CL-MAPF: Multi-Agent Path Finding for Car-Like Robots with Kinematic and Spatiotemporal Constraints

Licheng Wen[1], Zhen Zhang[1], Zhe Chen[1], Xiangrui Zhao[1], and Yong Liu[1,*]

***Abstract*— 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. They limit 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 Multi-Agent Path Finding for Car-Like robots (CL-MAPF) problem. For the first time, 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 shapes of the agents. We introduce a new algorithm called Spatiotemporal Hybrid-State A* as the single-agent path planner to generate path satisfying both kinematic and spatiotemporal constraints. We also present a sequential planning version of our method for the sake of efficiency. We compare our method with two baseline algorithms on a dedicated benchmark containing 3000 instances and validate it in real-world scenarios. The experiment results give clear evidence that our algorithm scales well to a large number of agents and is able to produce solutions that can be directly applied to car-like robots in the real world. The benchmark and source code are released in `https://github.com/APRIL-ZJU/CL-CBS`.**

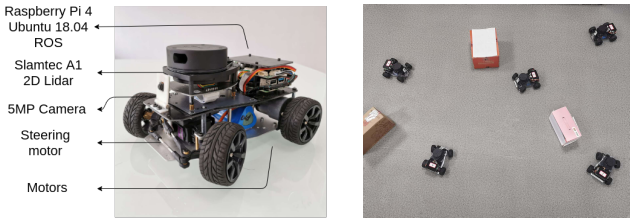
I. Introduction

Multi-Agent Path Finding, also known as MAPF, is a crucial planning problem for multiple agents. Each agent is required to move from an initial start 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 nonholonomic and designed as car-like vehicles in practice. However, almost all the above methods base on assumptions that agents are modelled as disks and are capable of rotating in place. These solvers also adopt discrete 4-neighbor grids as their search space.

[1]Licheng Wen, Zhen Zhang, Zhe Chen, Xiangrui Zhao and Yong Liu are with the State Key Laboratory of Industrial Control Technology and Institute of Cyber-Systems and Control, Zhejiang University, Zhejiang, 310027, China (*Yong Liu is the corresponding author, yongliu@iipc.zju.edu.cn)



(a) WeTech Robot (b) Snapshot of the field test

Fig. 1: Our proposed method tested on seven ackermann-steering robots produced by WeTech. The experiment video is available in the attachment of this paper.

In reality, car-like robots 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. 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 depend on their specific environments, and they can not represent all the collision scenarios.

To address these concerns, it is essential to formalize *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(CL-CBS) to settle this problem.

Our main contributions are summarized as follows:

- We present a complete CL-MAPF solver, which simply uses body conflicts to describe all inter-agent collision scenarios. Our approach also ensures the robustness for agents' execution error.
- We propose a new single-agent path planner for car-like robots, which generates path satisfying both kinematic and spatiotemporal constraints.
- We also introduce a sequential version of our original method, which significantly reduces search time with little sacrifice on performance.
- We conduct experiments in both simulated and physical environment. 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.

II. Related Works

MAPF problem has been widely studied in the robotic and AI community. One way to solve the problem is reduction to

other well-studied combinatorial problems [2]–[4]. Besides, several solvers using search techniques have been proposed to solve this problem. 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. Another complete and optimal MAPF solver is the Safe Interval Path Planning (SIPP) [8]. It runs an A* search in a graph where each node represents a pair of vertexes in the workspace and a safe time interval. One popular branch of MAPF solvers nowadays is based on a two-level optimal solver called Conflict-Based Search(CBS) [11]. ICBS [12] and CBSH [13] improves CBS further by classifying conflicts and resolving cardinal conflicts first. Finally, a prioritized approach [10] is also a common choice in numerous cases. However, it lacks a completeness guarantee.

Most of the methods above use some assumptions, like ignoring robot's kinematic constraints and using discrete grid graphs. MAPF-POST algorithm that works on differential-drive robots is proposed in [14]. It 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. SIPPwRT [16] combines the token passing algorithm with SIPP for pickup and delivery scenarios. In [17] a road-map based planner supporting different moving speeds is suggested, and a grid-based planner capable of handling any-angle moves using a variant of SIPP is proposed in [23].

There are also researches about distributed collision avoidance for multiple nonholonomic robots. Traditional approaches for single robot can be applied, including artificial potential field [24], dynamic window approach [25], and model predictive control [26]. The reciprocal velocity obstacle (RVO) [27] is a decentralized algorithm allowing robots to avoid each other with no communication between them. Optimal reciprocal collision-avoidance (ORCA) [28] 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) [29] and ϵCCA [30] are two adaptions of ORCA for car-like vehicles. Nevertheless, ORCA-based methods need global path planners to avoid deadlocks in scenarios with obstacles and cannot guarantee that each robot can reach its goal.

III. CL-MAPF Problem

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

A. Robot Kinematic Model

Kinematic constraints must be considered for nonholonomic 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 [31].

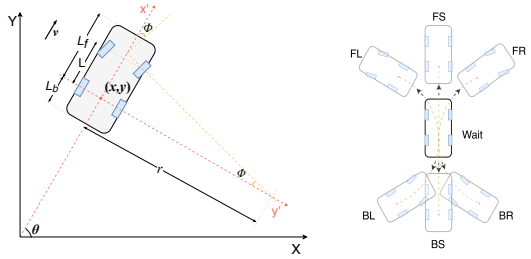


Fig. 2: Ackermann-steering model

The state for an Ackermann-steering robot is denoted by $\mathbf{s} = (x, y, \theta)$. 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. v represents the robot's velocity, and ϕ represents the steering angle of front wheels. When the steering angle is fixed at ϕ, radius of the circular trajectory which robot moves along is denoted as $r = L/\tan\phi$. Let $dw = r \cdot d\theta$ represents the distance along trajectory after time dt.

The kinematic relation between ϕ and $\dot{\theta}$ is defined as:

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

By discretizing and recursively integrating with sample time T_s , we can calculate robot state at timestep t as following:

$$\mathbf{s}_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + T_s \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{L}\tan\phi \end{bmatrix}_{t-1} \tag{2}$$

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.

B. Problem Definition

Given a workspace $\mathbf{W} \subset \mathbb{R}^3$ and a set of obstacles occupying an arbitrary region $\mathbf{O}_{ws} \subset \mathbf{W}$, we formalize CL-MAPF problem as follows.

There are K car-like agents $a_1, a_2, ..., a_K$. Time is assumed to be discretized. Let $\mathbf{s}_t^i$ be the state of agent a_i at timestep t. The start and goal state of a_i is respectively denoted as $\mathbf{s}_{start}^i \in \mathbf{W}$ and $\mathbf{s}_{goal}^i \in \mathbf{W}$. A *single-agent path* $p^i = [\mathbf{s}_0^i, ..., \mathbf{s}_{T_i}^i, \mathbf{s}_{T_i+1}^i, ...]$ for a_i is *feasible* iff all the following four conditions are satisfied:

a. Path of a_i should begin at its start state and stops at its goal states after limited timesteps T_i. That is $p^i[0] = \mathbf{s}_{start}^i$ and $p^i[T_i] = \mathbf{s}_{goal}^i$.
b. Agent a_i would stay at the goal position after reaching it, $\forall t \ge T_i$, $p^i[t] = \mathbf{s}_{goal}^i$.
c. Agent a_i should never collide with obstacles at any timestep t.

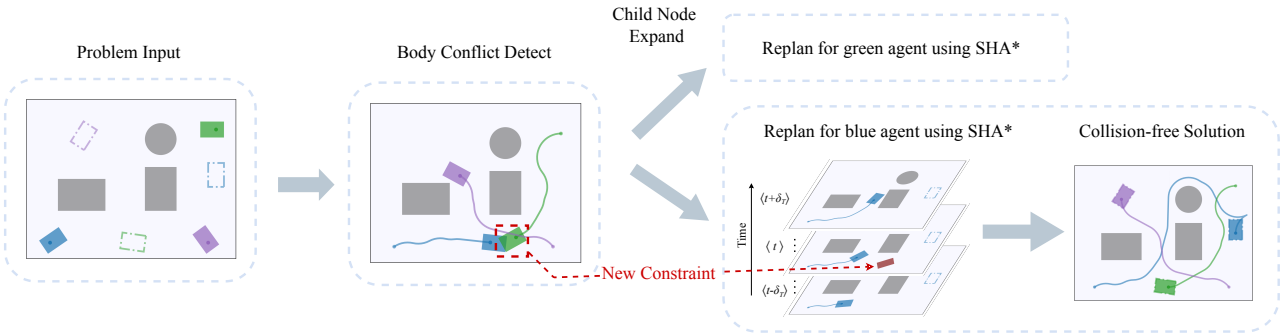


Fig. 3: A pipeline of Car-like CBS. Agents' start states are denoted as solid colored rectangle and goal states as dotted outline rectangles. Grey area represents the obstacle region $\mathbf{O}_{ws}$. A body conflict between blue agent and green agent is detected in middle figure. Then two child nodes are expanded with each contains a new constraint and spatiotemporal hybrid-state A* is performed for the agent receiving it.

d. Each move of an Ackermann-steering agent should satisfy the kinematic model. Thus given agent's max forwarding speed v_{fmax}, max reversing speed v_{rmax} and max steering angle ϕ_{max}, agent state $p^i[t]$ and $p^i[t+1]$ should obey Equation 2 for any timestep t.

As shown in Fig.2, we use L_f and L_b to denote distance from rear axle to robot front and robot rear, respectively. W_r denotes the width of robot. For an agent at state (x_0, y_0, θ_0), the rectangle shape of agent body C in Cartesian coordinate system can be defined as:

$$\begin{aligned} C &= \left\{(x,y) \in \mathbb{R}^2, f(x,y) \leq 2\right\} \\ f(x,y) &= \left|\frac{x'}{L_b+L_f} + \frac{y'}{W_r}\right| + \left|\frac{x'}{L_b+L_f} - \frac{y'}{W_r}\right| \end{aligned} \tag{3}$$

where,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} L_b \\ \frac{W_r}{2} \end{bmatrix}$$

We use a tuple $\langle a_i, a_j, C_t^i, C_t^j, t\rangle$ to denote a *collision* , or a *body conflict*, between agent a_i and agent a_j at timestep t when $C_t^i \cap C_t^j \neq \emptyset$. C_t^i represents body rectangle for a_i at timestep t.

A *solution* for CL-MAPF problem is a set of feasible paths for all K agents where every two paths are *collision-free*. A CL-MAPF example is shown in problem input of Fig.3.

Solutions can be evaluated using two commonly used functions: makespan and sum of costs. Since agents are not required to move at the same speed each timestep, definitions of these two functions slightly vary from classic MAPF. Makespan is the maximum length of paths for all agents to reach their goals, $\max_{i\in[1,K]} d(p^i)$. Sum of costs is the total length of all agents' paths, $\sum_{i=1}^{K} d(p^i)$. Since the original MAPF problem is proven to be NP-hard [2], finding an optimal solution for CL-MAPF are also NP-hard.

IV. Methodology

We introduce a novel solver for CL-MAPF problem called Car-like CBS. The high-level body conflict search tree is a variant of the conflict tree in CBS. As for low-level path finding method for 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 sacrifices on completeness.

A. 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. When the leaf node N with minimum cost popped out from BCT, a collision check is executed for the solution belonging to N. If a body conflict $\langle a_i, a_j, C_t^i, C_t^j, t\rangle$ has been detected, we produce two inter-agent constraints: $\langle a_i, C_t^j, [t-\delta_T, t+\delta_T]\rangle$ for a_i and $\langle a_j, C_t^i, [t-\delta_T, t+\delta_T]\rangle$ for a_j. The former constraint denotes that agent a_i should not pass through rectangle area C_t^j from timestep $t-\delta_T$ to time step $t+\delta_T$, likewise the latter. Then two child node of N are generated, each contains one of inter-agent constraints. At last we perform a low-level search in both of child nodes for the agent received the extra constraint. The pseudocode of BCT is shown from line 1 to line 19 in Algorithm 1.

In the view of robots will not execute as we expect in practice, our method retains certain robustness in both time and space dimensions. When there are execution errors on position for agents, we inflate the rectangle area C_t of inter-agent constraint. By multiplying an inflation coefficient k on param L_f, L_b and W_c in Equation 3, robots possess bigger space for others to bypass. The param δ_T of the constraint definition is used for eliminating errors in the time dimension.

B. Spatiotemporal Hybrid-State A*

As mentioned above, the high-level body conflict tree requires low-level search to:

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

Algorithm 1: Car-like CBS

1 $Root.constraints \leftarrow \emptyset$;
2 $Root.plan \leftarrow$ path for each agent using $SH\_Astar(a_i)$;
3 BCT$\leftarrow \{Root\}$;
4 **while** *BCT*$\neq \emptyset$ **do**
5 $Node \leftarrow \arg\min_{N' \in \text{BCT}} N'.\text{cost}$;
6 BCT $\leftarrow$ BCT $\setminus \{Node\}$;
7 $C \leftarrow CollisionDetect(Node.plan)$;
8 **if** C=$\emptyset$ **then**
9 **return** $Node.plan$;
10 **end**
11 **foreach** $C_i = \langle a_i, C_i^j, [t-\delta_T, t+\delta_T] \rangle \in C$ **do**
12 $New.plan \leftarrow Node.plan$;
13 $New.constraints \leftarrow Node.constraints \cup \{C_i\}$;
14 $New.plan\ for\ a_i \leftarrow SH\_Astar(a_i)$;
15 **if** $SH\_Astar(a_i) \neq \emptyset$ **then**
16 BCT $\leftarrow$ BCT $\cup \{New\}$;
17 **end**
18 **end**
19 **end**
20 **Function** `SH_Astar(`a_i`)`**:**
21 $Open \leftarrow \{(0, x^i_{start}, y^i_{start}, \theta^i_{start})\}$;
22 **while** $Open \neq \emptyset$ **do**
23 $N \leftarrow \arg\min_{N' \in \text{Open}} N'.\text{fScore}$;
24 **if** $(N.x, N.y, N.\theta)$ *near* $\mathbf{s}^i_{goal}$ **then**
25 $path_{toGoal} \leftarrow AnalyticExpand(\mathbf{s}^i_{goal})$;
26 **if** $CollisionDetect(path_{toGoal}) = \emptyset$ **then**
27 **return** $path_{whole}$;
28 **end**
29 **end**
30 **foreach** *act* $\in$ *steering actions* **do**
31 $N' = (N.t + T_s, x, y, \theta) \leftarrow Expand(N, act)$;
32 **if** ***not*** *SatisfyConstraint*(N') ***or*** $N' \in \mathbf{O}_{ws}$ **then**
33 **continue**
34 **end**
35 $N'.gScore = N.gScore + penalty(act) \times cost(act)$;
36 $N'.h = N'.gScore + heuristic(N', \mathbf{s}^i_{goal})$;
37 **if** $N' \notin Open$ **then**
38 $Open \leftarrow Open \cup \{N'\}$;
39 **else if** $N'.gScore < N_{inOpen}.gScore$ **then**
40 update Node with $N'.state, N'.gScore$
41 **end**
42 **end**
43 **end**
44 **return** $\emptyset$;

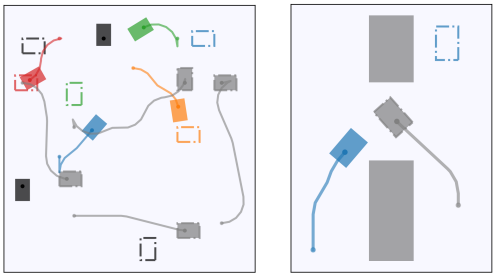
- Discretize paths by sample time T_s and return a state sequence $p[t]$;

A well-known path planner applied to the continuous 3D state space for car-like robots is Hybrid-State A* [32], but it cannot deal with spatiotemporal constraints. We proposed an adaptation called Spatiotemporal Hybrid-State A* (SHA*) as the low-level planner for Car-like CBS.

When planning for multiple agents, the ability to stay still at the current state in order to avoid other agents moving is necessitated. Thus, the seven steering actions for expanding child nodes 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. We denote actions besides moving straight and waiting as *turning actions*.

Spatiotemporal hybrid-state A* uses a 4D search space (t, x, y, θ), with $x, y, \theta \in \mathbb{R}$ and time t being discrete. For a node with search state $(t_0, x_0, y_0, \theta_0)$, its child nodes will have states like $(t_0 + T_s, x_1, y_1, \theta_1)$, where (x_1, y_1, θ_1) are computed using robot kinematic Equation 2. When a node popped from the open list, we use seven different steering actions to expand this node. For each of seven child states, we not only check collisions with obstacles in $\mathbf{O}_{ws}$, but also check if the state satisfies all the spatiotemporal constraints for this agent. Spatiotemporal hybrid-state A* is indicated from line 20 to line 44 in Algorithm 1 as *SH_Astar* function.

It is worth noted that we add three penalties to cost function $gSocre$ when the agent performs turning actions, driving backwards, and switching the moving direction. The heuristic function design and analytic expansion technique of our method are the same as the original hybrid-state A*.



(a) Sequential car-like CBS (b) A fail case

Fig. 4: (a) Sequential CL-CBS. (b) A simple fail case for sequential CL-CBS. The blue agent cannot reach its goal when the grey agent planned in former batch arrives at its goal (which locates between the obstacles) before the blue one passing through those obstacles.

C. Sequential Car-like CBS

In car-like CBS solver, the high-level conflict search tree is proven to be both optimal and complete [33]. However, spatiotemporal Hybrid-State A* search only ensures completeness but lacks theoretical optimality guarantees. Thus, the whole car-like CBS solver is complete and near-optimal.

As a result of expanding workspace from discrete space to continuous space, the low-level search time 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 [34]. 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-CL-MAPF 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.

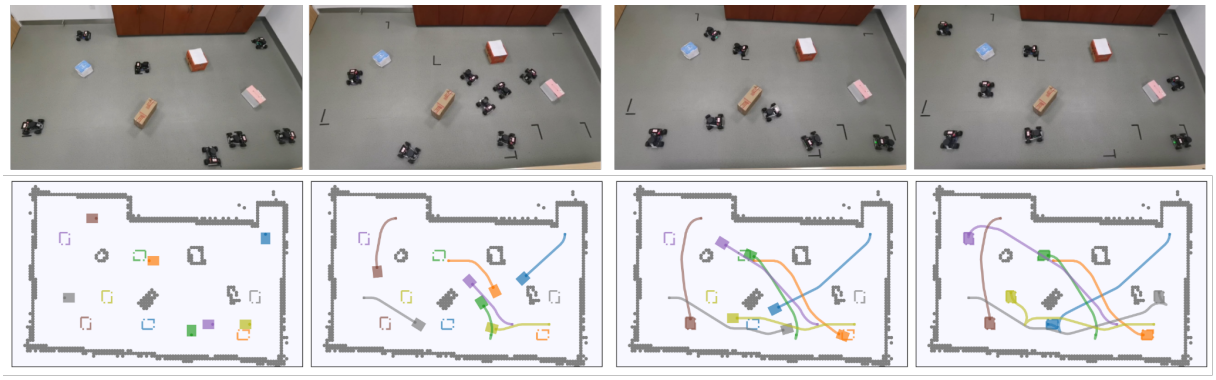


Fig. 5: Glimpses of field tests. Snapshots in the upper row show four keyframes during an experiment, and pictures in the lower row plot the trajectories agents have driven at the corresponding frame. Best viewed in color.

The procedure of this method is exhibited in Fig.4(a). The agents are divided into three batches. The paths in grey 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 overmuch agents at the same time, the sequential method shortens searching time by nearly an order of magnitude in our experiment. However, it should also be noted that this sequential method sacrifices the completeness guarantee of Car-like CBS. A simple fail case is shown and explained in Fig.4(b).

V. Experiments

In this section, we implement Car-like CBS solver in C++ using *boost* library for math calculation and OMPL library to produce Reeds-Shepp paths. The program are executed on a PC running Ubuntu 16.04 with Intel i7-8700@3.20GHz and 8G RAM. We test our car-like CBS solver in both simulated and physical environment and the experiment video is available in the attachment of this paper.

A. Simulated Experiment

Since it is the first time we proposed CL-MAPF problem, there are few direct competitors for our experiments. We adopt two methods, a centralized one and a decentralized one, acting as the baseline of our experiment. *i)* The centralized method is model predictive control with CBS (CBS-MPC) based on [35]. It applies original CBS solver to provide guide paths for each agent and use MPC to generate final trajectories. *ii)* The decentralized baseline we use is plain hybrid-state A* (HA*) for a single agent, without the high level-search tree and spatiotemporal constraints.

1) Benchmark: The classic MAPF benchmark like DAO map sets are all 4-neighbor grid-based benchmark and cannot be used for CL-MAPF problem. Therefore we generate a novel CL-MAPF benchmark for simulation experiment.

The benchmark contains two scenarios involving workspace with and without obstacles. Each scenario includes 25 map sets. These map sets possess three type of map size (300×300m,100×100m,50×50m) and distinct agent numbers from 5 to 100. Every map set has 60 unique instances, and the whole benchmark contains 3000 different instances.

For each instance in the benchmark, *i)* it describes a continuous $\mathbb{R}^3$ workspace; *ii)* the start and goal states of agents are guaranteed not collide with each other (for agents under 5×5m 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 circle obstacles with 1m radius and the entire obstacle region occupies 1% map area. We use *300x300_agents80_obs* to denote mapset with 80 agents in a 300×300m workspace with obstacles.

Based on the benchmark, we evaluate the performance of our method compared with two baseline algorithm HA* and CBS-MPC.

TABLE I: Comparision with CBS-MPC

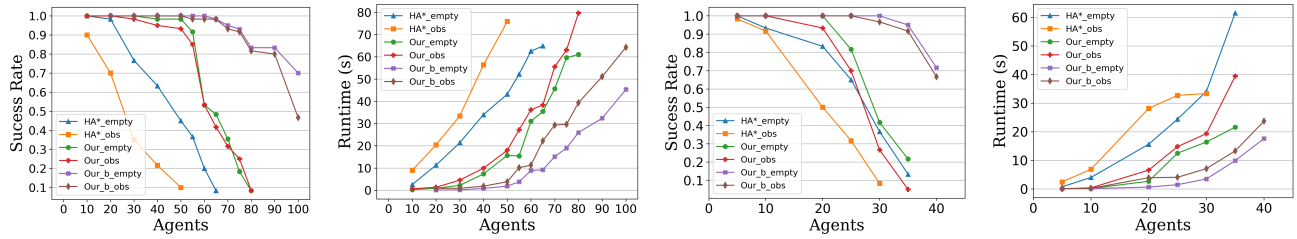
Mapsize(m^2)	Agents	Method	Empty / Obstacles	
			Makespan(m)	Collision Times
300x300	50	CBS-MPC	206.6/205.4	9.25/10.09
		Ours	**179.1/178.8**	**0/0**
100x100	30	CBS-MPC	71.92/67.90	9.43/9.96
		Ours	**70.73/67.25**	**0/0**
50x50	20	CBS-MPC	36.38/35.54*	7.28/9.26
		Ours	48.80/52.96	**0/0**

*Though having a smaller makespan, we don't consider CBS-MPC performs better due to the collisions in the solution.

2) Comparision with CBS-MPC: We assume agents in the experiments are homogeneous with the following parameters: the shape of agents is 2×3m as $L_f = 2m, L_b = 1m$, the maximum speed for both forward and backwards $v_{max} = 2m/s$, the minimum turning radius $r = 3m$. We set the runtime limit for each instance as 90 seconds and compare the average makespan and collision times in the solution of each map set. The results are shown in Table I.

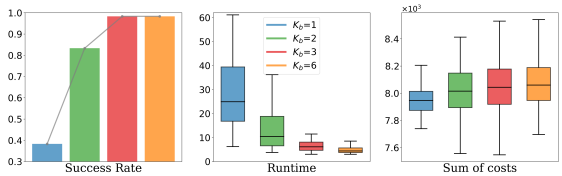
Our method outperforms CBS-MPC in almost all map sets. The average collision times of the solution by CBS-MPC are between 7 to 10 under different map sizes, which are all considered as failures. The agents would collide with each other when using MPC following their guided paths since the kinematic constraints are not considered in the high-level CBS. Our proposed method, however, has no collisions in any of the map sets and performs smaller makespans in 300×300m and 100×100m map sets as well.

3) Comparision with HA:* We assess then how our method scales to a large number of robots compared to the decentralized method HA*, seeing that CBS-MPC have

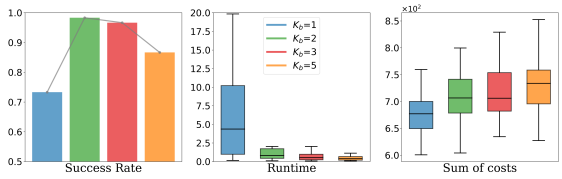


(a) Success rate on 300x300 mapset (b) Runtime on 300x300 mapset (c) Success rate on 100x100 mapset (d) Runtime on 100x100 mapset

Fig. 6: Scalability Comparision with HA*



(a) Results on 300x300_agents60_obs mapset



(b) Results on 50x50_agents15_empty mapset

Fig. 7: Sequential CL-CBS experiment results

collisions between agents in most cases. The sequential version of our method (with K_b=2) also participates in the comparison. We limit the computation time of each instance to 120 seconds, and the results are shown in Fig.6.

In 300×300m map sets, our CL-CBS approach outperforms HA* in both success rate and runtime. Our original method successfully solves over 50% of instances containing 60 agents in both empty and obstructive scenarios and the sequential version solves instances including 100 agents with over 70% success rate. On the contrary, HA* barely works out instances over 50 agents. The running time of HA* is more than twice as long as ours in the same map set. As for 100×100m map sets, our method solves instances up to 40 agents in 30 seconds while HA* costs over 60 seconds for instances containing 30 agents with success rate below 10%.

4) Comparision with sequential version: As we proposed the sequential version of our method in IV-C, we evaluate the performance of the original method and its sequential version with different batch numbers K_b. We perform a comparative test in two map sets: a large map with obstacles (300x300_agents60_obs) and a small empty one (50x50_agents15_empty). The time limit for each instance is 60 seconds. The results are shown in Fig.7.

In the large map set, our original method (K_b=1) achieves merely 38.3% success rate and costs 31.6 seconds runtime at average. Fig.7(a) shows the success rate increases rapidly to 98.3%, and the average runtime decreases to 6.8 seconds when we divide agents into three batches. The runtime reduced to 4.4 seconds when K_b=6, which is almost an order of magnitude smaller than the original method. Meanwhile, the average sum of costs has only increased by 2.5%.

As we mentioned before, the sequential method sacrifices completeness guarantee and may lead to some fail cases. In the 50×50m map set, when K_b increases, the success rate first rises to 98.3% and then falls to 86.6% as Fig.7(b). This is for the reason that previous agents are not aware of the existence of subsequent agents during the planning and may block their paths to the goal.

B. Field Test

We conduct field tests using seven 23×20 cm Ackermann-steering robots produced by WeTech as shown in Fig.1. The robot is able to move at 0.3m/s, and the minimum turning radius is 26cm. All the robots are equipped with a 2D Lidar from Slamtec, a 5-megapixel camera, and a Raspberry Pi 4 running Ubuntu 18.04 and ROS Melodic. We use a PC laptop running ROS as the central computing station to communicate with all agents use 2.4GHz Wifi.

Experiments are performed in a 5×3m room, including empty and obstructed scenarios. Before each experiment, we create a 2D occupancy map using lidar by gmapping algorithm. After appointing start and goal states for all agents, a solution is computed on the laptop. We then transfer paths to a sequence of velocity commands and send them to agents for execution. Amcl package is used when robots are running so that we can get the trajectory. The snapshot of the field test is shown in Fig.5, and full experiments are presented in the supplemental video.

VI. Conclusion and Futur Works

In this paper, we formalize the CL-MAPF problem that considers the kinematic and spatiotemporal constraints of car-like robots. We present CL-CBS, an efficient hierarchical search-based algorithm that is correct and complete for solving the problem. Experiments in simulated and physical environments show that our method outperforms baseline solvers in terms of both scalability and solution quality. One of the directions of future research is extending our method in order to plan for holonomic and nonholonomic agents under the same scenario, and another one is applying the proposed approach to combined target-assignment and path-finding (TAPF) problem.

REFERENCES

[1] J. Yu and 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 and 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*, 2015.

[4] E. Erdem, D. G. Kisa, U. Oztok, and P. Schüller, "A general formal framework for pathfinding problems with multiple agents," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[5] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. 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, and 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 and 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, and B.-H. Lee, "A computational interactive approach to multi-agent motion planning," *International Journal of Control, Automation, and Systems*, vol. 5, no. 3, pp. 295–306, 2007.

[10] M. Čáp, P. Novák, A. Kleiner, and M. Seleckỳ, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE transactions on automation science and engineering*, vol. 12, no. 3, pp. 835–849, 2015.

[11] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[12] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and 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, and 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, and 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, and 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, and 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, and 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, and P. Gulden, "Steady delivery: Wireless local positioning systems for tracking and autonomous navigation of transport vehicles and mobile robots," *IEEE Microwave Magazine*, vol. 18, no. 6, pp. 26–37, 2017.

[19] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[20] L. Wen, J. Yan, X. Yang, Y. Liu, and 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.

[21] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal, "Cobots: Robust symbiotic autonomous mobile service robots," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[22] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *arXiv preprint arXiv:1906.08291*, 2019.

[23] K. Yakovlev and A. Andreychuk, "Any-angle pathfinding for multiple agents based on sipp algorithm," *arXiv preprint arXiv:1703.04159*, 2017.

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[25] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings 1999 ieee international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 1. IEEE, 1999, pp. 341–346.

[26] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 6, pp. 1725–1740, 2014.

[27] J. Van den Berg, M. Lin, and 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.

[28] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.

[29] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 360–366.

[30] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 404–420, 2018.

[31] L. Marin, M. Vallés, A. Soriano, A. Valera, and P. Albertos, "Event-based localization in ackermann steering limited resource mobile robots," *IEEE/ASME Transactions on Mechatronics*, vol. 19, no. 4, pp. 1171–1182, 2013.

[32] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[33] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7643–7650.

[34] J. Park, J. Kim, I. Jang, and 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.

[35] R. R. Negenborn, B. De Schutter, and J. Hellendoorn, "Multi-agent model predictive control: A survey," *arXiv preprint arXiv:0908.1076*, 2009.