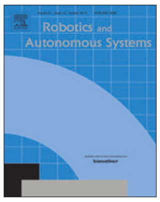
Robotics and Autonomous Systems 150 (2022) 103997

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



CL-MAPF: Multi-Agent Path Finding for Car-Like robots with kinematic and spatiotemporal constraints

Licheng Wen [a], Yong Liu [a,b,*], Hongliang Li [a]

[a] College of Control Science and Engineering, Zhejiang University, Hangzhou, China
[b] Huzhou Institute of Zhejiang University, Huzhou, China

A R T I C L E I N F O

Article history:
Received 26 May 2021
Received in revised form 21 December 2021
Accepted 22 December 2021
Available online 30 December 2021

Keywords:
Multi-agent systems
Path planning
Mobile robots

A B S T R A C T

Multi-Agent Path Finding has been widely studied in the past few years due to its broad application in the field of robotics and AI. However, previous solvers rely on several simplifying assumptions. This limits their applicability in numerous real-world domains that adopt nonholonomic car-like agents rather than holonomic ones. In this paper, we give a mathematical formalization of the Multi-Agent Path Finding for Car-Like robots (CL-MAPF) problem. We propose a novel hierarchical search-based solver called Car-Like Conflict-Based Search to address this problem. It applies a body conflict tree to address collisions considering the shapes of the agents. We introduce a new algorithm called Spatiotemporal Hybrid-State A* as the single-agent planner to generate agents' paths satisfying both kinematic and spatiotemporal constraints. We also present a sequential planning version of our method, sacrificing a small amount of solution quality to achieve a significant reduction in runtime. We compare our method with two baseline algorithms on a dedicated benchmark and validate it in real-world scenarios. The experiment results show that the planning success rate of both baseline algorithms is below 50% for all six scenarios, while our algorithm maintains that of over 98%. It also gives clear evidence that our algorithm scales well to 100 agents in 300 m $\times$ 300 m scenario and is able to produce solutions that can be directly applied to Ackermann-steering robots in the real world. The benchmark and source code are released in https://github.com/APRIL-ZJU/CL-CBS. The video of the experiments can be found on YouTube.

1. Introduction

Multi-Agent Path Finding, also known as MAPF, is a crucial planning problem in the study of multi-robot systems. Each agent is required to move from an initial starting place to a specified goal and avoid collisions with each other. Due to its broad applications in AI and robotics community, research on MAPF has been flourishing in the past few years.

MAPF is known to be an NP-hard problem [1]. Famed approaches to solve this problem can be classified into reduction-based methods [2–4], A*-based methods [5–8], prioritized methods [9,10] and dedicated search-based methods [11–13]. Some researches also take partial kinematic constraints into consideration [14–17].

MAPF can be applied to several contemporary scenarios including self-driving cars, autonomous straddle carriers [18], warehouse robots [19], unmanned surface vehicles [20] and office robots [21]. These industrial and service robots are generally non-holonomic and designed as car-like vehicles in practice. However, almost all the above methods are based on assumptions that agents are modeled as disks and are capable of rotating in place. These solvers also adopt discrete 4-neighbor grids as their search space.

The car-like robots, referred to agents based on the Ackermann-steering model, are in nature with rectangle shapes and have minimum turning radii. Original MAPF solvers can be applied by reducing the grid-graph resolution and adopting dedicated controllers to track generated paths. However, this may generate coarser solutions and degrade their practical applicability since the controllers cannot track paths precisely, especially those with sharp turns. Also, when the planning problem extends to the continuous workspace, the piecewise-linear path generated by the standard A* is not guaranteed to be executable by a non-holonomic agent. This causes these methods, which are optimal for the original MAPF problem, to be unsolvable for the CL-MAPF problem. These solvers also apply various types of conflicts, including vertex conflicts and edge conflicts, to avoid collisions between moving agents [22]. Nevertheless, the types of conflicts adopted in different situations depending on their

* Corresponding author at: College of Control Science and Engineering, Zhejiang University, Hangzhou, China.
E-mail addresses: wenlc@zju.edu.cn (L. Wen), yongliu@iipc.zju.edu.cn (Y. Liu), lihongliang_zju@zju.edu.cn (H. Li).

https://doi.org/10.1016/j.robot.2021.103997
0921-8890/© 2021 Elsevier B.V. All rights reserved.

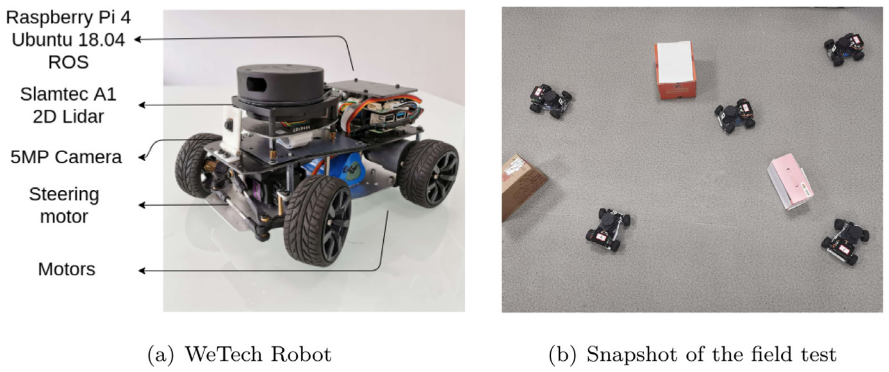


Fig. 1. Our proposed method tested on seven Ackermann-steering robots produced by WeTech.

specific environments, and they cannot represent all the collision scenarios.

To address these concerns, it is essential to formalize the MAPF problem for car-like robots. We propose a novel hierarchical search-based solver called Car-Like Conflict-Based Search (CL-CBS) to settle this problem. It gains the advantage of short computation time using a binary conflict search tree while possessing the ability to plan kinematic-feasible paths for a large number of car-like robots in a continuous workspace. Our main contributions are summarized as follows:

- We present a search-based CL-MAPF solver, which gains the advantage of short computation time using a binary conflict search tree while possessing the ability to plan kinematic-feasible paths in a continuous workspace for a large number of car-like robots.
- We also introduce a sequential version of our original method, which significantly reduces search time at the expense of a marginal loss in solution quality.
- We conduct experiments in both simulated and physical environments. They demonstrate our method can scale well to large amount of agents and produce solutions directly applied to car-like robots in real-world scenes.

2. Related works

MAPF problem has been widely studied in the robotic and AI community. Some methods from the early years are reduction to other well-studied combinatorial problems [2–4]. Recently, several solvers using search techniques have been proposed to solve this problem. Naive applications of such search algorithms are variants of A*. M*[5] expands search nodes to all possibilities when conflict occurs. OD-recursive-M* (ODrM*) [7] adapts the concept of Operator Decomposition [6] to keep the branching factor small. The Safe Interval Path Planning (SIPP) [8] runs an A* search in a graph where each node represents a pair of vertexes in the workspace and a safe time interval. Another well-known branch of MAPF solvers nowadays is based on a two-level optimal solver called Conflict-Based Search (CBS) [11], which conducts a combination search with the high-level binary search tree and the low-level space–time A*. ICBS [12] and CBSH [13] improves CBS further by classifying conflicts and resolving cardinal conflicts first. CBSH [13] improves it further by aggregating cardinal conflicts and computing admissible heuristics to guide the high-level search. Finally, the prioritized approach [10,23,24] is also a common choice in numerous cases. This kind of planner solves the original MAPF problem swiftly and provides solutions close to optimal, but it lacks the algorithm's completeness guarantee. Also, such solutions can hardly be applied to non-holonomic multi-robot systems.

Most of the methods above use some assumptions, like ignoring the robot's kinematic constraints and using discrete grid graphs. But when the planning problem extends to the continuous workspace, the piecewise-linear path generated by the standard A* is not guaranteed to be executable by a non-holonomic agent, i.e., the path is not drivable for car-like robots. Some researchers have started lifting some of the assumptions in their latest works. [14] takes velocity limits into account and provides a guaranteed safety distance between robots. [15] presents a generalized version of CBS for large agents that occupy more than one grid. It adds multiple constraints for one agent while expanding search nodes. [25] further generalizes the classic MAPF problem and proposes the method to plan for multiple agents with different sizes regarding time delay in robot execution. SIPPwRT [16] combines the token passing algorithm with SIPP for pickup and delivery scenarios. It plans paths for non-holonomic robots in various sizes. As for optimization-based approaches, [26] produce time-optimal trajectories for heterogeneous quadcopters. It plans in 2D Cartesian space instead of 3D physical space and suffers from a long planning time. A more efficient algorithm for multi-quadrotors trajectory planning in obstacle-dense environments is proposed in [27]. It adopts a front-end search-based approach to provide guidance for back-end trajectory optimization. [28] ports such approach to non-holonomic mobile robots in the obstacle-rich environment and presents a prioritized optimization method which decouples the problem and improves the computational efficiency significantly. However, optimization-based methods are excessively time-consuming and often fail to find a feasible solution in a limited time when the number of agents increases.

There are also noticeable studies on distributed collision avoidance for multiple non-holonomic robots. Traditional approaches for single robot can be applied, including artificial potential field [29], dynamic window approach [30], model predictive control [31] and machine learning technique [32]. The reciprocal velocity obstacle (RVO) [33] is a decentralized algorithm allowing multiple robots to avoid each other without any communication. Optimal reciprocal collision-avoidance (ORCA) [34] succeeds the concept of velocity obstacle and solves the problem faster by casting into a low-dimensional linear program. Bicycle reciprocal collision avoidance (B-ORCA) [35] and ϵCCA [36] are two adaptions of ORCA for car-like vehicles. They combine velocity obstacles with generic tracking control and generate collision-free motions under vehicles' kinematic constraints. Nevertheless, these ORCA-type methods need global path planners to avoid deadlocks in obstructed scenarios, and it does not guarantee that every robot will reach its goal.

3. CL-MAPF problem

Classic MAPF solvers usually consider holonomic agents moving in cardinal directions and neglect agents' sizes. This will cause

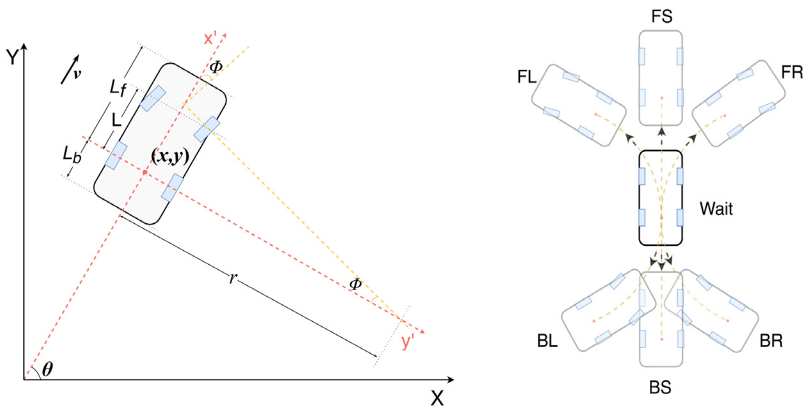


Fig. 2. Ackermann-steering model.

the generated solutions cannot be executed on real-world multi-agent systems, especially for those composed of car-like robots. In this section, we first present the robot kinematic model and then present the definition of Multi-Agent Path Finding for Car-Like robot (CL-MAPF) problem.

3.1. Robot kinematic model

Kinematic constraints must be considered for non-holonomic robots. Several path models like circular trajectories, asymptotically heading trajectories apply to different kinds of robots in practice. For car-like robots discussed in this paper, we commonly use Ackermann-steering geometry as the kinematic model shown in Fig. 2. The kinematic constraints forbid it to move laterally and rotate in place [37].

The state for an Ackermann-steering robot is denoted by $\mathbf{z} = [x, y, \theta]^{\mathrm{T}}$. The origin of rigid body frame (x, y) places at the center of robot's rear axle. The x-axis of body frame points alongside yaw angle θ, y-axis points to the left side of the robot. We use v to represent the robot's velocity, and ϕ to represent the steering angle of front wheels. When the steering angle is fixed at ϕ, radius of the circular trajectory r which robot moves along can be calculated by $r = L/\tan\phi$. L denotes agent's body length.

The kinematic relation between the steering angle ϕ and the angular velocity of yaw ω is defined as:

$$\omega = \dot{\theta} = \frac{v}{L}\tan\phi \tag{1}$$

We assume that time is discrete in the multi-agent system. The control input of the robot is defined as $\mathbf{u} = [v, \omega]^{\mathrm{T}}$. By discretizing and recursively integrating, we can calculate robot state at timestep t as following:

$$\mathbf{z}_t = [x, y, \theta]^{\mathrm{T}} = \mathbf{z}_{t-1} + [v\cos\theta, v\sin\theta, \omega]^{\mathrm{T}} \tag{2}$$

The robot's velocity v is bounded as $v_{bmax} \le v \le v_{fmax}$, where $v_{bmax} < 0$ and $v_{fmax} > 0$ represent the max speed when robot moves forward or backward, respectively. The steering angle is restricted by ϕ_{max}, which implies each Ackermann-steering robot should maintain a minimum turning radius r_{min} during the whole path.

3.2. Problem definition

We formalize the CL-MAPF problem as follows. Consider a multi-agent system containing N car-like agents $\{a_1, a_2, \ldots, a_N\}$ operating in a continuous workspace $\mathcal{W}$. The obstacles in the workspace are assumed to be known and occupy an arbitrary region $\mathcal{O} \subset \mathcal{W}$. The free workspace for agents is $\mathcal{F} = \mathcal{W}\backslash\mathcal{O}$. The state for an agent in the system is denoted as $\mathbf{z} = [x, y, \theta]^{\mathrm{T}}$.

We assume that time is discrete in the multi-agent system, and each agent keeps moving at its constant speed unless it is in a stop state at that time step. We also assume that the agent can reach a constant speed immediately after a stop state without additional acceleration time and vice versa. Given a car-like agent a_i, the region in $\mathcal{W}$ occupied by the body of a_i is denoted by $\mathcal{R}(\mathbf{z}^i)$. The function $\mathcal{R}$ is impacted by the size of the robot. Let $\mathbf{z}^i_t$ be the state of agent a_i at time t. A task is assigned to a_i to move from its start state $s^i \in \mathcal{F}$ to its goal state $g^i \in \mathcal{F}$, of course both states are within the free workspace. It is guaranteed that for all $i \neq j$, $\mathcal{R}(s_i) \cap \mathcal{R}(s_j) = \emptyset$ and $\mathcal{R}(g_i) \cap \mathcal{R}(g_j) = \emptyset$.

A path $\pi_i = [\mathbf{z}^i_0, \mathbf{z}^i_1, \ldots \mathbf{z}^i_{T_i}, \ldots]$ is *feasible* if all the following three conditions are satisfied:

- π_i should begin at its start state $\pi_i[0] = s^i$ and reach its goal state after limited timesteps $\pi_i[t] = g^i, \forall t \ge T_i$.
- Each movement in π_i should satisfy the Ackermann kinematic model as Eq. (2).
- Agent a_i should never collide with obstacles when moving along its path, $\mathcal{R}(\pi_i[t]) \subset \mathcal{F}, \forall t$.

We adopt the concept of generalized conflict proposed in [38] and use a tuple $\langle a_i, a_j, t\rangle$ to denote a *body conflict* between agent a_i and a_j. It implies those two agents collide at time t, that is $\mathcal{R}(\pi_i[t]) \cap \mathcal{R}(\pi_j[t]) \neq \emptyset$.

A *solution* for CL-MAPF problem is a set of feasible paths for all N agents where each two of them have no body conflict at any timestep. That is

$$\mathcal{R}(\pi_i[t]) \cap \mathcal{R}(\pi_j[t]) = \emptyset, \ \forall t \ge 0, i \neq j. \tag{3}$$

A CL-MAPF example is shown in problem input of Fig. 3. With reference to the classic MAPF problem [22], the solutions can be evaluated using two commonly used functions: makespan and average flowtime.

Makespan is the same as its original definition, which is defined as the time for all agents to reach their goals:

$$\max_{1\le i\le N} T_i \tag{4}$$

Average flowtime is a metric used to evaluate the mean performance of the entire solution, defined as the average time for each agent reaching their goal:

$$\frac{1}{N}\sum_{i=1}^{N} T_i \tag{5}$$

4. Methodology

The aforementioned optimal methods in Section 2 are aimed at solving the original MAPF problem in the discrete space. For

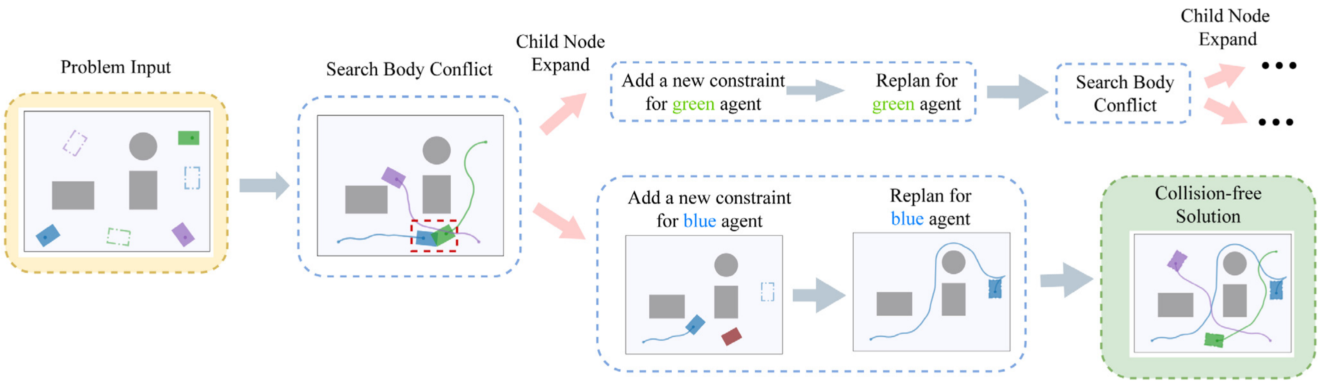


Fig. 3. A pipeline of Car-like CBS. Agents' start states are denoted as solid colored rectangle and goal states as dotted outline rectangles. Gray area represents the obstacle region $\mathcal{O}$. A body conflict between blue agent and green agent is detected in middle figure. Then two child nodes are expanded with each contains a new constraint and spatiotemporal hybrid-state A* is performed for the agent receiving it. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

example, several methods adopt the standard A* algorithm as the lower-level planner, which is optimal in the grid world. But when the planning problem extends to the continuous workspace, the piecewise-linear path generated by the standard A* is not guaranteed to be executable by a non-holonomic agent, i.e., the path is not drivable for car-like robots. This causes these methods, which are optimal for the original MAPF problem, to be unsolvable for the CL-MAPF problem.

We introduce a novel solver for the CL-MAPF problem called *Car-Like CBS* (CL-CBS). The high-level body conflict search tree is a variant of the conflict tree in original CBS. It only needs to handle inter-agent collisions, in particular, whether there is a body conflict in the whole solution without considering specific kinematic constraints. As for the low-level pathfinding method for a single agent, we proposed Spatiotemporal Hybrid-State A* algorithm to cope with both kinematic and spatiotemporal constraints. We also introduce a sequential planning version of our method at the end of this section. It remarkably shortens the searching time with little sacrifice on solution quality.

4.1. Body conflict tree

The classic MAPF solvers apply various types of conflicts (the most common ones are vertex conflicts and edge conflicts) to avoid collisions between two single-agent paths. Yet these conflicts cannot represent all situations of agent colliding. Benefit from planning in a continuous workspace, we can simply use body conflicts to describe all inter-agent collision scenarios. We propose a binary *body conflict tree* (BCT) and perform best-first search on it. Each node on BCT contains a set of inter-agent constraints and a solution that satisfies these constraints.

The expansion of the BCT works is shown in Fig. 3. First, the root node contains no inter-agent constraints for all agents. Subsequently, the path planner generates feasible paths from the start state to the goal state for each agent. When the BCT is not the empty set, node N with the smallest cost among all nodes in the BCT is popped out. In general, we define the cost of a node as the sum of all path lengths in it. We then conduct a body conflict detection for all paths in the node N.

If there are no body conflicts in N, then $N.path$ is a solution to the problem. Conversely, if there are one or more body conflicts in node N, we locate the earliest conflict in time as the body conflict to be handled at node N. Say the earliest conflict is denoted as $\langle a_i, a_j, t\rangle$, we produce two inter-agent constraints: $\langle a_j, N.path[j](t), t\rangle$ for a_i and $\langle a_i, N.path[i](t), t\rangle$ for a_j. The former constraint denote that a_i is expected to avoid the specified area $\mathcal{R}(N.path[j](t))$ at timestep t, likewise the latter. Then two

Algorithm 1: Body Conflict Tree

```
1  Root.constraints ← $\emptyset$;
2  Root.path ← path_planner($a_i$, $\emptyset$), for each $a_i$ in system;
3  BCT ← {Root};
4  while $BCT \neq \emptyset$ do
5      $N \leftarrow \min_{cost} N', \ \forall N' \in$ BCT;
6      BCT ← BCT \ {$N$};
7      $C$ ← search for first body conflict in $N.path$;
8      if $C = \emptyset$ then
9          return $N.path$;
10     end
11     foreach $a_i$ appears in $C$ do
12         $N' \leftarrow N$;
13         Add $\langle a_j, N.path[j](t), [t-k, t+k]\rangle$ to
             $N'.constraint[i]$;
14         $N'.path[i]$ ← path_planner($a_i$, $N'.constraint[i]$);
15         if $N'.path[i] \neq \emptyset$ then
16             BCT ← BCT $\cup \{N'\}$;
17         end
18     end
19 end
20 return $\emptyset$;
```

child node of N are generated, each contains one of inter-agent constraints. At last, we perform a low-level replanning process in each of the child nodes for the agent receiving the extra constraint. The pseudocode of BCT is shown in Algorithm 1.

Surely, it is possible that more than two agents in node N collide at the same moment t, especially when the system contains a large number of agents. Although our definition of the body conflict only represents the collision of two agents, the above case of more than two agents' collision can be decomposed into multiple body conflicts between two agents. For the current node N, we still handle only one body conflict and leave the remaining body conflicts to the child nodes of N for processing.

Due to factors such as communication interference, the robots will not execute paths as precisely as we scheduled in practice. For example, agent a_i is scheduled to arrive at $\pi_i[t]$ at time t, but due to a 2-timestep execution delay, it actually arrives at the location at $t+2$. We assume that each robot in the multi-agent system will have a total execution delay of up to k seconds. When an inter-agent conflict $\langle a_i, a_j, t\rangle$ occurs, we need to ensure that no further collisions due to execution delays occur within k seconds before or after time t. Thus in the algorithm

Algorithm 2: Spatiotemporal Hybrid-State A* for a_i

```
Root.state ← $s^i$;
Root.time ← 0;
Open ← {Root};
while $Open \neq \emptyset$ do
    $N \leftarrow \arg\min N'.f,\ \forall N' \in$ Open;
    if $N.state$ near $g^i$ then
        $N_{goal} \leftarrow$ Analytic_Expand($N.state, g^i$);
        $\pi_i \leftarrow$ Backtracking search from $N_{goal}$;
        if No collision in $\pi_i$ then
            return $\pi_i$;
        end
    end
    States ← Get_ChildNode($N$.state,$m_i$);
    foreach $s \in States$ do
        $N'$.state ← s;
        $N'$.time ← $N$.time +1;
        if Check_Collison($N'$.state ,$N'$.time) $= \emptyset$ then
            $N'.h \leftarrow$ Admissible_Heuristic($N'$.state,$g^i$);
            Update $N'.g$, $N'.f$;
            if $N'.state$ not appear in Open then
                Open ← Open $\cup \{N'\}$;
            else if $N'.g < N_{inOpen}.g$ then
                Update $N_{inOpen}$ with $N'$.state, $N'.f$;
            end
        end
    end
end
return $\emptyset$;
```

implementation, the inter-agent constraints last for a certain time window $\langle a_j, N.path[j](t), [t-k, t+k]\rangle$ to ensure the safety of the passage, see Line 13 in Algorithm 1.

4.2. Spatiotemporal hybrid-state A*

As mentioned above, the high-level body conflict tree requires single-agent planner to:

- Plan paths satisfying the kinematic constraint to be executed by Ackermann-steering agents;
- Plan paths satisfying spatiotemporal inter-agent constraints with other agents;

A well-known path planner applied to the continuous 3D state space for car-like robots is Hybrid-State A* [39], but it cannot deal with spatiotemporal constraints. We proposed an adaptation called Spatiotemporal Hybrid-State A* (SHA*) as the single-agent planner for Car-like CBS.

The SHA* path planner maintains an open list and conducts a best-first search in it. Each node N in the open list contains several key elements. The *state* and *time* of N represent the agent expanding to the given location at the specified time. The g, h and f function are the same as the original definitions of A*, where g is the cost of the path from the start node to N, h is the heuristic distance that estimates the cost of the cheapest path from N to the goal and $N.f = N.g + N.h$. When the node with the smallest f is popped out from the open list, we first check whether it is near the goal state. If so, the `Analytical_Expand` function is invoked to complete the path from the current state to the goal. Otherwise, we expand the node and perform a collision check on each newly generated node. If it satisfies the requirements, we compute the heuristic function of the node and update the open list. The detailed pseudocode is shown in Algorithm 2.

For a node in an open list, an overwhelming issue is to determine whether its status is legal. The robot model utilizes custom `Check_Collision` function to determine whether the state fits in the free workspace $\mathcal{F}$ and satisfies the inter-agent constraints. Specifically, node N is legal if and only if $\mathcal{R}(N.state) \cap C = \emptyset, \forall C \in \mathcal{C}_{N.time}$ and $\mathcal{R}(N.state) \subset \mathcal{F}$, where $\mathcal{C}_{N.time}$ denotes the set of inter-agent constraints at time $N.time$.

Given the continuous nature of the workspace and the discrete nature of time, we employ the concept of motion primitives [40] and discretize the agent's control inputs into an action set $\mathcal{U}$. For car-like robots, there are seven steering actions in action set $\mathcal{U}$, which are: forward max-left(FL), forward straight(FS), forward max-right(FR), backward max-left(BL), backward straight(BS), backward max-right(BR), and wait, as shown in Fig. 2. Employing `Get_ChildNode` function, we extend $N.state$ using each action u in the action set $\mathcal{U}$ and generate new states in the child nodes $N'.state$. It is worth noted that we add three penalties to cost function g when the agent performs turning actions, driving backwards, and switching the moving direction.

When adopting discrete actions, we expect the length of each extension to be small enough for combining a fine trajectory. Therefore, we limit the node extension length to be less than the agent's body length L in one time step, which implies that there is an overlap between the robot's two states $\mathcal{R}(\pi_i[t]) \cap \mathcal{R}(\pi_i[t+1]) \neq \emptyset$.

The admissible heuristic function design and the analytical expand technique of our method are the same as the original hybrid-state A*. We use the max of the non-holonomic without-obstacles cost and 2D Euclidean distance as our heuristic function, as described in [39]. Besides, using fixed steps to extend the path in a continuous workspace suffers from the problem of not being able to reach the goal state precisely. Hence, we employ `Analytical_Expand` function to reach the target state accurately when the agent is near its goal. The function disregards obstacles and inter-agent constraints and generate a Reeds–Shepp path [41] to connect the current position to the goal. If the path passes the collision check, it indicates that the planner has found a feasible path to the goal.

4.3. Sequential car-like CBS

As a result of expanding workspace from discrete space to continuous space, the computational time of single-agent planner suffers from scalability problems when the number of obstacles increasing and the workspace getting larger. Besides, the high-level search tree expands more nodes when multiple agents visiting the same region at the approximately same time. These will lead to a noticeable increase in the searching time of Car-like CBS.

Though the scalability problem of the single-agent planner is unavoidable, we propose a sequential planning method to reduce high-level search time inspired by [27]. We divide the K agents into K_b batches, and each batch contains $\lceil K/K_b \rceil$ agents except the last batch. Then we sequentially solve these sub problems for each batch and combines result paths together as the final solution of the whole problem. For a batch b, the actions of agents in subsequent batches are ignored. The paths planned out in former batches act as dynamic obstacles in the workspace, and they are added to the constraint set of the root node in BCT.

The procedure of this method is exhibited in Fig. 4(a). The agents are divided into three batches. The paths in gray planned out in the first batch act as dynamic obstacles for agents planning in the second batch (colored). Black agents denote agents of the third batch. As a result of avoiding solver to deal with too many

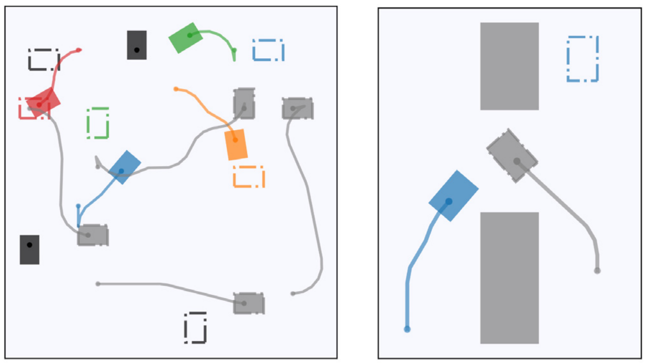


Fig. 4. (a) Sequential CL-CBS. (b) A simple fail case for sequential CL-CBS. The blue agent cannot reach its goal when the gray agent planned in former batch arrives at its goal (which locates between the obstacles) before the blue one passing through those obstacles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

agents at the same time, the sequential method shortens searching time by nearly an order of magnitude in our experiment. However, it should be noted that this sequential method may encounter failure in some cases. A simple fail case is shown and explained in Fig. 4(b).

5. Experiments

In this section, we implement Car-like CBS solver in C++ using boost library[1] for math calculation and OMPL library[2] to produce Reeds-Shepp paths. Since there is no benchmark for the CL-MAPF problem, we first generated a novel benchmark for subsequent experiments. We then compare our algorithm with two baseline planning algorithms and one suboptimal MAPF planner in the simulated environment and conduct performance tests. Finally, we perform field tests using seven Ackermann-steering robots in both obstacle-free and obstructed rooms. The source code of our method and benchmark are open source at Github[3].

5.1. Benchmarks

The classic MAPF benchmark, like DAO map sets taken from the game Dragon Age Origin [22], are all 4-neighbor grid-based benchmarks thus cannot be used for the CL-MAPF problem. Therefore we generate a novel CL-MAPF benchmark for simulated experiments.

The whole benchmark contains 3000 different instances, which involves workspaces with and without obstacles. Each scenario (w/ and w/o obstacles) includes 25 map sets. These map sets possess three types of map size (300 m $\times$ 300 m, 100 m $\times$ 100 m, 50 m $\times$ 50 m) and distinct agent numbers from 5 to 100. Every map set has 60 unique instances. The benchmark is also included in the Github repository.

For each instance in the benchmark, (i) it describes a continuous workspace; (ii) the start and goal states of agents are guaranteed not collide with each other (for agents under 5 m $\times$ 5 m size); (iii) the Euclidean distance between start and goal state of an agent is greater than 1/4 of the map width; (iv) for instances with obstacles, it contains 100 randomly generated obstacles with different radius. To be specific, it is 0.5 m, 1 m, and 2 m corresponding to the map from small to large, respectively. We use *300x300_agents80_obs* to denote mapset with 80 agents in a 300 m $\times$ 300 m workspace with obstacles.

5.2. Simulated experiments

Based on the novel benchmark described above, we can further compare CL-CBS with the baseline algorithms and examine its computational efficiency and quality of the solution. We assume agents in the simulated experiments are homogeneous with the following parameters: the shape of agents is 2 m $\times$ 3 m as L_f = 2 m, L_b = 1 m, the maximum speed for both forward and backward v_{max} = 2 m/s, the minimum turning radius r = 3 m. It should be noted that all programs are executed on a PC running Ubuntu 16.04 with Intel i7-8700@3.20 GHz and 8G RAM.

5.2.1. Comparison with baseline algorithms

We adopt two methods acting as the baselines of our experiment. (i) The centralized method is model predictive control with CBS (CBS-MPC) based on [42]. It applies the original CBS solver to provide guide paths for each agent and uses MPC to generate final trajectories. (ii) The second baseline we use is Spatiotemporal Hybrid-state A* (SHA*) using prioritized planning technique [24] to generate a priority sequence and plan in order from highest to lowest priority.

We have chosen three different scenarios for our comparison experiments, which are 300 m $\times$ 300 m with 50 agents, 100 m $\times$ 100 m with 30 agents, and 50 m $\times$ 50 m with 20 agents. Each scene contains two sub-scenes: with and without obstacles. We conducted 30 trials in each scenario for all three planners and set the runtime limit for each trial as 90 s. We adopt the sequential CL-CBS algorithm, in which batch size K_b = 2 to compete with two baseline algorithms. We compare the success rate, makespan, and average flowtime in the solution of each scenario, and the results are shown in Table 1.

In all six scenarios, the CBS-MPC algorithm has a success rate of less than 10% in five scenarios. This is due to the fact that the CBS front-end planning does not consider kinematic constraints, and the model predictive control causes the execution trajectory to deviate from the original path, which eventually leads to collisions between agents and task failure. The SHA* algorithm using the prioritized planning technique performs slightly better, achieving a success rate close to 50% in some scenarios. However, due to the more complex environment in obstructed scenarios, the agent with lower priority may not be able to find a feasible path. Therefore the success rate of SHA* then significantly drops, which is even inferior to that of CBS-MPC. In contrast, our method maintains a 100% success rate in four of the six scenarios and achieves a 98.3% success rate in two obstacle scenarios.

As for the solution quality, our approach surpasses the two baseline algorithms. In 300 m $\times$ 300 m and 50 m $\times$ 50 m scenarios, our method outperforms the baseline algorithms in both makespan and average flowtime. While in the 100 m $\times$ 100 m scenario, the metrics of our method are slightly higher than the baseline algorithm, which is due to the lower success rate of the baseline algorithms resulting in the failed instances' data are not included in the calculation of the solution quality metrics.

In summary, our method performs a much higher success rate compared to the two baseline algorithms, and the resulting multi-agent planning solution has superior quality.

[1] https://www.boost.org/.
[2] https://ompl.kavrakilab.org/.
[3] https://github.com/APRIL-ZJU/CL-CBS.

Table 1
Comparison with baselines.

Map size (m^2)	Agents	Method	Without obstacle			With obstacles		
			Success rate (%)	Makespan (s)	Average flowtime (s)	Success rate (%)	Makespan (s)	Average flowtime (s)
300 × 300	50	CBS-MPC	7.5	206.626	144.90	3.3	205.451	143.13
		SHA*	45.0	188.27	137.66	10	189.47	138.44
		Ours	**100**	**179.36**	**134.77**	**98.3**	**181.27**	**134.68**
100 × 100	30	CBS-MPC	6.7	69.9192	49.27	3.3	67.91	47.00
		SHA*	36.7	70.90	50.18	8.33	70.38	48.67
		Ours	**100**	83.73	56.32	**100**	85.76	56.60
50 × 50	20	CBS-MPC	28.2	46.38	35.64	8.3	55.35	34.51
		SHA*	15.0	49.88	32.15	3.33	54.38	32.54
		Ours	**100**	**46.25**	**30.25**	**98.3**	**47.32**	**30.81**

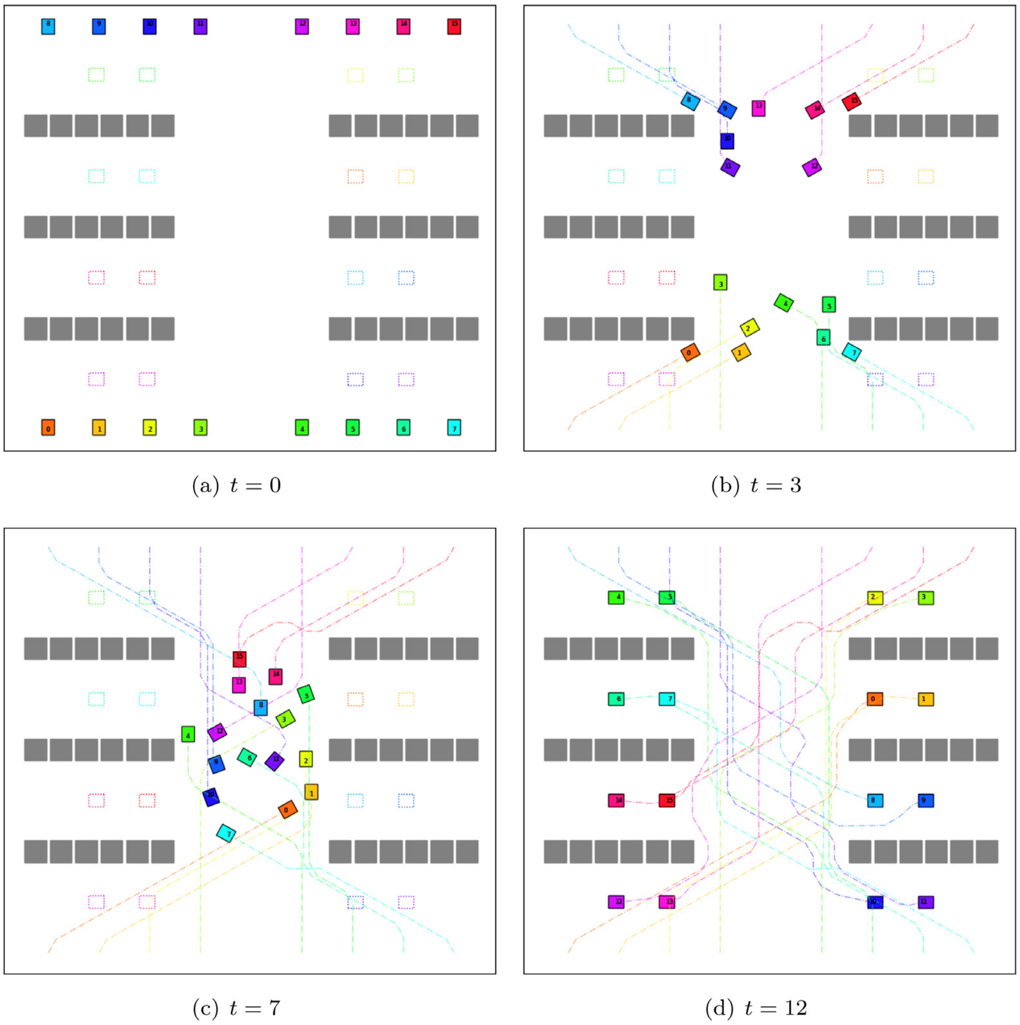

(a) $t = 0$ (b) $t = 3$ (c) $t = 7$ (d) $t = 12$

Fig. 5. The key snapshots illustrates the result trajectories of 16 car-like agents in the warehouse scenario. The makespan of the whole solution is 12 s, and the agents need to move from the edge of the scenarios to the middle of each shelf. The complexity of this scenario lies in the fact that all agents are concentrated in the center of the scene and avoid each other from $t = 4$ to $t = 8$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2.2. *Comparison with MAPF planner*

To compare with other MAPF planner, there are currently few suboptimal solvers for the CL-MAPF problem. Another suboptimal algorithm other than our CL-CBS algorithm was proposed by Li et al. [28]. Their method adopts enhanced conflict-based search (ECBS) as the multi-robot discrete path planner and uses trajectory optimization with safe corridor construction technique to convert the discrete path to kinematically feasible trajectories. Their methods include two planners: the coupled version and the prioritized version. The latter planner has a shorter computation time than the former one, but it suffers from a loss in the solution quality. We use the same warehouse scenario in their paper and compare our results with those of the two planners mentioned above.

The warehouse scenario, as shown in Fig. 5, has a size of 120 m × 100 m, and six shelves with 30 m × 6 m each are placed at the center of it. The agent size is 3 m × 2.5 m, which maximum speed is limited to 2.5 m/s. The agent number in one group is set to 3 in Li's coupled version planner. Other parameters of Li's planners are set to the program default. We use the sequential

Table 2
Comparison in warehouse scenario.

Agents	Method	Success rate (%)	Runtime (s)	Makespan (s)
20	Li's coupled	100	4.04	71.2
	Li's prioritized	100	**1.12**	74.4
	Ours	**100**	1.87	**56.2**
30	Li's coupled	100	16.73	86.4
	Li's prioritized	100	8.06	88.8
	Ours	**100**	**6.13**	**61**
40	Li's coupled	60	57.32	104.8
	Li's prioritized	90	17.63	107.2
	Ours	**100**	**11.22**	**61.6**

method of our approach for Comparison with Li's planners, and the number of robots per batch is set to 2, $K_b = 3$.

We create a set containing 40 locations evenly distributed in the warehouse scenario. For each trial, every robot randomly picks two locations from the set as the start and goal state. It is guaranteed that the start and goal states of any two robots are different. We performed tests with the agent number at 20, 30 and 40, respectively. The runtime of each planner is limited to 60 seconds. For each test, we conduct 10 trials for each planner and calculate the success rate, average runtime, and makespan, as shown in Table 2.

Our method maintains a 100% success rate as the number of robots increases, while Li's planers fail several times when the number of robots reaches 40. All failures are caused by the ECBS module in their planner running out of time. Regarding the runtime, the coupled version of Li's planner shows exponential growth while the other two planners maintain a linear increase. When planning for 40 robots, Li's planner's prioritized version takes 17.6 s caused by the infeasible grouping process, while our method takes only 11.2 s. The advantages of our approach become greater when it comes to metrics for evaluating the optimality of solutions, i.e., makespan.

When the number of robots becomes larger and the warehouse becomes more crowded, both two versions of Li's planner's solution have a noticeable increase in makespan. When there are 20 robots in the system, the makespan of Li's planner is 32.3% greater than our method, while the difference reaches 74% for the robot's number equals 40. As Li's method uses a framework of discrete path planning followed by trajectory optimization, the resulting trajectories rely on the paths planned in the grid map. The grid map discretely samples the continuous space, resulting in unnecessary detours in the optimized trajectory. Furthermore, the time robots wait for others to pass will grow significantly when the agent number increases.

5.2.3. Performance tests

In this section, we measure the performance of our approach when extended to a large number of agents. Four scenarios were chosen for testing: the 300 m × 300 m and 100 m × 100 m dataset, with each dataset divided into obstructed and obstacle-free scenarios. The two algorithms tested include the original CL-CBS and its sequential version (which batch size $K_b = 2$). We conduct 30 trials on the measured planners at each agent quantity (from 10 to 100) and set the runtime limit for each trial to 90 s. We calculate the success rate, average runtime, makespan, and average flowtime of the two methods in different scenarios, and the results are shown in Figs. 6 and 7.

In 300 m × 300 m scenarios, all planners have a 100% success rate when there are relatively few robots. When there are more than 50 robots, the original planner exceeds the runtime limit in some cases, and the success rate starts to decrease due to the increase in body conflicts. With more than 70 robots, the original planner merely finds out solutions in less than 20% of cases within the time limit. However, the sequential version of our method maintains an over 80% success rate for up to 100 robots in both obstructed and obstacle-free scenarios. The original planner's runtime tends to increase exponentially as the number of robots increases, while those of the sequential CL-CBS planner are shorter. The original planner takes an average of 59.62 s to compute a solution for 75 agents in the obstacle-free scenario, while the sequential version takes only 18.98 s, which is one-third of the former time.

As the number of agents increases, the behavior of avoidance and detours in the planning solution also rises, which eventually manifests itself as a rise in makespan (from less than 170 to a maximum of 187.9). The makespan does not differ much between two planners in the same test case of 300 m × 300 m scenarios. The solution's average flowtime of the sequential CL-CBS is slightly higher than that of the original planner in both obstructed and obstacle-free scenarios due to the fact that the sequential version reduces the quality of the solution in exchange for a shorter program runtime. However, the maximum difference between these two planners on this metric lies in 1.5%.

The results of 100 m × 100 m scenarios are shown in Fig. 7. The trend of solution quality with the increasing agents is similar to that of the 300 m × 300 m cases. The success rate of original CL-CBS drops sharply after the number of agents exceeds 20, while the sequential version can still maintain a success rate of 73.3% in cases with 50 agents. The success rate in the empty scenarios is generally greater than that in the obstacle ones owing to the larger feasible workspace. A visible difference in makespan and average flowtime is produced between sequential and original CL-CBS. There is no doubt that the original CL-CBS has a lower makespan and better solution quality in the same scenario, but the sequential CL-CBS keeps the solution's average flowtime within 117% of the original version. Therefore, we believe that it is worth sacrificing a small amount of solution quality to achieve a significant reduction in runtime and a considerable increase in success rate.

5.3. Field test

We conduct field tests using seven 23 cm × 20 cm Ackermann-steering robots produced by WeTech as shown in Fig. 1(a). The robot is able to move at 0.3 m/s, and the minimum turning radius is 0.26 m. All the robots are equipped with a 2D Lidar from Slamtec, a 5-megapixel camera, and a Raspberry Pi 4 running Ubuntu 18.04. We use a PC laptop running ROS Melodic as the central computing station to communicate with all agents using 2.4 GHz Wifi.

Experiments are performed in a 5 m × 3 m room, including both obstacle-free and obstructed scenarios. The obstacle-free scenario contains six Ackermann-steering agents, while there are seven agents operating in the obstructed scenario. We adopt the gmapping algorithm to create 2D occupancy maps for both scenarios. After the mapping was completed, 194 obstacles with a diameter of 0.05 m made up the room boundaries, and 86 obstacles of the same diameter represent the cardboard boxes placed in the obstructed scenario.

After appointing start and goal states for all agents, a solution is computed on the laptop with Intel i5-9300@2.40 GHz. For both scenarios, we apply the original CL-CBS instead of its sequential version and perform five attempts with 60 s time limit. The planner runs in a single thread on one core of the CPU, and then we calculate the average running time of the program. In the obstacle-free scenario, all five attempts are successful, with an average runtime of 2.204 s. There are six kinematic constraints in this scenario, which is equal to the agents' number, and the minimum average flowtime is 18.98 after adding seven

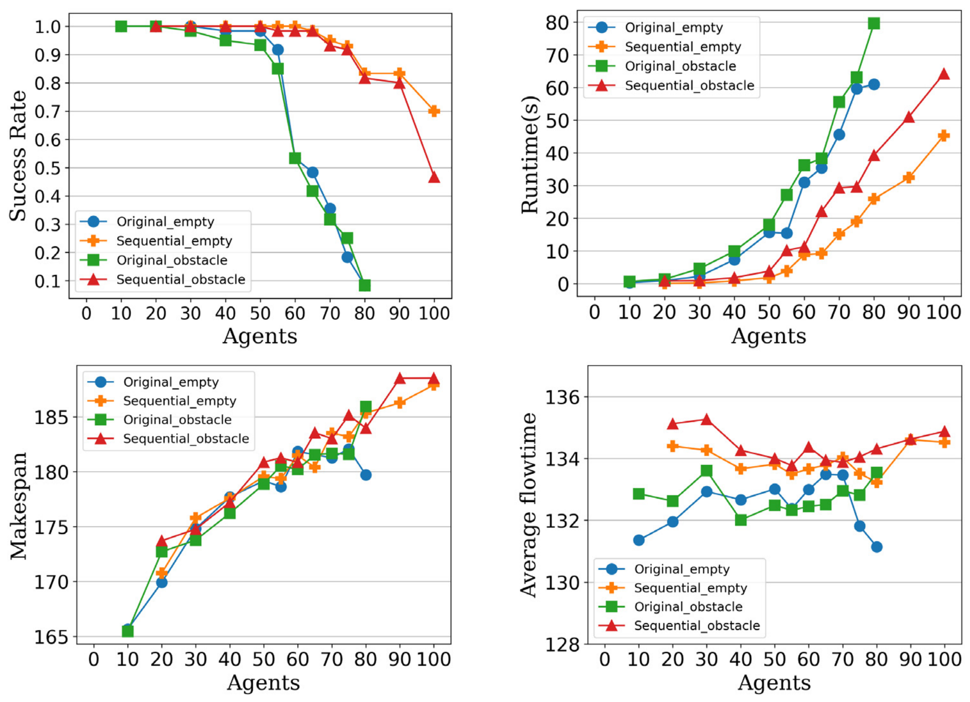


Fig. 6. Experiment results in the 300 m × 300 m dataset. The legend containing the word *original* means that it is the result obtained by the original method, and the word *sequential* for results obtained by the sequential method. *obstacle* and *empty* represent obstructed and obstacle-free scenarios respectively.

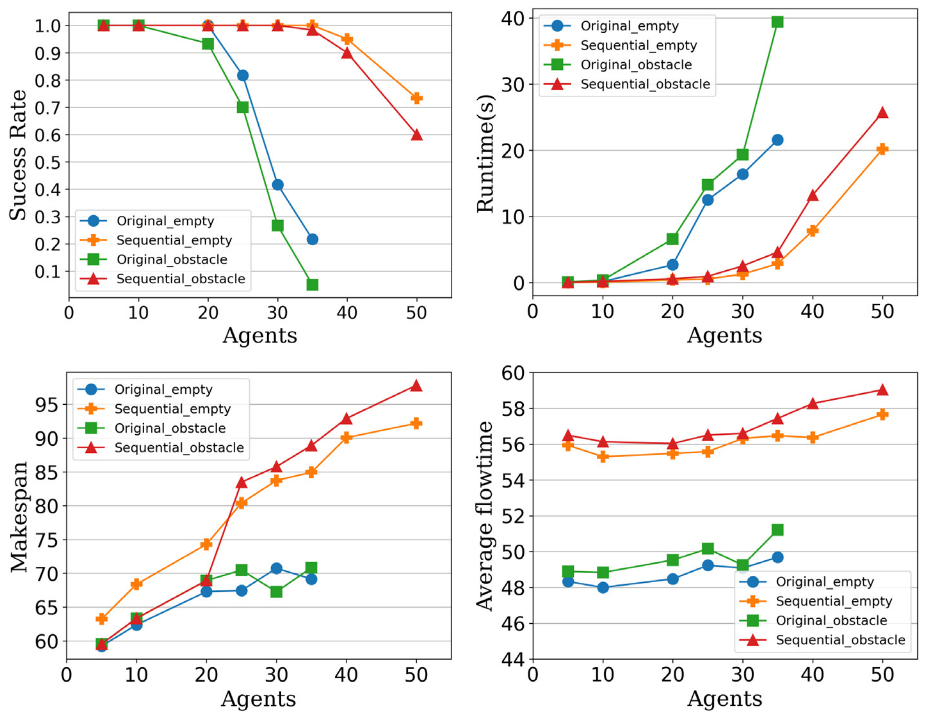


Fig. 7. Experiment results in the 100 m × 100 m dataset. The legend containing the word *original* means that it is the result obtained by the original method, and the word *sequential* for results obtained by the sequential method. *obstacle* and *empty* represent obstructed and obstacle-free scenarios respectively.

spatiotemporal constraints. As for the obstructed scenario, there is also a 100% success rate in five trials with an average runtime of 4.774 s. Seven kinematic constraints are constructed in this scenario, and the minimum average flowtime is 20.67 after adding nine inter-agent spatiotemporal constraints. Compared to the former one, the obstructed scenario is more complex, and the agents' paths are more prone to collision. Thus more inter-agent constraints are needed to complete the solution.

We transfer agents' paths to a sequence of velocity commands and send them to robots in the system for execution. Amcl

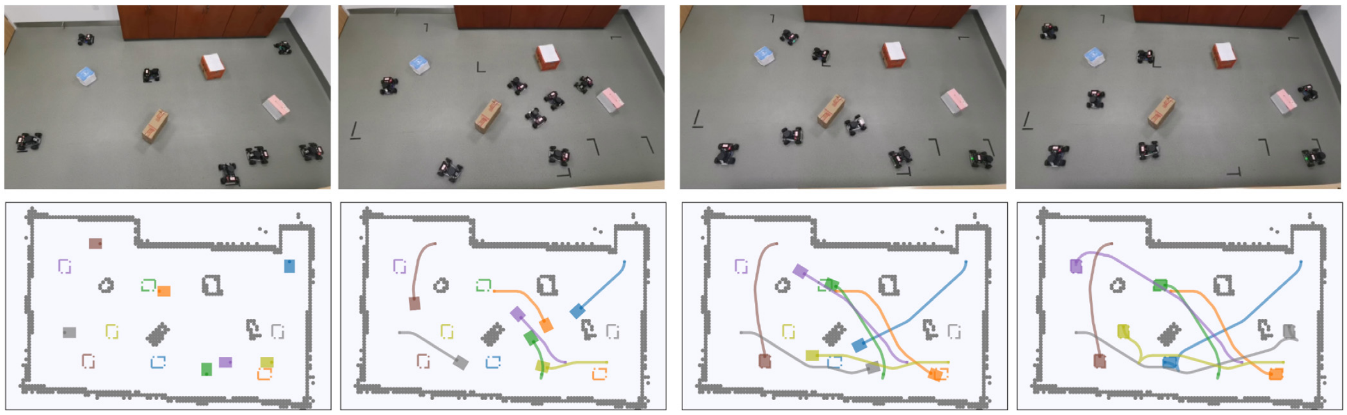


Fig. 8. Glimpses of field tests. Snapshots in the upper row show four keyframes during an experiment, and pictures in the lower row plot the trajectories agents have driven at the corresponding frame. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

package is used when robots are running so that we can get the trajectory. A snapshot taken from one of the field tests is shown in Fig. 8, and full experiments are presented in the YouTube video[4].

6. Conclusion and future work

In this paper, we formalize the CL-MAPF problem, which is to plan for multiple car-like robots in a continuous workspace while considering the kinematic and spatiotemporal constraints. We propose a novel hierarchical search-based solver called CL-CBS. It gains the advantage of short computation time using a binary conflict search tree while possessing the ability to plan kinematic-feasible paths for a large number of car-like robots. We also present a sequential version of our method that further reduces the algorithm computation time at the expense of a marginal loss in solution quality.

The experiment results show that the planning success rate of both baseline algorithms is below 50% for all six scenarios, while our algorithm maintains that of over 98%. The solution quality of our approach also surpasses the two baselines' quality. Compared to the other suboptimal MAPF solver, our method not only has an advantage in runtime but also has an average solution makespan that is more than 33% smaller than the comparison method. In performance tests, the sequential version of CL-CBS can solve for up to 100 agents (in 300 m $\times$ 300 m scenario) or 50 agents (in 100 m $\times$ 100 m scenario) agents in an acceptable runtime and keep a 60+% success rate. Finally, we perform field tests using seven Ackermann-steering robots in both obstacle-free and obstructed rooms.

One of the directions of future research is extending our method in order to plan for holonomic and non-holonomic agents under the same scenario. Another one is applying the proposed approach to the combined target-assignment and path-finding (TAPF) problem [43]. The TAPF problem first assigns agents in the system to suitable targets and then plans collision-free paths to the targets for the agents.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

4 https://youtu.be/KThsX04ABvc.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant 62088101 and the Key Research and Development Program of Zhejiang Province, China under Grant 2021C01035.

References

[1] J. Yu, S.M. LaValle, Structure and intractability of optimal multi-robot path planning on graphs, in: Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013.
[2] J. Yu, S. LaValle, Planning optimal paths for multiple robots on graphs, in: 2013 IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 3612–3617.
[3] P. Surynek, Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally, in: Twenty-Fourth International Joint Conference on Artificial Intelligence.
[4] E. Erdem, D.G. Kisa, U. Oztok, P. Schüller, A general formal framework for pathfinding problems with multiple agents, in: Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013.
[5] G. Wagner, H. Choset, M*: A complete multirobot path planning algorithm with performance bounds, in: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2011, pp. 3260–3267.
[6] T.S. Standley, Finding optimal solutions to cooperative pathfinding problems, in: AAAI, Vol. 1, Atlanta, GA, 2010, pp. 28–29.
[7] C. Ferner, G. Wagner, H. Choset, Odrm* optimal multirobot path planning in low dimensional search spaces, in: 2013 IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 3854–3859.
[8] M. Phillips, M. Likhachev, Sipp: Safe interval path planning for dynamic environments, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 5628–5635.
[9] S.-H. Ji, J.-S. Choi, B.-H. Lee, A computational interactive approach to multi-agent motion planning, Int. J. Control Autom. Syst. 5 (3) (2007) 295–306.
[10] M. Čáp, P. Novák, A. Kleiner, M. Seleckỳ, Prioritized planning algorithms for trajectory coordination of multiple mobile robots, IEEE Trans. Autom. Sci. Eng. 12 (3) (2015) 835–849.
[11] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artificial Intelligence 219 (2015) 40–66.
[12] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, E. Shimony, Icbs: The improved conflict-based search algorithm for multi-agent pathfinding, in: Eighth Annual Symposium on Combinatorial Search, Citeseer, 2015.
[13] A. Felner, R. Stern, S.E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, P. Surynek, Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges, in: Tenth Annual Symposium on Combinatorial Search, 2017.
[14] W. Hönig, T.S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, S. Koenig, Multi-agent path finding with kinematic constraints, in: ICAPS, Vol. 16, 2016, pp. 477–485.
[15] J. Li, P. Surynek, A. Felner, H. Ma, T.S. Kumar, S. Koenig, Multi-agent path finding for large agents, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 7627–7634.

[16] H. Ma, W. Hönig, T.S. Kumar, N. Ayanian, S. Koenig, Lifelong path planning with kinematic constraints for multi-agent pickup and delivery, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 7651–7658.
[17] K. Yakovlev, A. Andreychuk, V. Vorobyev, Prioritized multi-agent path finding for differential drive robots, in: 2019 European Conference on Mobile Robots, ECMR, IEEE, 2019, pp. 1–6.
[18] Y. Dobrev, M. Vossiek, M. Christmann, I. Bilous, P. Gulden, Steady delivery: Wireless local positioning systems for tracking and autonomous navigation of transport vehicles and mobile robots, IEEE Microw. Mag. 18 (6) (2017) 26–37.
[19] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T.S. Kumar, S. Koenig, H. Choset, PRIMAL: PAthfinding via reinforcement and imitation multi-agent learning, IEEE Robot. Autom. Lett. 4 (3) (2019) 2378–2385.
[20] L. Wen, J. Yan, X. Yang, Y. Liu, Y. Gu, Collision-free trajectory planning for autonomous surface vehicle, in: 2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 2020, pp. 1098–1105, http://dx.doi.org/10.1109/AIM43001.2020.9158907.
[21] M. Veloso, J. Biswas, B. Coltin, S. Rosenthal, CoBots: Robust symbiotic autonomous mobile service robots, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
[22] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.S. Kumar, et al., Multi-agent pathfinding: definitions, variants, and benchmarks, in: Twelfth Annual Symposium on Combinatorial Search, 2019.
[23] D. Silver, Cooperative pathfinding, AIIDE 1 (2005) 117–122.
[24] J.P. Van Den Berg, M.H. Overmars, Prioritized motion planning for multiple robots, in: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2005, pp. 430–435.
[25] D. Atzmon, Y. Zax, E. Kivity, L. Avitan, J. Morag, A. Felner, Generalizing multi-agent path finding for heterogeneous agents, in: Thirteenth Annual Symposium on Combinatorial Search, 2020.
[26] D.R. Robinson, R.T. Mar, K. Estabridis, G. Hewer, An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments, IEEE Robot. Autom. Lett. 3 (2) (2018) 1215–1222.
[27] J. Park, J. Kim, I. Jang, H.J. Kim, Efficient multi-agent trajectory planning with feasibility guarantee using relative Bernstein polynomial, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2020, pp. 434–440.
[28] J. Li, M. Ran, L. Xie, Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization, IEEE Robot. Autom. Lett. 6 (2) (2021) 405–412, http://dx.doi.org/10.1109/LRA.2020.3044834.
[29] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, in: Autonomous Robot Vehicles, Springer, 1986, pp. 396–404.
[30] O. Brock, O. Khatib, High-speed navigation using the global dynamic window approach, in: Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), 1, 1999, pp. 341–346 vol.1, http://dx.doi.org/10.1109/ROBOT.1999.770002.
[31] D. Morgan, S.-J. Chung, F.Y. Hadaegh, Model predictive control of swarms of spacecraft using sequential convex programming, J. Guid. Control Dyn. 37 (6) (2014) 1725–1740.
[32] J.-S. Park, B. Tsang, H. Yedidsion, G. Warnell, D. Kyoung, P. Stone, Learning to improve multi-robot hallway navigation, in: Proceedings of the 4th Conference on Robot Learning (CoRL), 2020.
[33] J. Van den Berg, M. Lin, D. Manocha, Reciprocal velocity obstacles for real-time multi-agent navigation, in: 2008 IEEE International Conference on Robotics and Automation, IEEE, 2008, pp. 1928–1935.
[34] J. Van Den Berg, S.J. Guy, M. Lin, D. Manocha, Reciprocal n-body collision avoidance, in: Robotics Research, Springer, 2011, pp. 3–19.
[35] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, R. Siegwart, Reciprocal collision avoidance for multiple car-like robots, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 360–366.
[36] J. Alonso-Mora, P. Beardsley, R. Siegwart, Cooperative collision avoidance for nonholonomic robots, IEEE Trans. Robot. 34 (2) (2018) 404–420.
[37] L. Marin, M. Vallés, A. Soriano, A. Valera, P. Albertos, Event-based localization in ackermann steering limited resource mobile robots, IEEE/ASME Trans. Mechatronics 19 (4) (2013) 1171–1182.
[38] W. Hönig, J.A. Preiss, T.S. Kumar, G.S. Sukhatme, N. Ayanian, Trajectory planning for quadrotor swarms, IEEE Trans. Robot. 34 (4) (2018) 856–869.
[39] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel, Practical search techniques in path planning for autonomous driving, Ann Arbor 1001 (48105) (2008) 18–80.
[40] A. Botros, S.L. Smith, Computing a minimal set of t-spanning motion primitives for lattice planners, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 2328–2335, http://dx.doi.org/10.1109/IROS40897.2019.8968150.
[41] J. Reeds, L. Shepp, Optimal paths for a car that goes both forwards and backwards, Pacific J. Math. 145 (2) (1990) 367–393.
[42] R. Negenborn, B. De Schutter, J. Hellendoorn, Multi-agent model predictive control: a survey, (04-010) Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, 2004.
[43] H. Ma, S. Koenig, Optimal target assignment and path finding for teams of agents, in: AAMAS '16, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, ISBN: 9781450342391, 2016, pp. 1144–1152.

Licheng Wen received his B.S. degree in College of Control Science and Engineering from Zhejiang University in 2019. He is currently a M.S. degree candidate of the institute of Cyber Systems and Control, Department of Control Science and Engineering, Zhejiang University. His latest research interests include mobile robots and motion planning.

Yong Liu received the B.S. degree in computer science and engineering and the Ph.D degree in computer science from Zhejiang University, Zhejiang, China, in 2001 and 2007, respectively. He is currently a professor of Institute of Cyber-Systems and Control at Zhejiang University. His main research interests include: intelligent robot systems, robot perception and vision, deep learning, big data analysis, and multi-sensor fusion. He has published over 30 research papers on machine learning, computer vision, information fusion, and robotics.

Hongliang Li received his B.S. degree in Industrial Automation from Zhejiang University in 1999, his M.S. degree in Control Science and Engineering from Zhejiang University in 2002, and his Ph.D. degree in Chemical Process Machinery and Control from Nanjing University of Technology in 2012. He is currently an associate researcher at the Institute of Network Systems and Control, Zhejiang University.