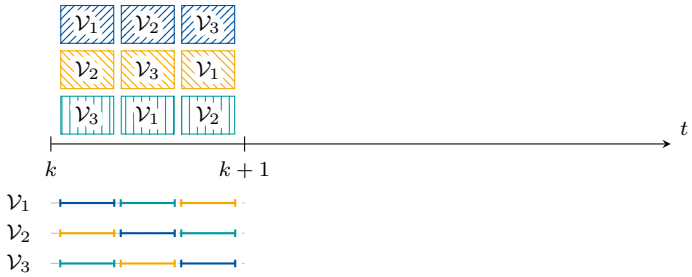
In general, for N_A agents, there exist up to $N_A!$ prioritizations. To approach the global optimum in PP, traditional approaches rely on heuristics to improve the solution quality (e.g., Chen et al. 2023; Kloock, Kragl, et al. 2019; Scheffe, Dorndorf, et al. 2022; Wu et al. 2020; Yao and F. Zhang 2018). Such heuristic approaches require domain-specific knowledge, making them unsuitable for generalized application. Further, their performance can be subpar,



(a) Solving the PP problem with a single prioritization (e.g., Chalaki and Malikopoulos 2022; Kloock, Kragl, et al. 2019; van den Berg and Overmars 2005; Wu et al. 2020; Yao and F. Zhang 2018).



(b) Solving the PP problem with three prioritizations sequentially (e.g., Bennewitz et al. 2002; Chan et al. 2023; Ma, Harabor, et al. 2019). Note how the order of computation changes with the prioritization.



(c) Solving the PP problem with three prioritizations simultaneously (our approach). Note how rows represent the different prioritizations from Figure 2b, and each agent solves only one agent planning problem corresponding to a different prioritization at a time.

Fig. 2. Exemplary illustration of the computation time in different prioritization approaches in an example with three agents represented in $\mathcal{V}_1$, $\mathcal{V}_2$, and $\mathcal{V}_3$; blocks above the time axis indicate the prioritization, i.e., the order of computation of agents; thick lines below the time axis indicate the computation time of the respective agents for the respective prioritization.

as heuristics are often tailored towards specific planning problems.

In domain-independent approaches, there is usually no a-priori knowledge about which prioritization yields a close-to-optimal solution. Thus, the prioritization must be optimized, which involves computing multiple solutions using different prioritizations. However, computing multiple solutions sequentially might be intractable for real-time applications with constrained computation time. Consequently, a simultaneous computation of multiple solutions is needed.

1.2 Contribution

In PP, solution quality depends on the chosen prioritization. Approaching the global optimum therefore requires evaluating multiple prioritizations, which is often infeasible under real-time constraints in distributed MAMP.

This article presents a method that exploits idle time induced by sequential PP to evaluate multiple prioritizations simultaneously. Our main contributions are: (i) a distributed algorithm for simultaneous evaluation of multiple prioritizations, with the same planning complexity per cycle as single-prioritization PP; (ii) a prioritization update rule that can maintain a well-performing prioritization across successive time steps. Besides these main contributions, we also extend the completeness and P-solvability definitions to time-variant prioritizations, and give a graph-theoretic upper bound on the number of distinct prioritizations that is tighter than the worst-case bound of $N_A!$ for sparse coupling graphs, with N_A being the number of agents.

Figure 2 illustrates our main idea in an example with three interacting, sequentially computing agents. Blocks display the prioritization, thick lines represent the computation time of agents. Figure 2a shows how approaches based on heuristics solve the PP problem for only a single prioritization. The solution quality strongly depends on the heuristic. Figure 2b shows how the PP problem can be solved multiple times sequentially regarding multiple prioritizations. Even though it can improve solution quality, it prolongs the computation time. In this work, we make use of the fact that in sequential computation the agents idle most of the time: in the example shown in Figure 2a, agents idle two-thirds of the available time. Figure 2c showcases our approach: By utilizing the idle time, we solve the PP problem regarding the three prioritizations from Figure 2b simultaneously.

Scope and assumptions. This article focuses on improving solution quality given short computation time in distributed MAMP. Accordingly, our evaluation isolates the planning component and does not model communication latency or packet loss. To isolate the effect of simultaneous prioritization without conflating it with network effects, we assume reliable, low-latency communication. This assumption is based on ongoing developments in communication technology targeting reliable, low-latency communication, such as 5G ultra-reliable and low-latency communications (URLLC) (e.g., Z. Li et al. 2018). A full end-to-end evaluation that explicitly includes communication time remains an important extension, but is outside the scope of this article.

We also consider only environments with exclusively CAVs, i.e., without uncontrolled vehicles. This restriction matches controlled environments such as warehouses or fully autonomous road traffic.

1.3 Related Work

This section reviews approaches from the literature mapped to a distributed computation architecture in which each agent computes its respective motion plan. We illustrate the computations of the literature in Figure 2, which displays the computations of three interacting, sequentially computing agents. The blocks represent a prioritization, i.e., an order of computations, while the thick lines display the computation time.

When searching a prioritization of high solution quality, heuristics are often consulted. Heuristics aim to improve the prioritization mostly for their domain-dependent objective, and are thus less likely to generalize well. The following approaches use objective-based heuristics for prioritization to improve the solution quality. As illustrated in Figure 2a, these approaches solve the PP problem using a single prioritization. In (Wu et al. 2020), the prioritization is determined by the number of possible paths a robot can take to reach its goal. The lower this number is, the higher a robot's priority. Thereby, the probability for each robot to find a solution is meant to be increased, which directly affects the solution quality. In (van den Berg and Overmars 2005), robots with longer estimated travel distances receive higher priority to more evenly distribute the travel time among robots. S. Zhang et al. (2022) proposes a prioritization algorithm based on machine learning, which performs competitively compared to heuristic algorithms. Our previous work (Kloock, Kragl, et al. 2019) prioritizes agents with the goal of increasing the traffic flow rate at a road intersection through a heuristic based on the remaining time before an agent enters the intersection. In (Yao and F. Zhang 2018), automated vehicles approaching an

intersection are prioritized using mixed integer programming. Both the prioritization and the vehicle's speed are optimized to minimize the travel time of the vehicles. The algorithm in (Chalaki and Malikopoulos 2022) prioritizes vehicles on intersections using job-shop scheduling and thereby reduces the vehicles' average travel time. Most of the above approaches have in common that the solution for a single prioritization is computed. Consequently, their computation time is low, as the complexity is linear in the number of agents, $O(N_A)$. However, the solution quality can be lower than with a different prioritization, or they can fail to find a solution, even though one exists with a different prioritization.

Instead of finding a prioritization heuristically, other works optimize the prioritization. As illustrated in Figure 2b, these approaches solve the networked planning problem by exploring multiple prioritizations sequentially. These approaches aim to improve the solution quality by investing computation time. Bennewitz et al. (2002) propose a randomized hill-climbing search for finding the optimal prioritization. Beginning at an initial prioritization, priorities of agents are swapped randomly to increase the solution quality. Recomputing the solution in each iteration increases the computation effort. Chan et al. (2023) and Ma, Harabor, et al. (2019) present an algorithm to lazily explore the space of all prioritizations. Whenever a conflict between two agents occurs, both possible prioritizations are added to a search tree. Thus, the computation effort depends on the number of needed iterations until a conflict-free prioritization is found. The above approaches can compute solutions to multiple prioritizations, and thus find solutions with higher quality than approaches considering only a single prioritization. However, the solution quality improvement comes at the cost of higher computation time. For example, the worst-case effort of conflict-based approaches is $O(N_A \cdot 2^{N_A^2})$, as for each conflict up to N_A plans must be recomputed, and for each of the possible N_A^2 conflicts two options for prioritization of the involved two agents exist.

Many of the above approaches to PP have a centralized computation architecture, i.e., a single, central processing unit computes solutions for all agents in sequence (e.g., Chalaki and Malikopoulos 2022; Chan et al. 2023; Ma, Harabor, et al. 2019; van den Berg and Overmars 2005; Yao and F. Zhang 2018; S. Zhang et al. 2022). Other centralized approaches explore a larger solution space than in PP through a conflict-based search (e.g., Barer et al. 2021; Guo and Yu 2024; J. Li, Ruml, et al. 2021; Pan et al. 2024; Sharon et al. 2015; Yu 2020). Although proposed as centralized approaches, they solve local subproblems and could thus be also applied to a distributed computation architecture. The iterative conflict resolution of the latter group of approaches can result in more sequential computations and communication than in approaches that solve a single prioritization, making them unsuitable when communication resources are scarce. Even though their computation is efficient in practice, the number of planning iterations is typically unbounded, making them unsuitable for real-time computation.

PP is suited for distributed execution by multiple agents with processing units on each agent (e.g., Čáp et al. 2015; Kloock, Kragl, et al. 2019; Kuwata et al. 2007; Wu et al. 2020). Its non-iterative computation scheme results in scarce usage of communication resources. These characteristics make it well-suited for the problem setting of MAMP with distributed computation and constrained computation time of this article.

With multiple processing units, a sequential computation regarding a prioritization yields periods of idle time in each unit. The problem of reducing idle time due to sequential operation is addressed in production automation. Johnson's rule (Almhanna et al. 2023) and sequencing (Mak et al. 2014) are methods to decrease idle time by optimizing the assignment and sequence for jobs on machines in assembly lines. Further, Çam and Sezen (2020) reduce idle time of control systems, i.e., unproductive time of the system, but not idle time during computation. In this article, we will utilize idle time in sequential computation for simultaneous computation of multiple prioritizations in PP, which should increase solution quality without increasing computation time.

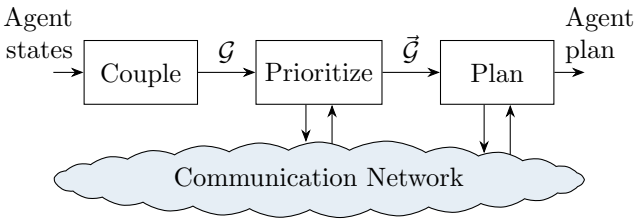


Fig. 3. The framework for PP in which our approach is embedded. Each agent builds an undirected coupling graph $\mathcal{G}$ which represents potential conflicts, negotiates a prioritization with the other agents to build a directed coupling graph $\vec{\mathcal{G}}$, and plans according to the directed coupling graph.

1.4 Structure of This Article

The rest of this article is organized as follows. In Section 2, we formalize time-variant coupling and prioritization of agents in a receding horizon planning framework. In Section 3, we formulate requirements for simultaneously computable prioritizations. We further present our algorithm for simultaneous computation, which can both retain prioritizations to utilize the predictive nature of receding horizon planning, and explore new prioritizations to increase the likelihood of finding a solution. In particular, the time required for exploration is spread across consecutive time steps, which keeps computation time at each time step short. We present our multi-agent motion planner for CAVs driving on roads in Section 4. In Section 5, we present a comparison of our approach to state-of-the-art prioritization approaches in numerical experiments with CAVs, and showcase real-time capabilities of our approach in an experiment with ten CAVs in our Cyber-Physical Mobility Lab (CPM Lab).

2 Preliminaries

This section provides the background to prioritized receding horizon planning. In PP, agents first receive all plans from their predecessors, then solve the planning problem, and finally send their plans to their successors. To determine the predecessors and successors of agents, we couple and prioritize them. An overview of the PP framework is given in Figure 3. In this work, we use exclusively distributed and decentralized algorithms for all components of the framework. The following subsections introduce our notation and the background on coupling, prioritization, and receding horizon planning.

2.1 Notation

In this article, we refer to agents whenever concepts are generally applicable to PP. We assume that states of agents are observable, and therefore we refer to states instead of outputs. The definitions and methods can be transferred to outputs as well, which we omit for brevity. We denote scalars with normal font, vectors with bold lowercase letters, matrices with bold uppercase letters, and sets with calligraphic uppercase letters. For any set $\mathcal{S}$, the cardinality of the set is denoted by $|\mathcal{S}|$. A variable x is marked with a superscript $x^{(i)}$ if it belongs to agent i. The actual value of a variable x at time k is written as $x(k)$, while values predicted for time $k+l$ at time k are written as $x_{k+l|k}$. A trajectory is denoted by replacing the time argument with $(\cdot)$ as in $x_{\cdot|k}$.

This article addresses motion planning for multiple agents. We refer to the entirety of all agents and the communication links between them as a networked control system (NCS), i.e., a control system whose components are connected by a communication network. We use the prefix *networked* throughout to denote quantities of the NCS as a whole, as opposed to those of an individual agent. We use graphs as a modeling tool of NCS. Every agent is associated with a vertex, so the terms are used synonymously.

(a) Example undirected coupling graph.

(b) Example directed coupling graph. An agent can start computation as soon as all its predecessors in the graph have finished their computations and communicated their results.

Fig. 4. Example undirected coupling graph that is transformed to a directed coupling graph by orienting the edges according to the agents' priorities. In the example, agent priority is according to vertex number.

Definition 2 (Undirected graph)**.** An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair of two sets, the set of vertices $\mathcal{V} = \{1, \ldots, N_A\}$ and the set of undirected edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The edge between i and j is denoted by $(i - j)$.

Definition 3 (Directed graph)**.** A directed graph $\vec{\mathcal{G}} = \left(\mathcal{V}, \vec{\mathcal{E}}\right)$ is a pair of two sets, the set of vertices $\mathcal{V} = \{1, \ldots, N_A\}$ and the set of directed edges $\vec{\mathcal{E}} \subseteq \mathcal{V} \times \mathcal{V}$. The edge from i to j is denoted by $(i \to j)$. An oriented graph is a directed graph obtained from an undirected graph by replacing each edge $(i - j)$ with either $(i \to j)$ or $(j \to i)$.

2.2 Coupling Agents

If agents are related via their objectives or constraints, we speak of coupled agents. A coupling graph represents this relation between agents.

Definition 4 (Undirected coupling graph)**.** An undirected coupling graph $\mathcal{G}(k) = \big(\mathcal{V}, \mathcal{E}(k)\big)$ is a graph that represents the interaction between agents at time step k. Vertices $i \in \mathcal{V} = \{1, \ldots, N_A\}$ represent agents, and edges $(i - j) \in \mathcal{E}$ represent coupling objectives and constraints between agents.

An example coupling graph with four agents is displayed in Figure 4a. The application in which PP is used determines which coupling objectives and constraints must be considered in an agent's planning problem. For example, in MAPF, a constraint is introduced to achieve collision avoidance between two agents. Such a constraint relates the two agents, so they are coupled.

Early approaches to PP coupled all agents, resulting in purely sequential computation (e.g., Erdmann and Lozano-Pérez 1987). Later approaches reduced average computation time by coupling agents ad hoc during planning whenever a conflict arises, thereby interweaving planning and coupling (e.g., Čáp et al. 2015). However, the worst-case computation time remains that of purely sequential computation. Other approaches determine couplings before planning starts using conservative approximations that are less restrictive than coupling all agents. For example, in robot applications with a fixed time horizon, the robots' movement range over that horizon can be used to determine couplings (Baras et al. 2003; Kuwata et al. 2007; Scheffe, Xu, et al. 2024). Alternatively, if robots follow fixed paths, path intersections define couplings between robots. This is often the case for road vehicles following lanes or for warehouse robots (J. Li, Hoang, et al. 2023; Ma, J. Li, et al. 2017). While pre-planning coupling has a higher expected computation time than ad hoc coupling due to its conservativeness, it has the advantage that the number of required sequential computations is known before planning begins. This knowledge allows limiting per-agent planning time to enable real-time operation.

2.3 Prioritizing Agents

In PP, we prioritize agents, which results in clear responsibilities considering coupling objectives and constraints during planning (Alrifaee et al. 2016; Kuwata et al. 2007). A fixed or time-invariant prioritization function

$p\colon \mathcal{V} \to \mathbb{N}$ prioritizes every agent. If $p(i) < p(j)$, then agent i has a higher priority than agent j.

By prioritizing, we can orient the edges of an undirected coupling graph to form a directed coupling graph.

Definition 5 (Directed coupling graph)**.** A directed coupling graph $\vec{\mathcal{G}}(k) = \left(\mathcal{V}, \vec{\mathcal{E}}(k)\right)$ is an oriented graph obtained from orienting the edges of an undirected coupling graph at time step k. If an edge $(i \to j)$ is directed from agent i to agent j, then agent j is responsible for considering the respective coupling objective and constraint in its planning problem.

An edge points towards the vertex with lower priority,

$$(i \to j) \in \vec{\mathcal{E}} \iff p(i) < p(j) \wedge (i - j) \in \mathcal{E}. \tag{2}$$

To relate agents that are coupled only indirectly, we use the notion of a path.

Definition 6 (Path)**.** A path from vertex i_1 to vertex i_{n+1} in a directed graph $\vec{\mathcal{G}}$ is a sequence of $n \geq 1$ edges

$$(e_z)_{z=1}^{n} = \big((i_1 \to i_2), (i_2 \to i_3), \ldots, (i_n \to i_{n+1})\big), \tag{3}$$

with the domain $\{1, \ldots, n\}$ and the codomain $\vec{\mathcal{E}}$, connecting the pairwise distinct vertices $i_1, \ldots, i_{n+1}$. The length of the path is defined as the number of edges n.

The prioritization function thus constitutes a strict partial order $\prec$ on the agent set $\mathcal{V}$, given by the transitive closure of the edge relation of $\vec{\mathcal{G}}$,

$$i \prec j \iff \text{there is a path from } i \text{ to } j \text{ in } \vec{\mathcal{G}}. \tag{4}$$

The order is partial, as opposed to total, since a relation exists only for agents that are coupled directly or indirectly. For the strict partial order—and thus the orientation of every edge—to be well-defined, a valid prioritization function needs to assign pairwise different priorities to vertices that are connected by an edge:

$$p(i) \neq p(j), \quad \forall\, i, j \in \mathcal{V}, \forall\, (i - j) \in \mathcal{E}, i \neq j. \tag{5}$$

Given the construction rule in (2), the directed coupling graph $\vec{\mathcal{G}}$ resulting from the undirected coupling graph $\mathcal{G}$ and a valid prioritization function p regarding (5) is a directed acyclic graph (DAG) (Scheffe, Kahle, et al. 2025).

Definition 7 (Distinct prioritizations)**.** Two prioritizations are considered distinct if they constitute a different strict partial order, or equivalently, if they induce a different directed coupling graph via (2).

We distinguish prioritizations in this way because prioritizations that induce the same directed coupling graph yield identical planning problems.

Since in PP, an agent requires its predecessors' plans to consider coupling objectives and constraints, the agents must solve their planning problems in a sequential order. This order, as well as the communication links, is deducible from the directed coupling graph by traversing the graph along its edges. An example directed coupling graph with four agents is displayed in Figure 4b.

2.4 Computation Time Definition

In general, the computation time of MAMP is the duration from measuring the states until control inputs are determined, which consists of the planning time in a locally distributed setting, and additional communication time in a physically distributed setting. This article is on prioritization of agents, which influences the required number of sequential computations. Since, in our problem setting, the planning is computationally demanding, we simplify the communication time to be constant and negligible compared to the planning time. The computation time T of the NCS is thus mainly determined by the time to solve the agents' planning problems.

With the following definitions, we formalize the computation time of the NCS based on our previous work (Scheffe, Kahle, et al. 2025).

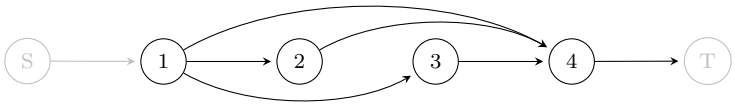


Fig. 5. Computation graph for the coupling graph in Figure 4b.

Definition 8 (Degree)**.** The degree

$$d^{(i)} = |\mathcal{V}^{(i)}| \tag{6}$$

of a vertex i denotes the number of its adjacent vertices. The number of incoming edges called in-degree is denoted by

$$d^{(i\leftarrow)} = |\mathcal{V}^{(i\leftarrow)}|. \tag{7}$$

The number of outgoing edges called out-degree is denoted by

$$d^{(i\rightarrow)} = |\mathcal{V}^{(i\rightarrow)}|. \tag{8}$$

Let the sources of $\vec{\mathcal{G}}$ be the set of vertices without incoming edges

$$\mathcal{V}_s = \left\{ i \in \mathcal{V} \mid d^{(i\leftarrow)} = 0 \right\} \tag{9}$$

and the sinks be the set of vertices without outgoing edges

$$\mathcal{V}_t = \left\{ i \in \mathcal{V} \mid d^{(i\rightarrow)} = 0 \right\}. \tag{10}$$

Note that since the coupling graph is a DAG, there is at least one source and one sink. Note further that the number of agent classes N_c is the diameter of $\vec{\mathcal{G}}$ plus one. The diameter of a graph is the greatest length of any shortest path (Definition 6) between each pair of vertices $\mathcal{V} \times \mathcal{V}$ that is connected by a path.

To determine the computation time of the NCS, we add a virtual source S and a virtual sink T to the set of vertices $\mathcal{V}$ of $\vec{\mathcal{G}}$, resulting in the set of vertices $\mathcal{V}'$. We furthermore add edges from the source S to all vertices in $\mathcal{V}_s$, and from all vertices in $\mathcal{V}_t$ to the sink T, resulting in the set of edges $\vec{\mathcal{E}}'$.

To obtain the computation time T, we weight edges by a weighting function $f_w \colon \mathcal{V}' \times \mathcal{V}' \to \mathbb{R}$

$$f_w\big((i \to j)\big) = \begin{cases} T^{(i)}, & \text{if } i \in \mathcal{V}, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

with the computation time $T^{(i)}$ of an edge's respective starting vertex. The result is the computation graph $\vec{\mathcal{G}}' = (\mathcal{V}', \vec{\mathcal{E}}', f_w)$. Figure 5 illustrates a computation graph. Let $P_{\max} \subseteq \vec{\mathcal{E}}'$ denote the longest path between S and T in $\vec{\mathcal{G}}'$. The weight of this path corresponds to the computation time of the NCS

$$T = \sum_{(i\to j)\in P_{\max}} f_w\left((i \to j)\right). \tag{12}$$

2.5 Receding Horizon Planning with Priorities

Traditional MAPF is a multi-agent planning problem in which agents move in an environment represented by a graph $\mathcal{G}_M = (\mathcal{V}_M, \mathcal{E}_M)$, consisting of a set of vertices $\mathcal{V}_M$, which are locations, and a set of edges $\mathcal{E}_M$, which are paths between locations. The objective for each agent i is to move quickly from a start vertex $s_i \in \mathcal{V}_M$ to a target vertex $t_i \in \mathcal{V}_M$. The constraints for each agent are to avoid collisions with other agents. A collision occurs when two agents i and j at a time k are located at the same vertex v, represented by a tuple $\langle i, j, v, k\rangle$, or when they traverse the same edge $(v - u) \in \mathcal{E}_M$, represented by a tuple $\langle i, j, v, u, k\rangle$.

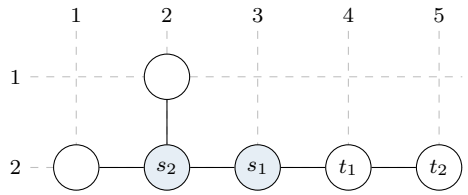


Fig. 6. A MAPF instance which is not TP-solvable.

A strategy to reduce computation time in MAPF is to introduce a fixed time horizon. Instead of solving the MAPF problem for the optimal sequence of actions from start to end, only a certain number of time steps N_p is considered. From the sequence of optimal actions for the fixed time horizon of duration $N_p \cdot T_s$, agents apply only the first. The problem is solved repeatedly after a time step duration T_s over the course of time. This practice is known as receding horizon planning or online replanning (J. Li, Tinka, et al. 2021; Scheffe, Pedrosa, et al. 2023; Shahar et al. 2021; Silver 2005).

In the context of PP, priorities can be reassigned in each time step of receding horizon planning, which results in a time-variant prioritization. We redefine the time-invariant prioritization function to a time-variant prioritization function. It is a function $p\colon \mathcal{V} \times \mathbb{N} \to \mathbb{N}$ which prioritizes each agent i at each time step k. We indicate a time-invariant prioritization by replacing the time argument with a dash, as in $p(i, -)$. In our approach, the priority ordering is consistent during planning for the planning horizon, but can change whenever agents plan.

In accordance with the usage in the literature (e.g., Ma, Harabor, et al. 2019; Wu et al. 2020), we define completeness as follows.

Definition 9 (Completeness)**.** A prioritized planning method is complete if, given an underlying planning method, any prioritization results in a solution for every MAPF instance.

We extend the considerations in (Ma, Harabor, et al. 2019) from fixed prioritizations to time-variant prioritizations.

Theorem 1. *In general, prioritized planning with time-variant prioritization is incomplete.*

Proof. A counterexample is given in Figure 6. This example requires the highest-priority agent to take a suboptimal action with regard to its own objective in order to create a solution to the MAPF problem. The highest-priority agent, however, always takes an optimal action in prioritized planning. □

Definition 10 (P-solvable, according to (Ma, Harabor, et al. 2019))**.** A MAPF instance is P-solvable iff there exists a solution for some fixed prioritization $p(i, -)$.

Definition 11 (TP-solvable)**.** A MAPF instance is TP-solvable iff there exists a solution for some time-variant prioritization $p(i, k)$.

Theorem 2. *The class of TP-solvable MAPF instances includes the class of P-solvable MAPF instances.*

Proof. A time-variant prioritization can mimic a fixed prioritization by using the same ordering at every time step,

$$p(i, k) := p(i, -). \tag{13}$$

Therefore, every MAPF instance that is solvable with a fixed prioritization is solvable with a time-variant prioritization. □

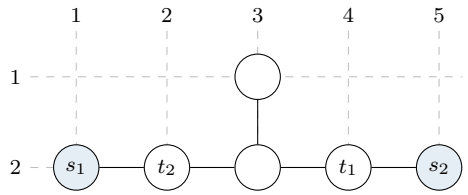


Fig. 7. A MAPF instance which is not P-solvable (Ma, Harabor, et al. 2019), but is TP-solvable with a priority flip in the third time step. Figure redrawn from (Ma, Harabor, et al. 2019).

Theorem 3. *Prioritized planning with a time-variant prioritization is incomplete for the class of TP-solvable MAPF instances.*

Proof. For the MAPF instance displayed in Figure 7, a time-variant prioritization that does not result in a solution is any fixed prioritization. □

3 Exploring Multiple Prioritizations Simultaneously

This section presents the core idea of this article, which is to improve the solution quality compared to a given prioritization by exploring multiple other prioritizations simultaneously, as illustrated in Figure 2c. In the PP framework (Figure 3), our approach replaces both the elements "Prioritize" and "Plan", as we will identify multiple prioritizations for which agents can plan simultaneously. We formalize parallelizable agent computations by the introduction of agent classes in Section 3.1. We present in Section 3.2 conditions to identify multiple prioritizations which can be computed simultaneously by these agent classes, which allows us to improve solution quality without increasing computation time. Building on these conditions, we present the algorithm for selecting a set of prioritizations and simultaneously computing with these prioritizations in Section 3.3. In Section 3.4, we show theoretically that our approach can both retain prioritizations to utilize the predictive nature of receding horizon planning and explore new prioritizations to increase the likelihood of finding a solution.

3.1 Computation Sequences of Agent Classes

In this section, we summarize how computations of agents can be executed in parallel as introduced in (Alrifaee et al. 2016). With the notion of agent classes to describe agents that can compute in parallel, we describe the computation order of agent classes with computation sequences and link computation sequences to prioritization functions.

The computation order of agents in PP can be structured with agent classes $\mathfrak{V}$. Agents belonging to the same agent class can solve their planning problem in parallel. An undirected coupling graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a prioritization function p form a directed coupling graph $\vec{\mathcal{G}}$ with (2). By topologically sorting the directed coupling graph, we map agents to classes $\mathfrak{V}$ according to our Algorithm 1. This algorithm is executed by every agent simultaneously. The algorithm inspects all vertices to find sources, i.e., vertices with no incoming edges (Line 7). All sources can start their computation, so we add them to a class (Line 8). The resulting class is next in solving its planning problem, so it is added to the computation sequence (Line 11). The processed vertices are removed from the set of vertices to process (Line 12). Finally, all edges connected to the processed vertices are removed from the coupling graph (Lines 13 to 15). The loop repeats until all vertices are assigned to classes. The output of Algorithm 1 is a sequence of agent classes $(s_z)_{z=1}^{N_c}$ which orders the agent classes $\mathfrak{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_{N_c}\}$. This computation sequence represents the order in which the agent classes can solve their planning problems.

Algorithm 1 Find agent classes, adapted from Alrifaee et al. (2016)

Input: Directed coupling graph $\vec{\mathcal{G}} = (\mathcal{V}, \vec{\mathcal{E}})$
Output: Agent classes in sequence $(s_z)_{z=1}^{N_c}$

$N_c \leftarrow 0$ ▷ *number of classes*
$\mathcal{V}_{\text{todo}} \leftarrow \mathcal{V}$ ▷ *remaining agents*
while $\mathcal{V}_{\text{todo}} \neq \emptyset$ **do**
  $N_c \leftarrow N_c + 1$
  $\mathcal{V}_{N_c} \leftarrow \emptyset$
  for $i \in \mathcal{V}_{\text{todo}}$ **do**
    if $d^{(i\leftarrow)} = 0$ **then**
      $\mathcal{V}_{N_c} \leftarrow \mathcal{V}_{N_c} \cup i$ ▷ *sources can compute*
  if $\mathcal{V}_{N_c} = \emptyset$ **then**
    return FAILURE ▷ *Graph is no DAG*
  $s_{N_c} \leftarrow \mathcal{V}_{N_c}$
  $\mathcal{V}_{\text{todo}} \leftarrow \mathcal{V}_{\text{todo}} \setminus \mathcal{V}_{N_c}$
  for $i \in \mathcal{V}_{N_c}$ **do**
    for $j \in \mathcal{V}_{\text{todo}}$ **do**
      $\vec{\mathcal{E}} \leftarrow \vec{\mathcal{E}} \setminus (i \rightarrow j)$
return $(s_z)_{z=1}^{N_c}$

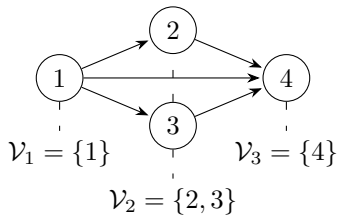


Fig. 8. Example coupling graph for Algorithm 1, and resulting agent classes in sequence.

Definition 12 (Computation sequence)**.** The set of agent classes $\mathfrak{V}$ is a countable set with $|\mathfrak{V}| = N_c$. The computation sequence defined as

$$(s_z)_{z=1}^{N_c} = \left(s_1, s_2, \ldots, s_{N_c}\right) \tag{14}$$

with the domain $\{1, \ldots, N_c\}$ and the codomain $\mathfrak{V}$ represents a permutation of the agent classes in $\mathfrak{V}$.

Figure 8 shows an example of Algorithm 1 for the directed coupling graph with four agents given in Figure 4b. The resulting computation sequence is $\left(\{1\}, \{2, 3\}, \{4\}\right)$.

Algorithm 1 returns a computation sequence with the minimum number of sequential computations for the given directed coupling graph. If we can invert this process, i.e., generate a valid prioritization corresponding to a computation sequence, we can abstract the exploration of different prioritizations to the exploration of different computation sequences. W.l.o.g., agents in different classes have different priorities,

$$p(i) \neq p(j), \quad \forall \mathcal{V}_1, \mathcal{V}_2 \in \mathfrak{V}, \mathcal{V}_1 \neq \mathcal{V}_2, \forall\, i \in \mathcal{V}_1, j \in \mathcal{V}_2. \tag{15}$$

According to (5), agents in the same class may have equal priorities as they are not connected by an edge. However, since we allow time-variant coupling graphs, we enforce unique priorities among agents (cf. Section 3.4). A valid prioritization corresponding to the computation sequence is given by

$$p(i) = Z \cdot N_A + i, \quad i \in s_Z \in (s_z)_{z=1}^{N_c}, \tag{16}$$

with an agent i, and the sequence index Z. For the example in Figure 8, this function would yield

$$p(1) = 5, \quad p(2) = 10, \quad p(3) = 11, \quad p(4) = 16. \tag{17}$$

With (16), we can express the search for a prioritization of high solution quality as the search of a computation sequence permutation of high solution quality.

3.2 Parallelizable Prioritizations

The problem of finding the optimal permutation of agent classes can formally be stated as follows.

Problem 1. Given a set of agent classes $\mathfrak{V}$, find a computation sequence $(s_z)_{z=1}^{N_c}{}^*$ resulting in a prioritization function p_q with (16) that minimizes the sum of the costs of all agents

$$p_q = \arg\min_p J_p. \tag{18}$$

Note that finding the optimal sequence does not necessarily coincide with finding the optimal prioritization, as the permutations of sequences may not express the permutations of prioritizations.

Problem 1 is known as the Drilling Problem, which is a problem in combinatorial optimization (Korte and Vygen 2018, p.1). Given a set of holes that need to be drilled, the Drilling Problem consists of finding a permutation of the order in which they need to be drilled to minimize the total distance the drill needs to travel. As the set of classes contains $|\mathfrak{V}| = N_c$ elements, for Problem 1, this would result in $|\mathcal{S}| = N_c!$ different permutations. A possible solution to that problem is enumerating all $N_c!$ sequences (Azarm and Schmidt 1997; Korte and Vygen 2018): Given a set $\mathfrak{V}$ of N_c classes, solve PP problems regarding each of the $N_c!$ sequences and choose the one with minimal cost. However, the computation time of this approach increases rapidly with the number of classes.

Solving Problem 1 is computationally intractable with real-time constraints. According to the goal of this article to increase solution quality without increasing computation time, we aim to solve the following problem.

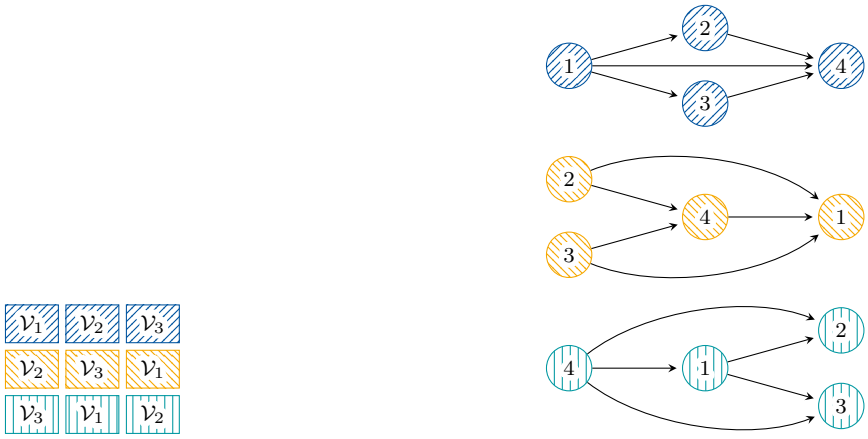
Problem 2. Given a set of agent classes $\mathfrak{V}$, find $N_c = |\mathfrak{V}|$ different computation sequences such that all sequences can be computed simultaneously.

Our goal in Problem 2 is to find N_c sequences out of $N_c!$ that can be computed simultaneously in a distributed setting. A class $\mathcal{V}_l \in \mathfrak{V}$ can solve exactly one planning problem regarding a specific computation sequence in each time slot. Note that in a centralized setting any computation sequence can be computed simultaneously, given enough computational resources. However, this is only possible since all information required to solve every agent's planning problem, such as their objective, state, and constraints, is present in the central entity.

To formalize the task, we introduce a computation schedule matrix $\boldsymbol{S} \in \mathfrak{V}^{N_c \times N_c}$. Each cell contains an agent class, and each row contains a computation sequence. Each column of $\boldsymbol{S}$ represents a time slot in the networked computation. The computation schedule matrix thus defines for each agent the prioritization for which it should plan: an agent looks up its agent class and finds the corresponding prioritization, i.e., the computation sequence, in the matrix. The matrix $\boldsymbol{S}$ represents a valid computation schedule if its entries fulfill

$$s_{ij} \neq s_{ik}, \qquad i, j, k \in \{1, \dots, N_c\}, \ j \neq k, \text{ and} \tag{19}$$

$$s_{ij} \neq s_{kj}, \qquad i, j, k \in \{1, \dots, N_c\}, \ i \neq k. \tag{20}$$



(a) Example computation schedule matrix.

(b) Corresponding directed coupling graphs; vertices are arranged in rows and columns of the computation schedule matrix. Edges in black between agents indicate coupling. An agent can start computation as soon as all its predecessors in the graph have finished their computations and communicated their results.

Fig. 9. Example computation schedule matrix and computation graph for our explorative prioritization. Four agents in three agent classes simultaneously compute three different prioritizations. Agent classes are from the example in Figure 8.

We ensure valid computation sequences with (19), and we ensure that each class solves exactly one planning problem in each time slot with (20). The matrix resulting from these conditions is also known as a Latin square. Adapted from McKay and Wanless (2005), a Latin square $\boldsymbol{L} \in \{1, \dots, N\}^{N \times N}$ is a matrix in which each row and column contains only distinct entries. That is, each row and column contains every element in $\{1, \dots, N\}$ exactly once. For $\boldsymbol{S}$, the property of distinct entries in each row and column is given by (19) and (20), respectively. The problem is related to a Sudoku puzzle: the solution to a Sudoku puzzle is a Latin square with $N = 9$ with the additional rule that each of the nine non-overlapping 3×3 partial matrices contains every element in $\{1, \dots, N\}$ exactly once.

An example computation schedule matrix and the corresponding coupling graphs are displayed in Figure 9. Starting from the prioritization and resulting agent classes of the example in Figure 8, a valid computation schedule matrix is presented in Figure 9a. Note how each agent class appears exactly once per column (at the same time, an agent class can only do one computation) and once per row (each computation sequence permutation requires one computation per agent class). The three coupling graphs corresponding to the computation schedule matrix are given in Figure 9b. With three different coupling graphs, three different solutions are computed simultaneously.

3.3 Simultaneous Computation with Multiple Prioritizations

This section describes our algorithms for simultaneous computation with multiple prioritizations. We first present our decentralized algorithm through which every agent obtains the same computation schedule matrix which fulfills (19) and (20). We then show how agents solve PP problems with multiple prioritizations. Finally, we show how agents select a solution for the PP problems.

Our goal for the computation schedule matrix $\boldsymbol{S}$ is to explore N_c prioritizations in a time step to increase the probability of finding the optimal prioritization according to Problem 1. Algorithm 2 is our decentralized

algorithm to find a computation schedule matrix $\boldsymbol{S}$, which is executed by each agent simultaneously. We restate that a row $\boldsymbol{s}_{q\cdot} \in \mathfrak{V}^{1\times N_c}$ corresponds to a computation sequence q, and a column $\boldsymbol{s}_{\cdot m} \in \mathfrak{V}^{N_c\times 1}$ corresponds to a time slot m during the networked computation. First, we initialize the computation schedule matrix $\boldsymbol{S}$ with the size $N_c \times N_c$ and fill the first row $\boldsymbol{s}_{1\cdot}$ with the classes of $\mathfrak{V}$ in sequence (Lines 1 and 2). Note that this initial row can change from one time step to another as it depends on the initial prioritization. We enforce a common random seed among agents for consistent results of the algorithm (Line 4). The algorithm determines the remaining rows in a loop (Line 5). For each row, the algorithm loops over all N_c columns that need to be populated (Lines 7 and 8). We use the number of available classes in each column that can form a valid computation schedule matrix as a heuristic for the order in which the columns are populated (Line 10). This number of available classes is stored in the `options` array. It is an array of sets, each containing all elements of $\mathfrak{V}$ that respect (19) regarding the current row q and that respect (20) regarding the column corresponding to the array index. The next column to populate is the one with the least number of available classes (Line 11). One of these classes is selected at random (Line 15). If all columns are populated with valid entries, we progress to the next row (Line 18). If there is a column for which no classes are available to select from, the current row is incompatible with the rest of the computation schedule matrix (Line 12), so we reset the current row (Line 20) and start over.

Algorithm 2 Decentralized construction of computation schedule matrix $\boldsymbol{S}$

Input: Set of agent classes $\mathfrak{V} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_{N_c}\}$
Output: Computation schedule matrix $\boldsymbol{S}$

1: $\boldsymbol{S} \leftarrow$ empty matrix of size $N_c \times N_c$
2: $\boldsymbol{s}_{1\cdot} \leftarrow [\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_{N_c}]$
3: $q \leftarrow 1$
4: initialize random seed with current time step ▷ *for consistency across agents*
5: **while** $q < N_c$ **do** ▷ *find remaining columns*
6: `is_valid` $\leftarrow$ **true**
7: $N_{\text{cols-todo}} \leftarrow N_c$
8: **while** $N_{\text{cols-todo}} > 0$ **do**
9: **for** $m = 1, \ldots, N_c$ **do**
10: `options`$[m] \leftarrow \mathfrak{V} \setminus (\boldsymbol{s}_{q\cdot} \cup \boldsymbol{s}_{\cdot m})$
11: $m \leftarrow \arg\min_i |$`options`$[i]|$
12: **if** $|$`options`$[i]| = 0$ **then**
13: `is_valid` $\leftarrow$ **false**
14: **break**
15: $s_{qm} \leftarrow$ RANDOM(`options`$[m]$)
16: $N_{\text{cols-todo}} \leftarrow N_{\text{cols-todo}} - 1$
17: **if** `is_valid` **then**
18: $q \leftarrow q + 1$
19: **else**
20: $\boldsymbol{s}_{q\cdot} \leftarrow$ empty vector of size $1 \times N_c$
21: **return** $\boldsymbol{S}$

We illustrate Algorithm 2 with an example in Figure 10. The figure contains four agent classes, and depicts the construction of the second row of the computation schedule matrix. Each other row in the figure corresponds either to determining the remaining options of computation classes for each cell (Line 10), or to the random selection of an option (Line 15) from the most constrained cell (Line 11).

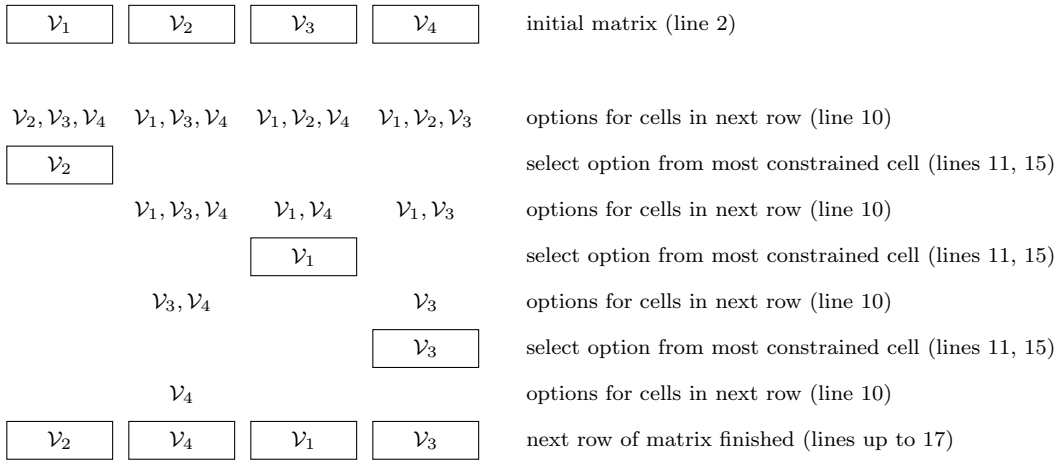


Fig. 10. Example for Algorithm 2 which builds a computation schedule matrix.

Our Algorithm 2 builds a computation schedule row by row, which is valid according to (19) and (20). A partly built, valid computation schedule can always be completed. Therefore, our Algorithm 2 will eventually converge to a valid computation schedule. Due to the randomness of the approach, there is no bound on the number of required iterations until convergence, which could be addressed with a more sophisticated algorithm. In practice, the computation time to build a computation schedule matrix is negligible when compared to the computation time for motion planning.

Computation with multiple prioritizations is similar in concept to computation with a single prioritization. The main difference is that there are N_c different directed coupling graphs $\vec{\mathcal{G}}_{p_q} = (\mathcal{V}, \vec{\mathcal{E}}_{p_q})$, and with it sets of predecessors $\mathcal{V}_{p_q}^{(i\leftarrow)}$, successors $\mathcal{V}_{p_q}^{(i\rightarrow)}$, and neighbors $\mathcal{V}_{p_q}^{(i)}$, defined as

$$\mathcal{V}_{p_q}^{(i\leftarrow)} = \left\{ j \in \mathcal{V} \mid (j \rightarrow i) \in \vec{\mathcal{E}}_{p_q} \right\}, \tag{21}$$

$$\mathcal{V}_{p_q}^{(i\rightarrow)} = \left\{ j \in \mathcal{V} \mid (i \rightarrow j) \in \vec{\mathcal{E}}_{p_q} \right\}, \tag{22}$$

$$\mathcal{V}_{p_q}^{(i)} = \mathcal{V}_{p_q}^{(i\leftarrow)} \cup \mathcal{V}_{p_q}^{(i\rightarrow)}. \tag{23}$$

Each agent performs N_c computations in N_c time slots. An agent in class $\mathcal{V}_l$ finds the sequence $\boldsymbol{s}_{q\cdot}$ corresponding to a time slot m by looking for its agent class in the respective column of the computation schedule matrix. For this computation sequence it must hold

$$s_{qm} = \mathcal{V}_l \,. \tag{24}$$

From this sequence, the agent constructs the directed coupling graph $\vec{\mathcal{G}}_{p_q}$ with (2) and (16). The agent solves its planning problem after receiving the required information from its predecessors $\mathcal{V}_{p_q}^{(i\leftarrow)}$. It then stores the solution cost corresponding to the sequence q as an element in a solution cost array.

After all computation sequences are solved, each agent broadcasts its solution cost array. With this information, each agent can determine the solution quality of every computation sequence. The solution quality $J(k)$ of a

time step k for each computation sequence q is measured by the networked cost: the sum of the costs of all agent planning problems regarding the corresponding computation sequence

$$J_{p_q}(k) = \sum_{i=1}^{N_A} J_{p_q}^{(i)}(k), \tag{25}$$

with the cost $J_{p_q}^{(i)} : \mathbb{N} \to \mathbb{R}$ of an agent i, the sequence q corresponding to a prioritization function p_q with (16), and a time step k. After receiving the cost arrays of all other agents, the value for the solution quality for each computation sequence is identical in each agent. Each agent selects the computation sequence with the highest solution quality, i.e., the minimal networked cost

$$p_{\text{explore}} = \arg\min_{p_q} J_{p_q}(k). \tag{26}$$

In case two prioritizations yield the same cost, the one with the lower index is chosen by the algorithm.

3.4 Initial Prioritization

Our goal of improving the solution quality compared to a given prioritization implies the assumption that an initial valid prioritization according to (5), i.e., a prioritization constituting a strict partial order, is given. Finding an initial valid prioritization is simple, e.g., a prioritization that assigns priorities according to unique vertex numbers is valid. Other prioritization algorithms exist (Kuwata et al. 2007; Scheffe, Dorndorf, et al. 2022; Scheffe, Kahle, et al. 2025). The selection of the initial prioritization, and with Algorithm 1 the initial computation sequence as well, determines which prioritizations can possibly be explored. We identified two objectives for selecting the initial prioritization.

The time-variance of priorities in receding horizon planning has both advantages and disadvantages. An advantage of time-variance is that situations which dictate varying priorities over time to find a solution can be solved, as illustrated by Figure 7. A disadvantage is that the constraints in an agent's planning problem change with the prioritization. In a receding horizon planning algorithm, an agent's control inputs are optimized while adhering to constraints in the future, i.e., the time horizon. A change in the constraints can counteract the predictive nature of a receding horizon planning algorithm, which can affect the solution quality (Scheffe, Kahle, et al. 2025). For example, if an agent must come to a full stop during the time horizon to avoid collision with a higher-priority agent, it might decelerate. However, if the priorities flip and the other agent avoids the collision, deceleration reduces solution quality unnecessarily. Therefore, one objective is to enable the algorithm to retain priorities, and thus avoid it being forced to change priorities. The other objective is to explore many prioritizations to find the best available solution. A poor selection of an initial set of agent classes can limit the exploration of prioritizations to a fraction of the number of distinct prioritizations.

We achieve the first objective by using the computation sequence with the lowest cost according to (25) from a previous time step for the initial prioritization. From this computation sequence, we generate priorities with (16). To guarantee the validity of these priorities according to (5), they must be unique to account for the time-variability of the coupling graph. In the next time step, we orient the undirected coupling graph by prioritizing agents with the retained priorities, thus creating a directed coupling graph. From this directed coupling graph, we construct the initial computation sequence with our Algorithm 1. With this process, it is always possible to maintain the priorities of the previous time step even with a time-variant coupling graph.

We took a step towards the second objective by random construction of the computation schedule matrix in Algorithm 2. However, given a computation sequence, only a subset of computation schedule matrices can be constructed. The question remains: which computation sequences can be explored given an initial sequence? We explore different computation sequences if we build unique computation schedule matrices. Two computation schedule matrices are distinct if the sets of their rows are different, i.e., if one matrix contains a computation

sequence that the other one does not contain. In other words, they are the same if one can be mapped to the other by rearranging its rows. It is easy to see that for $N_c = 1$ and $N_c = 2$, we obtain the optimal computation sequence. There exists only a single computation schedule matrix, so all sequences can be explored. For $N_c = 3$, there exist two unique computation schedule matrices which do not share a single row. With the initial prioritization selected by retaining priorities, we can transition from one computation schedule matrix to another between time steps as long as they share an identical row, i.e., an identical computation sequence. Consequently, retaining the result for an initial prioritization for $N_c = 3$ restricts us to examining only three out of six possible computation sequences. More formally, we can construct a computation schedule graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ with the set of all unique computation schedule matrices as vertices $\mathcal{V}_c$, which are connected by an edge if they share an identical row

$$(i - j) \in \mathcal{E}_c \iff \exists\, n, m \colon \boldsymbol{s}_{n\cdot}^{(i)} = \boldsymbol{s}_{m\cdot}^{(j)} \tag{27}$$

with the row $\boldsymbol{s}_{n\cdot}^{(i)}$ belonging to the computation schedule matrix in vertex i, and the row $\boldsymbol{s}_{m\cdot}^{(j)}$ belonging to the computation schedule matrix in vertex j.

Theorem 4. *Let the initial prioritization be selected by retaining priorities, and let the computation schedule matrix be created with Algorithm 2. For $N_c \geq 4$, we can explore all possible computation sequences of length N_c given enough time.*

Proof. We can prove this theorem by showing that the computation schedule graph $\mathcal{G}_c$ contains a spanning tree. Let an ascending sequence denote a sequence in which the indices of the agent classes are consecutive for consecutive elements, except for the index N_c, which is followed by 1. We prove our theorem by showing that all computation schedule matrices are connected to a computation schedule matrix which contains the ascending sequence $(\mathcal{V}_1, \ldots, \mathcal{V}_{N_c})$. All matrices containing the computation sequence $(\mathcal{V}_1, \ldots, \mathcal{V}_{N_c})$ are connected according to (27). All N_c different ascending sequences fit into one computation schedule matrix by shifting the starting index by one for each row. It follows that if any ascending sequence is part of a computation schedule matrix, the computation sequence $(\mathcal{V}_1, \ldots, \mathcal{V}_{N_c})$ is part of a connected matrix. If the initial sequence does not allow any ascending sequence to be part of the computation schedule matrix, exactly one element of this initial sequence is responsible for making exactly one of the ascending sequences invalid. Put differently: as long as there are two agent classes with consecutive indices in the initial sequence—both of which belong to a single ascending sequence and thus invalidate the same ascending sequence—one ascending sequence must be valid in the corresponding computation schedule matrix. In the case that the initial sequence does not allow any ascending sequence to be part of the computation schedule matrix, we can place any ascending sequence and switch the element that would make the sequence invalid with its following element. This results in a sequence where $N_c - 2$ elements are consecutive in their agent class index. If $N_c \geq 4$, more than two elements of this computation sequence are consecutive in their agent class index. As pointed out above, this matrix is connected to a matrix which contains a valid ascending sequence, which in turn is connected to a matrix containing the sequence $(\mathcal{V}_1, \ldots, \mathcal{V}_{N_c})$. Therefore, a spanning tree of $\mathcal{G}_c$ is rooted in the corresponding vertex. □

3.5 Discussion of Our Approach for Simultaneous Computation with Multiple Prioritizations

The number of prioritizations that our approach considers equals the number of agent classes, which is upper bounded by the number of agents. A commonly referenced upper bound for the number of distinct prioritizations is $N_A!$, which is the number of total orders on the set of agents, i.e., the number of permutations of the set of agents. Consequently, this bound guarantees only a ratio of explored prioritizations to distinct prioritizations that decreases rapidly with the number of agents. However, since only a partial order is required, the number

of distinct prioritizations is related to the connectivity of the coupling graph, which is captured by the vertex degrees $d^{(i)}$ (Definition 8).

Theorem 5. *The number of distinct prioritizations N_{prio} of an undirected coupling graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is upper bounded by*

$$N_{prio} \leq \min \left\{ \prod_{i \in \mathcal{V}} \left(d^{(i)} + 1 \right),\ N_A! \right\}. \tag{28}$$

Proof. A valid prioritization function orients every edge of $\mathcal{G}$ with (2), and the resulting directed coupling graph is acyclic (Section 2.3). Conversely, every acyclic orientation of $\mathcal{G}$ is induced by the prioritization that assigns priorities according to a topological order of that orientation. With Definition 7, N_{prio} thus equals the number of acyclic orientations of $\mathcal{G}$. The first bound is the upper bound on the number of acyclic orientations given in (Manber and Tompa 1984, Theorem 10). The second bound follows since each such topological order is one of the $N_A!$ total orders on $\mathcal{V}$. □

Remark 1. According to Theorem 5, the bound on the number of distinct prioritizations decreases as the coupling graph becomes sparser, i.e., as the vertex degrees decrease. On sparse coupling graphs, our approach therefore explores a larger fraction of the distinct prioritizations than the bound $N_A!$ suggests.

Our approach only explores a fraction of all distinct prioritizations. However, if the networked planning problem changes only slightly from one time step to another, or if the approach is applied repeatedly, our approach can improve the solution quality incrementally. A significant benefit of our approach is that it is general and does not require any domain-specific knowledge.

4 Motion Planning for CAVs

MAMP is a variant of MAPF. The evaluation of the approach presented in this article takes place in the context of MAMP for CAVs. Extending the description of the problem setting in Section 1, we state the following differences between traditional MAPF and our formulation of MAMP for CAVs.

(1) *System dynamics*: In MAPF, the dynamics of robots are usually abstracted in the graph search problem (e.g., Banfi et al. 2020; J. Li, Tinka, et al. 2021). In our MAMP, we explicitly consider the system dynamics in the planning problem by encoding motion in a motion primitive automaton (MPA).
(2) *Objective*: In MAPF, the objective of agents is to plan a path to reach a target vertex in the graph representing the environment without collisions. In our MAMP, the objective of agents is to plan motions to stay close to a reference path without collisions. The reference path is the result of a routing layer and does not consider obstacles.
(3) *Planning time horizon*: In MAPF, the planning time horizon is variable and extends until the agent has reached its target vertex. An agent plans once, and the plan reaches from the agent's start vertex to its target vertex. In our MAMP, the planning time horizon is fixed. This allows for short and predictable computation times. An agent plans at every time step, thereby shifting its planning horizon, an approach known as receding horizon control (RHC).
(4) *Environment representation*: In MAPF, the environment with its obstacles is encoded in a graph, where vertices represent locations at which agents can stop, and edges represent discretized motions of agents. In our MAMP, the environment with its obstacles is represented by a list of polygons. Each agent is required to avoid these polygons when planning motions.

There are works that go beyond traditional MAPF and extend the above-listed categorization; e.g., system dynamics are considered in (Alonso-Mora et al. 2018; Le and Plaku 2018; Luo et al. 2016).

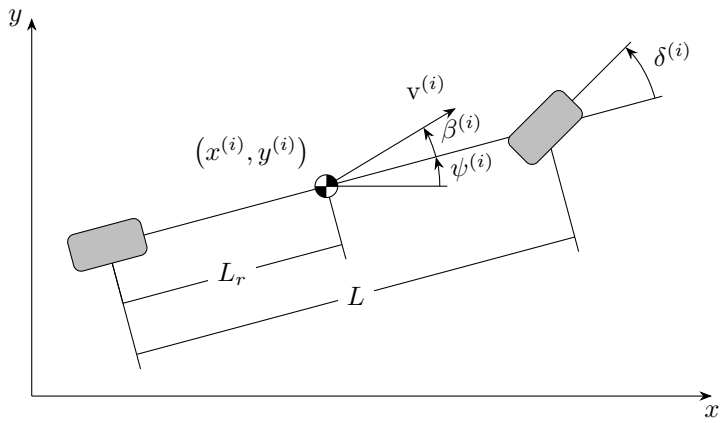


Fig. 11. Kinematic single-track model of a vehicle. Figure redrawn from (Scheffe, Pedrosa, et al. 2023).

We formulate motion planning as an optimal control problem (OCP) in Section 4.1. We present our model for considering kinodynamic constraints in Section 4.2, and we show how we couple agents in Section 4.3. Section 4.4 presents our algorithm to solve the planning problem of each agent, which is based on model predictive control (MPC).

4.1 Motion Planning as an Optimal Control Problem

This section presents the ordinary differential equations describing the vehicle dynamics and our cost function before both are incorporated into an OCP for motion planning. The variables and equations in this section belong to a single agent i.

As introduced in Section 2.1, we refer to the entirety of all agents as an NCS. We control multiple agents in an NCS with a combination of prioritized planning and distributed model predictive control (DMPC) (Richards and How 2007). This combination will be referred to as prioritized distributed model predictive control (Kloock, Scheffe, Botz, et al. 2019; Scheffe, Dorndorf, et al. 2022; Scheffe, Pedrosa, et al. 2024; Scheffe, Xu, et al. 2024). Finding the agents' control inputs is modeled as an OCP. We denote by agent OCPs the OCPs solved by the agents, and by networked OCP the OCP for the entire prioritized NCS.

Figure 11 shows an overview of variables for the nonlinear kinematic single-track model (Rajamani 2006, p. 21). Assuming low velocities, we model no slip on the front and rear wheels, and no forces acting on the vehicle. The resulting equations are

$$\begin{cases} \dot{x}^{(i)}(t) = \mathrm{v}^{(i)}(t) \cdot \cos(\psi^{(i)}(t) + \beta^{(i)}(t)), \\ \dot{y}^{(i)}(t) = \mathrm{v}^{(i)}(t) \cdot \sin(\psi^{(i)}(t) + \beta^{(i)}(t)), \\ \dot{\psi}^{(i)}(t) = \mathrm{v}^{(i)}(t) \cdot \frac{1}{L} \cdot \tan(\delta^{(i)}(t)) \cos(\beta^{(i)}(t)), \\ \dot{\mathrm{v}}^{(i)}(t) = u_{\mathrm{v}}^{(i)}(t), \\ \dot{\delta}^{(i)}(t) = u_{\delta}^{(i)}(t), \end{cases} \tag{29}$$

with

$$\beta^{(i)}(t) = \tan^{-1}\left(\frac{L_r}{L}\tan(\delta^{(i)}(t))\right), \tag{30}$$

where $x^{(i)} \in \mathbb{R}$ and $y^{(i)} \in \mathbb{R}$ describe the position of the center of gravity (CG), $\psi^{(i)} \in [0, 2\pi)$ is the orientation, $\beta^{(i)} \in [-\pi, \pi)$ is the side slip angle, $\delta^{(i)} \in [-\pi, \pi)$ and $u_\delta \in \mathbb{R}$ are the steering angle and its derivative, respectively, $\mathrm{v}^{(i)} \in \mathbb{R}$ and $u_\mathrm{v} \in \mathbb{R}$ are the speed and acceleration of the CG, respectively, L is the wheelbase length and L_r is the length from the rear axle to the CG.

The system dynamics defined in (29) are compactly written as

$$\dot{\boldsymbol{x}}^{(i)}(t) := \frac{d}{dt}\boldsymbol{x}^{(i)}(t) = f\big(\boldsymbol{x}^{(i)}(t), \boldsymbol{u}^{(i)}(t)\big) \tag{31}$$

with the state vector

$$\boldsymbol{x}^{(i)} = \begin{pmatrix} x^{(i)} & y^{(i)} & \psi^{(i)} & \mathrm{v}^{(i)} & \delta^{(i)} \end{pmatrix}^\mathrm{T} \in \mathbb{R}^5, \tag{32}$$

the control input

$$\boldsymbol{u}^{(i)} = \begin{pmatrix} u_\mathrm{v}^{(i)} & u_\delta^{(i)} \end{pmatrix}^\mathrm{T} \in \mathbb{R}^2 \tag{33}$$

and the vector field f defined by (29). Transferring (31) to a discrete-time nonlinear system representation yields

$$\boldsymbol{x}_{k+1}^{(i)} = f_d\big(\boldsymbol{x}_k^{(i)}, \boldsymbol{u}_k^{(i)}\big) \tag{34}$$

with a time step $k \in \mathbb{N}$ and the vector field $f_d \colon \mathbb{R}^5 \times \mathbb{R}^2 \to \mathbb{R}^5$.

We define the cost function for agent i to minimize given a prioritization p in our motion planning problem as

$$J_p^{(i)}(k) = \sum_{l=1}^{N_p} \left(\boldsymbol{x}_{k+l\,|\,k}^{(i)} - \boldsymbol{r}_{k+l\,|\,k}^{(i)}\right)^\mathrm{T} \boldsymbol{Q} \left(\boldsymbol{x}_{k+l\,|\,k}^{(i)} - \boldsymbol{r}_{k+l\,|\,k}^{(i)}\right) \tag{35}$$

with the prediction horizon $N_p \in \mathbb{N}$, a reference trajectory $\boldsymbol{r}_{\cdot\,|\,k}^{(i)} \in \mathbb{R}^5$, and the positive semi-definite, block diagonal matrix

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{I}_2 & \boldsymbol{0}_{2\times 3} \\ \boldsymbol{0}_{3\times 2} & \boldsymbol{0}_3 \end{pmatrix} \in \mathbb{R}^{5\times 5}. \tag{36}$$

The OCP of agent i follows in (37). Since multiple agents are involved in the equation, we will include the agent superscript. We assume a full measurement or estimate of the state $\boldsymbol{x}^{(i)}(k)$ is available at the current time k.

$$\underset{\boldsymbol{u}_{\cdot\,|\,k}^{(i)}}{\text{minimize}} \quad J_p^{(i)}(k) \tag{37a}$$

subject to

$$\boldsymbol{x}_{k+l+1\,|\,k}^{(i)} = f_d\big(\boldsymbol{x}_{k+l\,|\,k}^{(i)}, \boldsymbol{u}_{k+l\,|\,k}^{(i)}\big), \quad l = 0, \dots, N_p - 1, \tag{37b}$$

$$\boldsymbol{x}_{k+l\,|\,k}^{(i)} \in \mathcal{X}, \qquad l = 1, \dots, N_p - 1, \tag{37c}$$

$$\boldsymbol{x}_{k+N_p\,|\,k}^{(i)} \in \mathcal{X}_f, \tag{37d}$$

$$\boldsymbol{x}_{k\,|\,k}^{(i)} = \boldsymbol{x}^{(i)}(k), \tag{37e}$$

$$\boldsymbol{u}_{k+l\,|\,k}^{(i)} \in \mathcal{U}, \qquad l = 0, \dots, N_u - 1, \tag{37f}$$

$$c_\mathrm{c}^{(i\leftarrow j)}\left(\boldsymbol{x}_{k+l\,|\,k}^{(i)}, \boldsymbol{x}_{k+l\,|\,k}^{(j)}\right) \le 0, \qquad l = 1, \dots, N_p, \quad j \in \mathcal{V}_p^{(i\leftarrow)}(k). \tag{37g}$$

Here, $f_d \colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a vector field representing the discrete-time nonlinear system model with $n \in \mathbb{N}$ as the number of states and $m \in \mathbb{N}$ as the number of inputs, $\boldsymbol{u}_{\cdot\,|\,k}^{(i)} = (\boldsymbol{u}_{k\,|\,k}^{(i)}, \boldsymbol{u}_{k+1\,|\,k}^{(i)}, \dots, \boldsymbol{u}_{k+N_u-1\,|\,k}^{(i)})$ is the input

trajectory, $\mathcal{X}$ is the set of feasible states, $\mathcal{X}_f$ is the set of feasible terminal states, and $\mathcal{U}$ is the set of feasible inputs. The coupling constraint $c_c^{(i\leftarrow j)} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ in (37g) enforces networked constraints among agents. The set of predecessors $\mathcal{V}_p^{(i\leftarrow)}(k)$ contains the neighbors of i with higher priority given the prioritization p. An agent i solves its OCP after having received the plans of all predecessors $\mathcal{V}_p^{(i\leftarrow)}(k)$. Afterwards, it communicates its own plan to agents in $\mathcal{V}_p^{(i\rightarrow)}(k)$. Since the planning problem requires the predecessors' plans in (37g), the agents must solve their OCPs in a sequential order. Each agent solves the OCP (37) repeatedly after a time step duration T_s and with updated values for the states and constraints, which establishes the RHC. Only the first input $\boldsymbol{u}_{k\,|\,k}^{(i)}$ of the input trajectory is applied.

4.2 System Modeling with a Motion Primitive Automaton

This section presents how we model the state-continuous system (34) as an MPA (Frazzoli et al. 2005; Scheffe, Pedrosa, et al. 2023). The MPA incorporates the constraints on system dynamics (37b), on control inputs (37f), and on both the steering angle and the speed (37c) and (37d).

From the system dynamics (29), we derive a finite state automaton which we call MPA and define as follows.

Definition 13 (Motion primitive automaton, from (Scheffe, Pedrosa, et al. 2023))**.** An MPA is a 5-tuple $(\mathcal{Q}, \mathcal{S}, \gamma, Q_0, \mathcal{Q}_f)$ composed of:

- $\mathcal{Q}$ is a finite set of automaton states Q;
- $\mathcal{S}$ is a finite set of transitions S, also called motion primitives (MPs);
- $\gamma : \mathcal{Q} \times \mathcal{S} \times \mathbb{N} \to \mathcal{Q}$ is the update function defining the transition from one automaton state to another, dependent on the time step in the horizon;
- $Q_0 \in \mathcal{Q}$ is the initial automaton state;
- $\mathcal{Q}_f \subseteq \mathcal{Q}$ is the set of final automaton states.

An automaton state is characterized by a specific speed v and steering angle δ. An MP is characterized by an input trajectory and a corresponding state trajectory which starts and ends with the speed and steering angle of an automaton state. It has a fixed duration which we choose to be equal to the time step duration T_s. MPs can be concatenated to form longer state trajectories by rotation and translation. Our MPA discretizes both the state space with the update function γ and the time space with a fixed duration T_s for all MPs. This MPA replaces the system representation (34). Note that the dynamics model on which our MPA is based is interchangeable. Its complexity is irrelevant computation-wise for motion planning since MPs are computed offline. The MPA design forces vehicles to come to a complete stop at the end of the prediction horizon, which guarantees that solutions to (37) are recursively feasible; for details see our previous work (Scheffe, Dorndorf, et al. 2022; Scheffe, Pedrosa, et al. 2023).

4.3 Coupling of Vehicles

Although coupling all vehicles guarantees that coupling constraints for all vehicles will be considered, it also prolongs computation time and increases the number of prioritizations compared to coupling fewer vehicles. Therefore, we only couple vehicles that can potentially collide within the horizon N_p. We couple two vehicles at a time step k if their reachable sets intersect at at least one time step within the horizon N_p.

Definition 14 (Reachable set of a time interval, from Scheffe, Xu, et al. (2024))**.** The reachable set $\mathcal{R}_{[t_1,t_2]\,|\,t_0}^{(i)}$ of the states of vehicle i between time t_1 and time t_2 starting from time t_0 is

$$\mathcal{R}_{[t_1,t_2]\,|\,t_0}^{(i)} = \left\{ \bigcup_{t' \in [t_1,t_2]} \int_{t_0}^{t'} f(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}) dt \;\middle|\; \boldsymbol{x}^{(i)}(t_0) \in \mathcal{X}(t_0), \boldsymbol{u}^{(i)} \in \mathcal{U} \right\}.$$

The reachable sets $\mathcal{R}^{(i)}_{[k+l,k+l+1]\,|\,k}$ over the prediction horizon of a vehicle can be computed offline with the MPA presented in Section 4.2. The set of coupling edges is

$$\mathcal{E}(k) = \left\{ (i-j) \;\middle|\; i,j \in \mathcal{V},\ i \neq j,\ \exists\, l \in \{0,\dots,N_p-1\}\colon\ \mathcal{R}^{(i)}_{[k+l,k+l+1]\,|\,k} \cap \mathcal{R}^{(j)}_{[k+l,k+l+1]\,|\,k} \neq \emptyset \right\}. \tag{38}$$

This results in a time-variant coupling graph $\mathcal{G}(k) = (\mathcal{V}, \mathcal{E}(k))$. Because the coupling rule in (38) is deterministic, each agent can compute the coupling graph locally from the current states and reachable-set estimates of all agents. Hence, all agents obtain the same coupling graph and no central entity is required. The required state and reachable-set information can be exchanged directly among agents, i.e., in a distributed manner.

4.4 Planning Using Monte-Carlo Tree Search in a Model Predictive Control Framework

It is computationally hard to find the global optimum to the planning problem (37) due to its nonlinearity and nonconvexity. We substitute the system model (37b) with an MPA based on our previous work (Scheffe, Pedrosa, et al. 2023). Our MPA is general rather than tailored to a specific environment. Therefore, our motion planning is a receding horizon tree search rather than a graph search. Obstacles are included in the state constraints (37c) of our OCP formulation. In the Monte Carlo tree search (MCTS), we represent obstacles and predecessors' trajectories as polygons.

The cost of edges in the tree is given by the cost function (37a). In this article, we use a sampling-based approach based on MCTS to find a path in the tree with low cost. Due to its anytime property, MCTS is a popular approach to complex control problems (Baby and HomChaudhuri 2023; Choudhury et al. 2022; Kurzer et al. 2018), as it allows balancing computation effort and solution quality. In each time step, our MCTS randomly expands a fixed number of vertices in the search tree. The algorithm thus inspects only a part of the complete search tree. After the expansions, the path in the search tree with the lowest cost according to (37a) is selected. The algorithm has no guarantee on optimality, but nearly constant computation time.

Because the MAMP runs online, vehicles require an input at every time step. However, since PP is incomplete (Theorem 1), the MCTS might fail to find a feasible solution. In such cases, vehicles reuse their previous, recursively feasible solution, ensuring safe motion plans (Scheffe, Dorndorf, et al. 2022).

5 Evaluation

We evaluate the presented approach for increasing solution quality by exploring multiple prioritizations in the context of MAMP for CAVs. In Section 5.1, we describe the experiment setup, a scenario on a road network with up to 20 vehicles. In Sections 5.2 and 5.3, we evaluate the quality of the motion plans of the vehicles in the NCS, and the computation time of the NCS, respectively. In Section 5.4, we demonstrate our approach in an experiment in the CPM Lab, an open-source, remotely accessible testbed for CAVs (Kloock, Scheffe, Maczijewski, et al. 2021).

We compare our prioritization algorithm with five algorithms from the literature, which we selected for their suitability for distributed execution and their similarity in computational complexity. Each algorithm is represented by a prioritization function $p\colon \mathcal{V} \to \mathbb{N}$. The first function prioritizes vehicles according to their vertex number (Alrifaee et al. 2016) and is denoted by p_{constant}. The second function prioritizes vehicles randomly at each time step (Bennewitz et al. 2002) and is denoted by p_{random}. The third function prioritizes vehicles with a constraint-based heuristic (Scheffe, Dorndorf, et al. 2022). A higher number of potential collisions with other vehicles results in a higher priority. The function is denoted by $p_{\text{constraint}}$. The fourth function prioritizes vehicles to decrease the number of agent classes, and thus the computation time, via graph coloring (Kuwata et al. 2007; Scheffe, Kahle, et al. 2025). The function is denoted by p_{color}. The fifth function evaluates all distinct prioritizations by solving the networked OCP for all acyclic orientations of the coupling graph. The function then chooses the prioritization which results in the highest solution quality. It is denoted by p^*. In our evaluation, all algorithms above group agents into agent classes for parallel computation as described in Section 3.1.

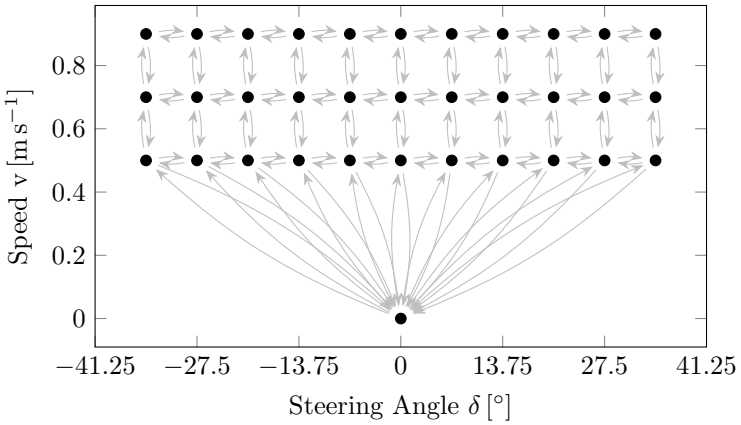


Fig. 12. The MPA used in our experiments.

5.1 Setup

In our experiments, vehicles move on the road network shown in Figure 17. It consists of an urban intersection with eight incoming lanes, a highway, and highway on- and off-ramps. With this variety of road segments, we challenge our motion planner with merging, crossing, and overtaking scenarios.

Our MCTS uses the MPA illustrated in Figure 12, which is based on the kinematic single-track model as presented in (Scheffe, Pedrosa, et al. 2023). The MPA is discretized with four speed levels. The number of expansions during low-level planning for a single vehicle is 6000.

We set the duration for our numerical experiments to 7 s with a time step size of 0.2 s. We evaluate the performance in 50 randomized variants on the road network. The vehicles start at random positions on the road network and follow a randomly selected path which starts and ends at the same point. The reference speeds for the vehicles in the experiments are selected at random from the speed levels of our MPA shown in Figure 12.

During the experiment, the vehicles need to stay within the road boundaries. Lower-prioritized vehicles must avoid collisions with higher-prioritized ones. Therefore, coupling constraints with higher-prioritized vehicles reduce the set of feasible system states $\mathcal{X}$ by the system states the higher-prioritized vehicles occupy during a motion primitive in the MPA (Scheffe, Pedrosa, et al. 2023). Two vehicles in an NCS are coupled whenever their reachable sets overlap (Scheffe, Xu, et al. 2024). As the reachable set is limited in size, and vehicles move along the road network, couplings between vehicles appear and disappear over the course of an experiment. Therefore, the coupling graph is time-variant. We initialize priorities in the very first time step of an experiment with the agents' vertex numbers.

The numerical experiments ran on an off-the-shelf laptop with an Apple M4 Pro 14-Core CPU and 24 GB of RAM using MATLAB 2025b, and the low-level motion planning algorithm implemented in C++. The physical experiment (Section 5.4) ran on ten Intel NUCs (NUC7i5BNK), one for each participating vehicle, with an Intel Core i5 7260U CPU with 2.2 GHz and 16 GB of RAM using MATLAB 2023a, and the low-level motion planning algorithm implemented in MATLAB. The NUCs are connected via Ethernet, and communication is realized by the ROS 2 Toolbox for MATLAB. The open-source algorithms implemented in MATLAB and C++are available online[1].

[1] github.com/pscheffe/p-dmpc

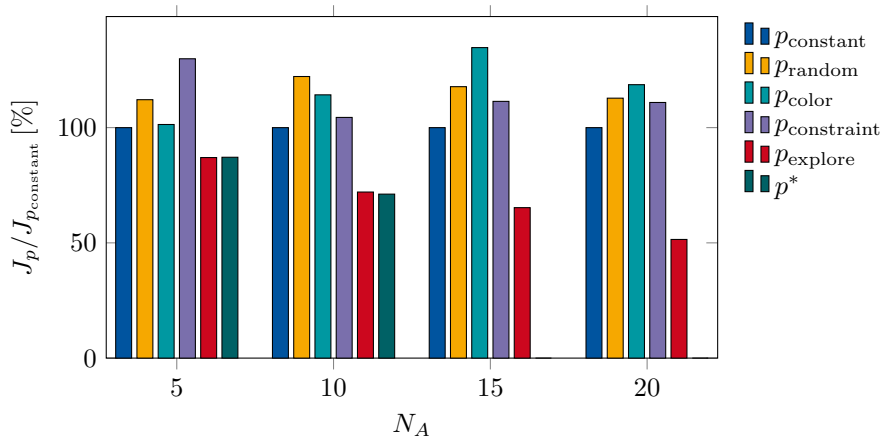


Fig. 13. Comparison of the networked cost among multiple prioritization algorithms, normalized to the cost of p_{constant}. There is no value for p^* for 15 and 20 vehicles as the optimization did not terminate.

5.2 Solution Quality

We measure the solution quality with the networked cost J_p of a prioritization p. The networked cost according to the cost of the planning problem of each vehicle is given as

$$J_p = \sum_{k=0}^{k_{\text{experiment}}} \sum_{i=1}^{N_A} J_p^{(i)}(k), \tag{39}$$

with $k_{\text{experiment}}$ being the number of time steps in the experiment, and $J_p^{(i)}(k)$ given in (35). Figure 13 shows the cost normalized to that of the prioritization p_{constant}.

Our approach, denoted by p_{explore}, outperforms the approaches from the literature in terms of networked cost across all our experiments. For five and ten vehicles, the networked cost of our approach is within 1% of the cost of the optimal prioritization. For 15 vehicles, we reduce the networked cost by more than 53% compared to the baseline prioritization (p_{constant}). There is no value for the cost of the optimal prioritization for 15 and 20 vehicles as the algorithm did not terminate in reasonable computation time.

Surprisingly, Figure 13 indicates that constant priorities can yield better solution quality than time-varying priorities even in dynamic environments. A possible reason is that agents can make full use of the predictive capabilities of MPC: the agent OCPs remain similar between time steps, as opposed to the changing collision avoidance constraints that come with changing priorities. However, there are instances in which a different prioritization yields better solutions or a different prioritization is required to even enable progress at all. In contrast to approaches from the literature, our approach can either remain at a prioritization or switch to another one, determined by the solution quality.

At the start of each scenario, vehicles are distributed on the road network, leaving them with enough room to maneuver. During operation, due to the limited horizon of the planning algorithm, they can end up in standstill situations that cannot be resolved by PP, referred to as deadlock. Table 1 displays the success rate of each prioritization, i.e., the ratio of deadlock-free scenario variations to all scenario variations. For scenarios with five and ten vehicles, all approaches perform similarly and can solve most variants without a deadlock. For scenarios with 15 and 20 vehicles, however, our proposed approach p_{explore} exhibits a much higher success rate, cutting deadlock scenarios by up to 88%.

Table 1. Success rate of evaluated prioritizations, measured as the ratio of the number of deadlock-free scenario variations to all scenario variations. Our proposed approach p_{explore} has the highest success rate compared to other prioritizations. There is no value for p^* for 15 and 20 vehicles as the optimization did not terminate.

N_A	p_{constant}	p_{random}	p_{color}	$p_{\text{constraint}}$	p_{explore}	p^*
5	1.00	1.00	1.00	0.98	1.00	1.00
10	0.94	0.94	0.94	0.98	1.00	1.00
15	0.82	0.88	0.84	0.84	0.98	–
20	0.54	0.60	0.62	0.54	0.86	–

If computation time allows, the solution quality of the random prioritization p_{random} as well as our prioritization p_{explore} could be improved by computing multiple rounds of PP (cf. Section 6.2).

5.3 Computation Time

This section evaluates the computation time of our approach. We first extend the definition of the networked computation time to the case of multiple simultaneously computed prioritizations in Section 5.3.1, and then analyze the resulting computation times in Section 5.3.2.

5.3.1 Computation Time Definition. The computation time when planning with a single prioritization was presented in Section 2.4. Our approach explores one prioritization per agent class. When exploring multiple prioritizations, the networked computation time is at least as high as the maximum of all considered prioritizations. It can even be higher, as computations of multiple prioritizations are interdependent: agents need to finish their computations before the next prioritization can be computed.

This additional relationship needs to be addressed in the computation graph as illustrated in Figure 14 for the coupling graphs of Figure 9b. The edges in black indicate the couplings of agents in the three distinct prioritizations. Since an agent can only compute one solution at a time, the subgraphs of these prioritizations are connected by additional edges in teal accounting for the sequential computation. All edges in black and teal are weighted with the computation time $T^{(i)}$ of an edge's respective starting vertex. The weight of the longest path in this graph corresponds to the computation time of the NCS.

5.3.2 Computation Time Analysis. For five and ten vehicles, the maximum computation time of our approach is considerably lower than that of the optimal prioritization, while achieving a similar solution quality. For a larger number of vehicles, planning with optimal prioritization is computationally too expensive, as the number of distinct prioritizations grows rapidly with the number of vehicles. As shown in Figure 15, our approach scales well with the number of vehicles and only slightly increases the computation time compared to the baseline prioritization. In particular, the median computation time is only slightly increased.

The anytime MCTS algorithm we use for planning has a fixed limit of 6000 expansions. From the numerical experiment data, we get an average time per expansion of 3 µs, which results in an expected computation time of 18 ms. If the computation times of all agents were exactly equal, the networked computation time in Figure 15 would scale with the number of agent classes in Figure 16. Looking at the example of p_{constant} with 20 vehicles, the maximum number of agent classes is 11, from which we expect a networked computation time of 198 ms. The actual maximum networked computation time is only 69 ms. This discrepancy is due to dissimilar computation times among agents.

Dissimilarities in the computation time are due to the fact that in some instances of planning, the search tree has been fully explored with fewer expansions than the limit of 6000. A primary example of such a case is when

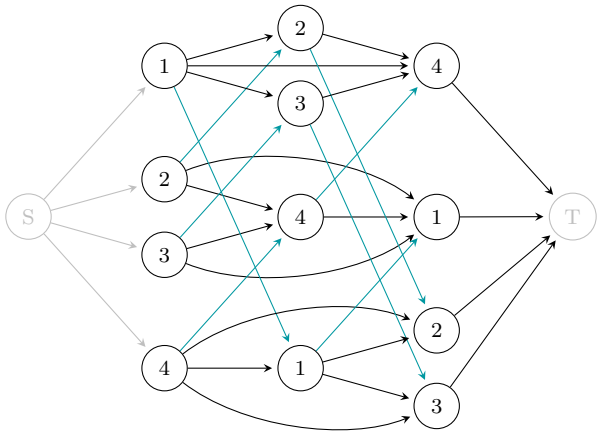


Fig. 14. Computation graph corresponding to the coupling graphs in Figure 9b; vertices are arranged in rows and columns of the computation schedule matrix. Edges in black between agents indicate coupling. Edges in teal are given in addition to coupling, since an agent can only work on one prioritization at a time. Edges and nodes in light gray are for determining the networked computation time via a longest path algorithm.

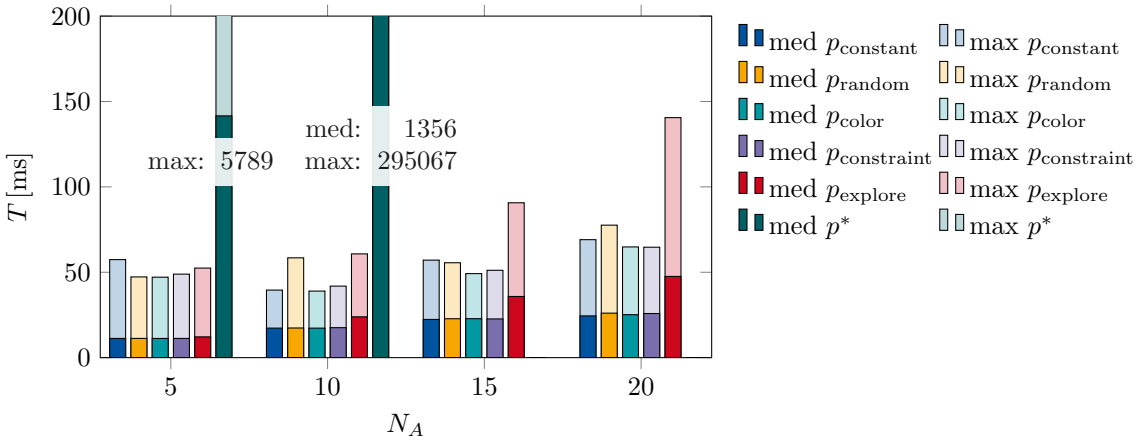


Fig. 15. Comparison of the networked computation time among multiple prioritization algorithms. There is no value for p^* for 15 and 20 vehicles as the optimization did not terminate.

a vehicle is blocked by others. This example is particularly noticeable when the number of agent classes is high, as vehicles are in close proximity to one another. Interestingly, this effect decreases the maximum networked computation time compared to the expected networked computation time from the number of agent classes for all single-shot prioritizations, but not for the proposed prioritization p_{explore}. For the explanation of this phenomenon we need to look at the computation graph $\vec{\mathcal{G}}'$ displayed in Figure 14. Since we weight all outgoing edges with the computation time associated with solving the vertex's planning problem, the networked computation time is determined by the longest path in $\vec{\mathcal{G}}'$. The edges in teal chain the N_c computations of each agent, so the

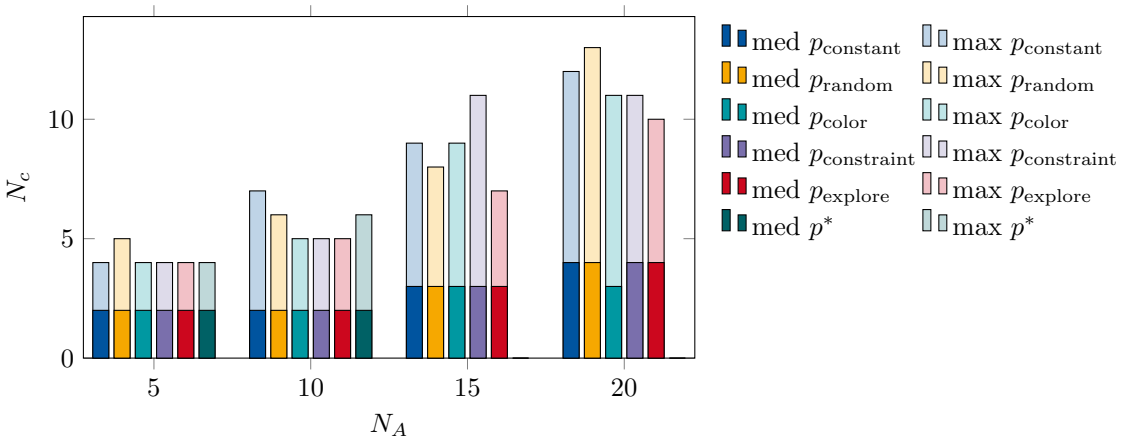


Fig. 16. Comparison of the number of agent classes among multiple prioritization algorithms. The value of p^* corresponds to the number of agent classes of the selected prioritization. There is no value for p^* for 15 and 20 vehicles as the optimization did not terminate.

computations of a single agent already form a path in $\vec{\mathcal{G}}'$. The weight of this path is the sum of that agent's computation times across all prioritizations. It is therefore sufficient that a single agent requires the full expected computation time in each of its computations for the networked computation time to be near its expected value.

Remark 2. If the computation time of all agents for each planning problem of every prioritization is exactly equal, there is no overhead in computation time for our approach.

In its present form, our framework lays the foundations for guaranteed real-time planning. Even though our approach finishes before the sample time T_s in our experiments, it does not account for the worst-case execution time that can occur when the number of agent classes equals the number of agents. Two natural approaches for real-time guarantees suggest themselves. One, limiting the number of agent classes to a desired value by enabling additional parallel computations while ensuring safety by restricting the solution space (Scheffe, Xu, et al. 2024). Two, limiting the number of expansions and thus the expected computation time per agent depending on the number of agent classes. Both approaches are only possible since coupling and prioritization are performed before planning in the framework used in this article.

5.4 Experiment

We conducted an experiment with ten vehicles in the CPM Lab (Kloock, Scheffe, Maczijewski, et al. 2021). The motion planning is executed on ten PCs connected via Ethernet, with each running the motion planning algorithm in MATLAB. A snapshot of the experiment is shown in Figure 17; a video is available online[2]. To achieve real-time capability, we combine our approach with our previous work of limiting the number of agent classes (Scheffe, Xu, et al. 2024). We chose a limit of $N_c = 2$.

In our experiment, we achieve a networked computation time with a median of 72 ms and a maximum of 111 ms across all time steps. With our time step duration of $T_s = 200$ ms, this networked computation time allows enough leeway for other subordinate computations and for communication.

Although communication time is excluded from the main algorithmic analysis, we report it here as an implementation-specific observation. The communication time between the motion planners in the experiment

[2] youtu.be/Mb59zQ3j3s0

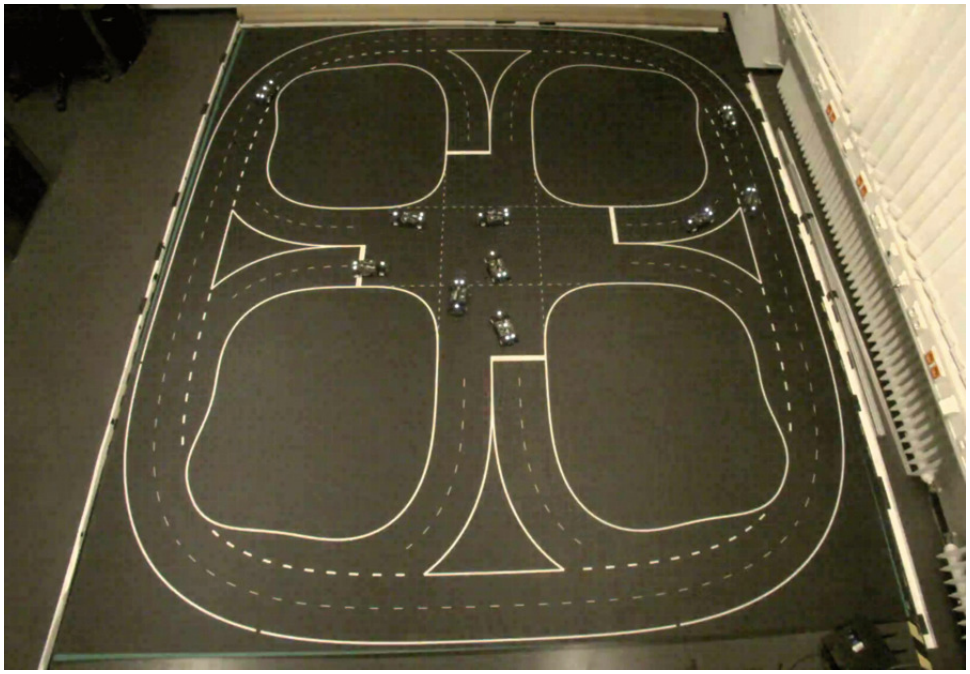


Fig. 17. Experiment in the CPM Lab with ten vehicles.

has a median of 5 ms and a 99th percentile of 25 ms. This communication time is surprisingly high considering the PCs are connected via Ethernet. In a simulation study of three scenario variations with 20 PCs connected via Ethernet, we observed a median communication time of 4 ms, and a maximum of 33 ms. With our simultaneous approach, the communication time increased to 11 ms in the median, and to 157 ms in the maximum.

These measurements are reported for completeness and to characterize the prototype implementation; they are not used to support the main comparison of this article, which focuses on planning performance under the scope assumptions stated in the introduction.

In our approach, the number of prioritizations explored is equal to the number of agent classes. For each prioritization, communication is required. Consequently, this results in an increase in the total communication effort by a factor of the number of agent classes. However, the plans regarding each prioritization can be communicated in parallel, resulting in the same number of $N_c - 1$ sequential communications. Additionally, agents communicate the cost of their solution to determine the solution quality of each prioritization. In practice, the exchanged data is small, as it only consists of the trajectory over the prediction horizon and the cost of the planning problem. Our implementation with MATLAB and the ROS 2 toolbox is not intended for efficiency in communication and seems to struggle with this higher number of messages, resulting in high latencies.

6 Conclusion

This section summarizes the contributions of this article and outlines directions for future work.

6.1 Summary

We presented an approach to utilize idle time in prioritized planning to simultaneously compute with multiple prioritizations. With our approach, we can achieve high solution quality with significantly less computational effort than that of sequential computation of the considered prioritizations. Additionally, our approach is general, as it does not rely on domain-specific heuristics.

Our approach can simultaneously examine as many prioritizations as there are agent classes. The higher the number of agent classes, the more prioritizations are examined. However, the number of distinct prioritizations increases more quickly with the number of agent classes. Therefore, our approach is more likely to find the optimal prioritization with a lower number of agent classes. Additionally, a lower number of agent classes results in a smaller increase in computation time. These benefits make the approach well-suited to be combined with limiting the number of agent classes for real-time computation (Scheffe, Xu, et al. 2024).

6.2 Future Work

Our approach is general and can thus be transferred to prioritized computations in other applications, where communication is required and planning time outweighs communication time. It needs to be analyzed how well the approach transfers, especially regarding the communication effort and the computation time.

Since the planning problems of multiple prioritizations are solved simultaneously, the communication effort compared to computing with a single prioritization is multiplied by the number of agent classes. In our experiments, agents communicated via LAN. It seems likely that the communication effort is decreased if computations run on the same machine, and increased if agents communicate via WLAN. In any case, the gain in solution quality must outweigh the increase in communication effort compared to computing with a single prioritization. Further studies are required to verify that communication can be time-efficient enough to warrant the distributed execution of our approach for a large number of agents.

We presented solving an OCP as a control method to find control inputs and to generate a motion plan. However, our approach works well with any control method in which agents have similar computation times. Therefore, the approach could also be evaluated with different control methods.

We presented simulations with up to 20 agents and eight agent classes, and an experiment with ten agents and two agent classes. Conducting experiments with a larger number of agents and agent classes would be interesting to study the convergence to the optimal prioritization. An application with a convex OCP might be suitable to still achieve real-time capable computations.

Our algorithm to build a computation schedule matrix selects prioritizations randomly from all valid prioritizations. The algorithm might improve if it instead selects prioritizations based on knowledge of the solution quality of prioritizations it has solved before.

We aimed to increase the solution quality without increasing the computation time by utilizing unused computation resources in distributed PP. Thus, we construct a Latin square to explore N_c prioritizations in N_c sequential computations. However, the approach can be generalized to a desired number of sequential computations or a desired number of explored prioritizations N_{explored}, making the computation schedule matrix in a Latin rectangle. Then, the number of explored prioritizations can be selected according to the available computation time.

Our approach for simultaneous computation of multiple prioritizations is intended for PP where the coupling graph can be determined before planning. With extensions to limit the number of agent classes (Scheffe, Xu, et al. 2024), this kind of PP can realize real-time control. When constructing a coupling graph a priori is impossible, PP with ad-hoc construction of a coupling graph can be used (e.g., Chan et al. 2023; Ma, Harabor, et al. 2019). Adapting our algorithm for this setting could increase the utilization of available computation resources.

Acknowledgments

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) within the Priority Program SPP 1835 "Cooperative Interacting Automobiles" (grant number: KO 1430/17-1).

References

M. S. Almhanna, F. S. Al-Turaihi, and T. A. Murshedi. Oct. 1, 2023. "Reducing Waiting and Idle Time for a Group of Jobs in the Grid Computing." *Bulletin of Electrical Engineering and Informatics*, 12, 5, (Oct. 1, 2023), 3115–3123. doi:10.11591/eei.v12i5.4729.

J. Alonso-Mora, P. Beardsley, and R. Siegwart. Apr. 2018. "Cooperative Collision Avoidance for Nonholonomic Robots." *IEEE Transactions on Robotics*, 34, 2, (Apr. 2018), 404–420. doi:10.1109/TRO.2018.2793890.

B. Alrifaee, F.-J. Heßeler, and D. Abel. 2016. "Coordinated Non-Cooperative Distributed Model Predictive Control for Decoupled Systems Using Graphs." *IFAC-PapersOnLine*, 49, 22, 216–221. doi:10.1016/j.ifacol.2016.10.399.

K. Azarm and G. Schmidt. Apr. 1997. "Conflict-Free Motion of Multiple Mobile Robots Based on Decentralized Motion Planning and Negotiation." In: *Proceedings of International Conference on Robotics and Automation*. Proceedings of International Conference on Robotics and Automation. Vol. 4. (Apr. 1997), 3526–3533 vol.4. doi:10.1109/ROBOT.1997.606881.

T. V. Baby and B. HomChaudhuri. May 2023. "Monte Carlo Tree Search Based Trajectory Generation for Automated Vehicles in Interactive Traffic Environments." In: *2023 American Control Conference (ACC)*. 2023 American Control Conference (ACC). (May 2023), 4431–4436. doi:10.23919/ACC55779.2023.10156530.

J. Banfi, V. Shree, and M. Campbell. Sept. 28, 2020. "Planning High-Level Paths in Hostile, Dynamic, and Uncertain Environments." *Journal of Artificial Intelligence Research*, 69, (Sept. 28, 2020), 297–342. doi:10.1613/jair.1.12077.

J. S. Baras, X. Tan, and P. Hovareshti. Dec. 2003. "Decentralized Control of Autonomous Vehicles." In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. 42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475). Vol. 2. (Dec. 2003), 1532–1537 Vol.2. doi:10.1109/CDC.2003.1272829.

M. Barer, G. Sharon, R. Stern, and A. Felner. Sept. 1, 2021. "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem." *Proceedings of the International Symposium on Combinatorial Search*, 5, 1, (Sept. 1, 2021), 19–27. doi:10.1609/socs.v5i1.18315.

M. Bennewitz, W. Burgard, and S. Thrun. Nov. 30, 2002. "Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots." *Robotics and Autonomous Systems*. Ninth International Symposium on Intelligent Robotic Systems 41, 2, (Nov. 30, 2002), 89–99. doi:10.1016/S0921-8890(02)00256-7.

Ö. N. Çam and H. K. Sezen. Mar. 14, 2020. "The Formulation of a Linear Programming Model for the Vehicle Routing Problem in Order to Minimize Idle Time." *Decision Making: Applications in Management and Engineering*, 3, 1, (Mar. 14, 2020), 22–29, 1, (Mar. 14, 2020). doi:10.31181/dmame2003132h.

M. Čáp, P. Novák, A. Kleiner, and M. Selecký. July 2015. "Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots." *IEEE Transactions on Automation Science and Engineering*, 12, 3, (July 2015), 835–849. doi:10.1109/TASE.2015.2445780.

B. Chalaki and A. A. Malikopoulos. 2022. "A Priority-Aware Replanning and Resequencing Framework for Coordination of Connected and Automated Vehicles." *IEEE Control Systems Letters*, 6, 1772–1777. doi:10.1109/LCSYS.2021.3133416.

S.-H. Chan, R. Stern, A. Felner, and S. Koenig. July 2, 2023. "Greedy Priority-Based Search for Suboptimal Multi-Agent Path Finding." *Proceedings of the International Symposium on Combinatorial Search*, 16, 1, (July 2, 2023), 11–19, 1, (July 2, 2023). doi:10.1609/socs.v16i1.27278.

D. Chen, M. R. Hajidavalloo, Z. Li, K. Chen, Y. Wang, L. Jiang, and Y. Wang. Nov. 2023. "Deep Multi-Agent Reinforcement Learning for Highway On-Ramp Merging in Mixed Traffic." *IEEE Transactions on Intelligent Transportation Systems*, 24, 11, (Nov. 2023), 11623–11638. doi:10.1109/TITS.2023.3285442.

S. Choudhury, J. K. Gupta, P. Morales, and M. J. Kochenderfer. Mar. 10, 2022. "Scalable Online Planning for Multi-Agent MDPs." *Journal of Artificial Intelligence Research*, 73, (Mar. 10, 2022), 821–846. doi:10.1613/jair.1.13261.

M. Erdmann and T. Lozano-Pérez. Nov. 1, 1987. "On Multiple Moving Objects." *Algorithmica*, 2, 1, (Nov. 1, 1987), 477. doi:10.1007/BF01840371.

E. Frazzoli, M. A. Dahleh, and E. Feron. Dec. 2005. "Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries." *IEEE Transactions on Robotics*, 21, 6, (Dec. 2005), 1077–1091. doi:10.1109/TRO.2005.852260.

T. Guo and J. Yu. Oct. 31, 2024. "Expected 1.x Makespan-Optimal Multi-Agent Path Finding on Grid Graphs in Low Polynomial Time." *Journal of Artificial Intelligence Research*, 81, (Oct. 31, 2024), 443–479. doi:10.1613/jair.1.16299.

M. Kloock, L. Kragl, J. Maczijewski, B. Alrifaee, and S. Kowalewski. 2019. "Distributed Model Predictive Pose Control of Multiple Nonholonomic Vehicles." In: *IEEE Intelligent Vehicles Symposium (IV)*, 1620–1625. doi:10.1109/IVS.2019.8813980.

M. Kloock, P. Scheffe, L. Botz, J. Maczijewski, B. Alrifaee, and S. Kowalewski. 2019. "Networked Model Predictive Vehicle Race Control." In: *IEEE Intelligent Transportation Systems Conference (ITSC)*, 1552–1557. doi:10.1109/ITSC.2019.8917222.

M. Kloock, P. Scheffe, J. Maczijewski, A. Kampmann, A. Mokhtarian, S. Kowalewski, and B. Alrifaee. 2021. "Cyber-Physical Mobility Lab: An Open-Source Platform for Networked and Autonomous Vehicles." In: *European Control Conference (ECC)*. European Control Conference (ECC), 1937–1944. doi:10.23919/ECC54610.2021.9654986.

B. Korte and J. Vygen. 2018. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Vol. 21. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN: 978-3-662-56038-9. doi:10.1007/978-3-662-56039-6.

K. Kurzer, C. Zhou, and J. Marius Zöllner. June 2018. "Decentralized Cooperative Planning for Automated Vehicles with Hierarchical Monte Carlo Tree Search." In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018 IEEE Intelligent Vehicles Symposium (IV). (June 2018), 529–536. doi:10.1109/IVS.2018.8500712.

Y. Kuwata, A. Richards, T. Schouwenaars, and J. P. How. July 2007. "Distributed Robust Receding Horizon Control for Multivehicle Guidance." *IEEE Transactions on Control Systems Technology*, 15, 4, (July 2007), 627–641. doi:10.1109/TCST.2007.899152.

D. Le and E. Plaku. Oct. 31, 2018. "Cooperative, Dynamics-based, and Abstraction-Guided Multi-robot Motion Planning." *Journal of Artificial Intelligence Research*, 63, (Oct. 31, 2018), 361–390. doi:10.1613/jair.1.11244.

J. Li, T. A. Hoang, E. Lin, H. L. Vu, and S. Koenig. June 26, 2023. "Intersection Coordination with Priority-Based Search for Autonomous Vehicles." *Proceedings of the AAAI Conference on Artificial Intelligence*, 37, 10, (June 26, 2023), 11578–11585, 10, (June 26, 2023). doi:10.1609/aaai.v37i10.26368.

J. Li, W. Ruml, and S. Koenig. May 18, 2021. "EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding." *Proceedings of the AAAI Conference on Artificial Intelligence*, 35, 14, (May 18, 2021), 12353–12362. doi:10.1609/aaai.v35i14.17466.

J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig. 2021. "Lifelong Multi-Agent Path Finding in Large-Scale Warehouses." *Proceedings of the AAAI Conference on Artificial Intelligence*, 35, 13, 11272–11281.

Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh. Aug. 2018. "5G URLLC: Design Challenges and System Concepts." In: *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*. 2018 15th International Symposium on Wireless Communication Systems (ISWCS). IEEE, Lisbon, (Aug. 2018), 1–6. ISBN: 978-1-5386-5005-9. doi:10.1109/ISWCS.2018.8491078.

W. Luo, N. Chakraborty, and K. Sycara. July 2016. "Distributed Dynamic Priority Assignment and Motion Planning for Multiple Mobile Robots with Kinodynamic Constraints." In: *2016 American Control Conference (ACC)*. 2016 American Control Conference (ACC). IEEE, Boston, MA, USA, (July 2016), 148–154. ISBN: 978-1-4673-8682-1. doi:10.1109/ACC.2016.7524907.

H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig. July 17, 2019. "Searching with Consistent Prioritization for Multi-Agent Path Finding." *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, (July 17, 2019), 7643–7650. doi:10.1609/aaai.v33i01.33017643.

H. Ma, J. Li, T. K. S. Kumar, and S. Koenig. May 8, 2017. "Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks." In: *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS '17). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, (May 8, 2017), 837–845.

H.-Y. Mak, Y. Rong, and J. Zhang. May 2014. "Sequencing Appointments for Service Systems Using Inventory Approximations." *Manufacturing & Service Operations Management*, 16, 2, (May 2014), 251–262. doi:10.1287/msom.2013.0470.

U. Manber and M. Tompa. Feb. 1984. "The Effect of Number of Hamiltonian Paths on the Complexity of a Vertex-Coloring Problem." *SIAM Journal on Computing*, 13, 1, (Feb. 1984), 109–115. doi:10.1137/0213008.

B. D. McKay and I. M. Wanless. Oct. 2005. "On the Number of Latin Squares." *Annals of Combinatorics*, 9, 3, (Oct. 2005), 335–344. doi:10.1007/s00026-005-0261-7.

J. Morag, Y. Zhang, D. Koyfman, Z. Chen, A. Felner, D. Harabor, and R. Stern. June 1, 2024. "Prioritised Planning with Guarantees." *Proceedings of the International Symposium on Combinatorial Search*, 17, (June 1, 2024), 82–90. doi:10.1609/socs.v17i1.31545.

Z. Pan, R. Wang, Q. Bi, X. Zhang, and J. Yu. Aug. 27, 2024. "RH-ECBS: Enhanced Conflict-Based Search for MRPP with Region Heuristics." *Robotica*, (Aug. 27, 2024), 1–11. doi:10.1017/S0263574724000894.

R. Rajamani. 2006. *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer Science, New York. 470 pp. ISBN: 978-0-387-26396-0.

A. Richards and J. P. How. Sept. 1, 2007. "Robust Distributed Model Predictive Control." *International Journal of Control*, 80, 9, (Sept. 1, 2007), 1517–1531. doi:10.1080/00207170701491070.

P. Scheffe, G. Dorndorf, and B. Alrifaee. 2022. "Increasing Feasibility with Dynamic Priority Assignment in Distributed Trajectory Planning for Road Vehicles." In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE International Conference on Intelligent Transportation Systems (ITSC), 3873–3879. doi:10.1109/ITSC55140.2022.9922028.

P. Scheffe, J. Kahle, and B. Alrifaee. Jan. 18, 2025. *Graph Coloring to Reduce Computation Time in Prioritized Planning*. (Jan. 18, 2025). arXiv: 2501.10812 [cs]. Pre-published.

P. Scheffe, M. V. A. Pedrosa, K. Flaßkamp, and B. Alrifaee. 2024. "Prioritized Trajectory Planning for Networked Vehicles Using Motion Primitives." In: *Cooperatively Interacting Vehicles: Methods and Effects of Automated Cooperation in Traffic*. Ed. by C. Stiller, M. Althoff, C. Burger, B. Deml, L. Eckstein, and F. Flemisch. Springer International Publishing, Cham, 253–275. ISBN: 978-3-031-60494-2. doi:10.1007/978-3-031-60494-2_9.

P. Scheffe, M. V. A. Pedrosa, K. Flaßkamp, and B. Alrifaee. 2023. "Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning." *IEEE Transactions on Control Systems Technology*, 31, 3, 1092–1105. doi:10.1109/TCST.2022.3214718.

P. Scheffe, J. Xu, and B. Alrifaee. June 25, 2024. "Limiting Computation Levels in Prioritized Trajectory Planning with Safety Guarantees." In: *2024 European Control Conference (ECC)*. 2024 European Control Conference (ECC). IEEE, Stockholm, Sweden, (June 25, 2024), 297–304. ISBN: 978-3-907144-10-7. doi:10.23919/ECC64448.2024.10591179.

T. Shahar, S. Shekhar, D. Atzmon, A. Saffidine, B. Juba, and R. Stern. Mar. 7, 2021. "Safe Multi-Agent Pathfinding with Time Uncertainty." *Journal of Artificial Intelligence Research*, 70, (Mar. 7, 2021), 923–954. doi:10.1613/jair.1.12397.

G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Feb. 1, 2015. "Conflict-Based Search for Optimal Multi-Agent Pathfinding." *Artificial Intelligence*, 219, (Feb. 1, 2015), 40–66. doi:10.1016/j.artint.2014.11.006.

D. Silver. 2005. "Cooperative Pathfinding." *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 1, 1, 117–122, 1. doi:10.1609/aiide.v1i1.18726.

J. P. van den Berg and M. H. Overmars. Aug. 2005. "Prioritized Motion Planning for Multiple Robots." In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. (Aug. 2005), 430–435. doi:10.1109/IROS.2005.1545306.

W. Wu, S. Bhattacharya, and A. Prorok. May 2020. "Multi-Robot Path Deconfliction through Prioritization by Path Prospects." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, Paris, France, (May 2020), 9809–9815. ISBN: 978-1-7281-7395-5. doi:10.1109/ICRA40945.2020.9196813.

N. Yao and F. Zhang. Aug. 2018. "Resolving Contentions for Intelligent Traffic Intersections Using Optimal Priority Assignment and Model Predictive Control." In: *2018 IEEE Conference on Control Technology and Applications (CCTA)*. 2018 IEEE Conference on Control Technology and Applications (CCTA). IEEE, Copenhagen, (Aug. 2018), 632–637. ISBN: 978-1-5386-7698-1. doi:10.1109/CCTA.2018.8511561.

J. Yu. Mar. 2020. "Average Case Constant Factor Time and Distance Optimal Multi-Robot Path Planning in Well-Connected Environments." *Autonomous Robots*, 44, 3–4, (Mar. 2020), 469–483. doi:10.1007/s10514-019-09858-z.

J. Yu. Jan. 2016. "Intractability of Optimal Multirobot Path Planning on Planar Graphs." *IEEE Robotics and Automation Letters*, 1, 1, (Jan. 2016), 33–40. doi:10.1109/LRA.2015.2503143.

J. Yu and S. LaValle. June 29, 2013. "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs." *Proceedings of the AAAI Conference on Artificial Intelligence*, 27, 1, (June 29, 2013), 1443–1449, 1, (June 29, 2013). doi:10.1609/aaai.v27i1.8541.

S. Zhang, J. Li, T. Huang, and S. Koenig. 2022. "Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding." *Proceedings of the Symposium on Combinatorial Search*, 15, 1, 208–216.

A Supplementary Materials

Code github.com/pscheffe/p-dmpc

Video youtu.be/Mb59zQ3j3s0

Received 27 October 2025; accepted 02 August 2026