2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Decentralized Lifelong Path Planning for Multiple Ackerman Car-Like Robots

Teng Guo Jingjin Yu

***Abstract*— Path planning for multiple non-holonomic robots in continuous domains constitutes a difficult robotics challenge with many applications. Despite significant recent progress on the topic, computationally efficient and high-quality solutions are lacking, especially in lifelong settings where robots must continuously take on new tasks. In this work, we make it possible to extend key ideas enabling state-of-the-art (SOTA) methods for multi-robot planning in discrete domains to the motion planning of multiple Ackerman (car-like) robots in lifelong settings, yielding high-performance centralized and decentralized planners. Our planners compute trajectories that allow the robots to reach precise $SE(2)$ goal poses. The effectiveness of our methods is thoroughly evaluated and confirmed using both simulation and real-world experiments.**

I. Introduction

The rapid development of robotics technology in recent years has made possible many revolutionary applications. One such area is multi-robot systems, where many large-scale systems have been successfully deployed, including, e.g., in warehouse automation for general order fulfillment [1], grocery order fulfillment [2], and parcel sorting [3]. However, upon a closer look at such systems, we readily observe that the robots in such systems largely live on some discretized grid structure. In other words, while we can effectively solve multi-robot coordination problems in grid-like settings, we do not yet see applications where many non-holonomic robots traverse smoothly in continuous domains due to a lack of good computational solutions. Whereas many factors contribute to this (e.g., state estimation), a major roadblock is the lack of efficient computational solutions tackling the lifelong motion planning for non-holonomic robots in continuous domains.

Toward clearing the above-mentioned roadblock, in this work, we proposed two algorithms specially designed to solve static/one-shot and lifelong path/motion planning tasks for Ackerman car-like robots. The basic idea behind our methods is straightforward: to enable the adaptation of discrete search strategies for car-like robots, we use a small set of fixed but representative motion primitives to transition the robots' states. These fixed motion primitives, when properly put together, yield near-optimal trajectories that "almost" connect the starts and goals for robots. Some local adjustments are then used to complete the full trajectory. The first algorithm and our main contribution in this research, *Priority-inherited Backtracking for Car-like Robots* (PBCR) adapts a decentralized strategy that leverages search-prioritization strategies from Priority Inheritance and Backtracking (PIBT) [4]. The second algorithm, *Enhanced Conflict-based search for Car-like Robots* (ECCR), is a centralized method building on the principles of Enhanced Conflict-Based Search (ECBS) [5] and CL-CBS [6], the car-like robot extension of (basic) conflict-based search [7]. We further boost algorithms' success rates by introducing carefully designed, effective heuristics which also reduce the occurrence of deadlocks. Thorough simulation-based evaluations confirm that our methods deliver scalable SOTA performances on many key practical metrics. While our centralized methods tend to find shorter trajectories due to their access to global information, our decentralized method produces better scalability, yielding a higher success rate.

G. Teng, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. Emails: {teng.guo, jingjin.yu}@rutgers.edu. This work was supported by NSF award IIS-1845888 and an Amazon Research Award.

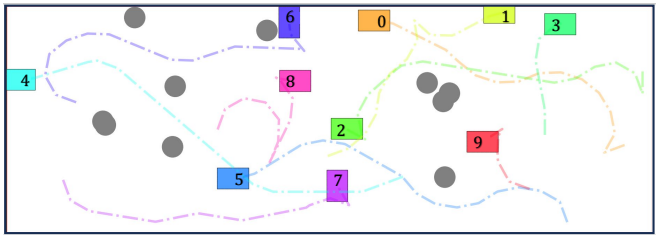


Fig. 1. A simulated example on a 60×30 map with 10 obstacles and robots. The collision-free trajectories for the car-like robots are shown.

Related Work. Multi-Robot Path Planning (MRPP) has garnered extensive research interests in robotics and artificial intelligence in general. The graph-based MRPP variant, also known as Multi-Agent Path Finding (MAPF) [8], is to find collision-free paths for a set of robots within a given graph environment. Each robot possesses a distinct starting point, and a goal position, and the challenge lies in determining paths that ensure their traversal from start to goal without collisions. It has been proven many times over that optimally solving the graph-based problem to minimize objectives like makespan or cumulative costs is NP-hard; see, e.g., [9], [10].

Computational methods for MRPP can be broadly classified into two categories: *centralized* and *decentralized*. Centralized solvers operate under the assumption that robot paths can be computed centrally and subsequently executed with minimal coordination errors. On the other hand, decentralized solvers capitalize on the autonomy of individual robots, enabling them to calculate paths independently while requiring coordination for effective decision-making. Centralized solutions often involve reducing MRPP to well-established problems [9], [11], [12], employing search algorithms to explore the joint solution space [5], [7], [13]–[15] or apply human-designed rules for coordination and collision-avoidance [16], [17]. Decentralized approaches [4],

[18] leverage efficient heuristics to address conflicts locally, bolstering their success rates. Machine learning and reinforcement learning methods for MRPP have also started to emerge, with researchers putting forward data-driven strategies to directly learn decentralized policies for MRPP [19], [20]. Nevertheless, we note that the paths generated by graph-based MRPP algorithms cannot be directly applied to physical robots due to their disregard for robot kinematics.

As autonomous driving gains momentum, interest in path planning for multiple car-like vehicles is also steadily rising [21], [22]. This has led to the development of new methods, including adaptations of CBS to car-like robots (CL-CBS) [6], prioritized trajectory optimization [23], sampling-based techniques [24], and optimal control strategies [25]. Decentralized approaches have been introduced as well, such as B-ORCA [26] and ϵCCA [27], both stemming from adaptations of ORCA [28] designed for car-like robots. These methods offer faster computation compared to centralized methods. However, their reliance on local information means they cannot provide a guarantee that robots will successfully reach their destinations. Moreover, they are prone to deadlock issues and tend to achieve much lower success rates, particularly in densely populated environments.

Organization. The rest of the paper is organized as follows. Sec. II covers the preliminaries, including the problem formulation. In Sec. III-IV, we demonstrate our algorithms in detail. In Sec. V, we conduct evaluations on the proposed methods in static settings and lifelong settings and discuss their implications. We conclude in Sec. VI.

II. Problem Formulation

A. Multi-Robot Path Planning for Car-Like Robots

A static instance of this problem is specified as $(\mathcal{W}, \mathcal{S}, \mathcal{G})$, where $\mathcal{W}$ constitutes a map with dimensions $W \times H$, housing a set of obstacles $\mathcal{O} = \{o_1, ..., o_{n_o}\}$. $\mathcal{S} = \{s_1, ...s_n\}$ defines the initial configurations of n robots within the workspace, and $\mathcal{G} = \{g_1, ...g_n\}$ represents the corresponding goal configurations.

Each robot's $SE(2)$ configuration, denoted as $v_i = (x_i, y_i, \theta_i)$, comprises a 3-tuple of position coordinates and the yaw orientation θ_i. The car-like robot is modeled as a rectangular shape with length ℓ and width w. The subset of $\mathcal{W}$ occupied by a robot's body at a given state v is denoted by $\Gamma(v)$. The motion of the robot adheres to the Ackermann-steering kinematics (see Fig. 2(a)):

$$\dot{x} = u\cos\theta, \quad \dot{y} = u\sin\theta, \quad \dot{\theta} = \frac{u}{\ell_b}\tan\phi, \tag{1}$$

in which $u \in [-u_m, u_m]$ is the linear velocity of the robot, $\phi \in [-\phi_m, \phi_m]$ is the steering angle. These are the control inputs. ℓ_b is the wheelbase length, i.e., the distance between the front and back wheels.

To make planning more tractable, we discretize into intervals Δt. The goal is to ascertain a feasible path for each robot i, expressed as a state sequence $P_i = \{p_i(0), ...p_i(t), ...p_i(T)\}$ that adheres to the following constraints: (i) $p_i(0) = s_i$ and $p_i(T) = g_i$; (ii) $\forall t \in [0, T], \forall i \neq j, \Gamma(p_i(t)) \bigcap \Gamma(p_j(t)) = \emptyset$; (iii) The path follows the Ackermann-steering kinematic model. The time interval is kept small during the discretization process. This choice ensures that the distance moved within each step remains smaller than the size of the robot. Consequently, the need to account for swap conflicts [8] is eliminated.

Trajectory quality is evaluated using the following criteria: (i) Makespan: $\max_{1\leq i\leq n} len(P_i)/u_{max}$; (ii) Flowtime: $\sum_{1\leq i\leq n} len(P_i)/u_{max}$, where $len(P_i)$ denotes the length of trajectory P_i.

B. The Lifelong Setting

In a lifelong setting, we assume an infinite stream of tasks for each robot. Upon completing the current task, a robot immediately receives a new target. In this scenario, evaluating the system's efficiency often relies on throughput—the number of tasks accomplished (or goal states arrived) within a specified number of timesteps.

III. Priority-Inherited Backtracking for Car-Like Robots (PBCR)

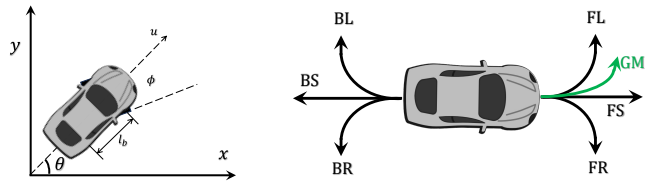
In this section, we propose a *decentralized* method called PBCR for car-like robots, adapting the prioritization mechanism of the decentralized MRPP algorithm PIBT [4]. In conjunction the introduction of PBCR, we also describe the general methodology we adopt to plan continuous trajectories for non-holonomic robots that is generally applicable, provided that the optimal trajectories connecting the robot can be compactly represented using a few motion primitives.

A. Motion Primitives

To render planning for the continuous system feasible, we constrain the robot's possible state transitions within the discretized time interval Δt, forming a set of *motion primitives*. Our motion primitives $\mathcal{M}$ are defined similarly as done in [6], [23], which contain a total of seven actions: 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(b). For the first six motion primitives, we assume that the robot maintains a constant velocity of u_m throughout a single time interval Δt. When the robot turns, we assume it is pivoting using the maximum steering angle ϕ_m and then tracing an arc with a turning radius r_m. This arc has a $u_m \Delta t$ length.

To ensure a robot can reach its goal state for PBCR, one additional *greedy motion primitive* (GM) is added to $\mathcal{M}$. This greedy motion primitive is derived by truncating the first segment of length $u_m \Delta t$ from the shortest path connecting the current state to the goal state for a single robot. In the absence of obstacles on the map, this shortest path corresponds to the (optimal) Reeds-Shepp [29] path while if obstacles are present, the path can be determined using the vanilla (non-spatio-temporal) hybrid A* algorithm. It's important to recognize that this greedy motion primitive could be identical to other motion primitives, resulting in the

same subsequent state as those alternatives. In such cases, we opt to retain solely the greedy one.



(a) (b)

Fig. 2. (a) Ackermann steering kinematic model. (b) The predefined motion primitives. We have at most 8 motion primitives, which are forward max-left (FL), forward straight (FS), forward max- right (FR), backward max-left (BL), backward straight (BS), backward max-right (BR), wait, and greedy motion primitive (GM)

B. Algorithm Skeleton

PBCR is outlined in Alg. 1-2, at the core of which lies the `PIBTLoop`, which orchestrates the decision-making process for the robots at each time step t. `PIBTLoop`'s primary objective is to determine each robot's next motion and succeeding state without encountering collisions.

`PIBTLoop` initializes two essential sets: UNDECIDED and OCCUPIED. The UNDECIDED set keeps track of robots that have yet to finalize their next actions, while the OCCUPIED set stores the information about occupied states, preventing multiple robots from attempting to occupy the same space simultaneously. At the outset of each iteration, the algorithm updates the priorities of all robots based on relevant factors. The priorities are crucial in determining which robot takes precedence in decision-making. This step ensures that the higher-priority robots have their actions determined earlier, facilitating swift and efficient navigation. PBCR's decision-making process revolves around an iterative approach. The algorithm enters a loop that continues until all robots in the UNDECIDED set have their actions determined. The robot with the highest priority within the loop is selected from the UNDECIDED set. This robot is designated as robot i for the current iteration.

Algorithm 1: Priority selection at time step t

```
Function PIBTLoop():
    UNDECIDED ← R(t)
    OCCUPIED ← ∅
    update all priorities
    while UNDECIDED! = ∅ do
        i ← the robot with the highest priority in
          UNDECIDED
        PIBT(i, NONE, NONE)
    end
```

The `PIBT` function is invoked for the selected robot i, along with placeholders for two parameters: the parent robot j which robot i is inherited from, and the potential succeeding state v_j of the parent robot. This function serves as the primary interface for the robot's decision-making. It evaluates potential actions considering the robot's current state, goals, obstacles, and the presence of other robots. Based on these evaluations, the robot's next action is determined to ensure collision-free and goal-oriented navigation.

Algorithm 2: PIBT function

```
1  Function PIBT(i, j, v_j):
2      UNDECIDED ← R(t) − i
3      NbrUndecided ←
         the neighboring robots of i that remain undecided
4      NbrOcc ←
         the occupied states of the neighboring decided robots
5      C ← {ValidSuccState(p_i(t), m_i) | m_i ∈ M_i}
6      while C ≠ ∅ do
7          v_i ← argmax_{v∈C} Q_i(v) and remove v_i from C
8          if FindCollision(v_i, NbrOcc ⋃ {p_j(t), v_j})
             then
9              continue
10         end
11         OCCUPIED.add(v_i)
12         if ∃k ∈ NbrUndecided such that
             FindCollision(p_k(t), v_i)= true then
13             if PIBT(k, i, v_i) is valid then
14                 p_i(t + 1) ← v_i
15                 return valid
16             end
17             else
18                 OCCUPIED.remove(v_i)
19             end
20         end
21     end
22     UNDECIDED.add(i)
23     return invalid
```

The `PIBT` function plays a pivotal role within PBCR, encapsulating the decision-making process for an individual robot. This function is called iteratively for each robot to compute its next action, considering its current state, goals, and interactions with neighboring robots. The function identifies the set UNDECIDED, which comprises all robots yet to finalize their actions, and marks robot i as decided temporarily. Additionally, NbrUndecided contains neighboring robots of i that remain undecided, and NbrOcc holds the occupied states of the neighboring robots that have already determined their actions. These sets provide crucial contextual information to facilitate informed decision-making. The function computes a set C of potential future states v_i for the current robot i. These states are generated based on all feasible actions m_i available to robot i at its current state $p_i(t)$. The goal is to explore various potential actions to lead to a successful next state for r_i. The function selects the potential state v_i within a loop that maximizes the robot's utility function $Q_i(v)$ from the set C. This action selection aims to optimize the robot's decision by choosing the most promising action according to the utility criterion. However, before finalizing the action, the function checks for collisions between the selected v_i and the occupied states of neighboring robots, as well as the state of the parent robot j (if it is not NONE) and its potential succeeding state v_j. If a collision is detected, the function continues to the next iteration of the loop, considering alternative potential actions.

If no collisions are found for the chosen v_i, v_i is added to the OCCUPIED set, indicating the robot intends to occupy this state. The function then checks if an undecided neighboring robot k will collide with v_i. If such a robot k is found, the function recursively calls `PIBT` for robot k with the parent

robot i and the tentative state v_i as the parameters. If the resulting action is valid, robot i updates its next state $p_i(t+1)$ to v_i, and the function returns "valid." If PIBT for robot k fails, the state v_i is removed from OCCUPIED. After evaluating all potential actions and considering interactions with neighboring robots, robot i remains in the UNDECIDED set. The function returns "invalid" if a valid action cannot be determined. Otherwise, it returns "valid" along with the updated next state $p_i(t+1)$ for robot i.

We note that PBCR's one-step planning and execution nature makes it applicable to static and lifelong scenarios.

C. Performance-Boosting Heuristics

Multiple heuristics are introduced to boost the performance of PBCR, while some are basic, others are involved.

Distance heuristic. We use the max of the holonomic cost with obstacles, the length of the shortest Reeds-Shepp path between two states, and 2D Euclidean distance as our distance heuristic function (DistH) similar to [6], [30]. This distance heuristic is admissible.

Priority heuristic. PBCR constitutes a single-step, priority-driven planning approach that necessitates the ongoing adjustment of each robot's priority at each time step. Initially, the priority assigned to each robot is determined based on the number of time steps that have transpired since its preceding task was updated. The robot with more time since its previous goal update receives a higher priority ranking. In static scenarios, the elapsed time is reset to zero whenever a robot reaches its designated goal state. This reset mechanism prevents robots that have achieved their goal state from obstructing the progress of other robots. When multiple robots share the same elapsed time, prioritization is resolved by favoring those robots with a larger heuristic distance value to their respective goal states. This heuristic principle finds widespread application in the realm of prioritized planning [31] to improve the success rate.

The Q-function. The Q-function is a critical tool for assessing the optimal choice of a motion primitive in guiding the robot from its current state toward its goal state. PIBT algorithm employs the single-agent shortest path length from the current vertex to its goal vertex, neglecting inter-agent collisions, as the basis for its evaluation function. This approach is feasible for discrete 2D graphs since these shortest path lengths can be pre-calculated and stored in a table. Nevertheless, this approach is impractical for the state space we're addressing with car-like robots. Firstly, the state space is infinite. Secondly, determining a single-robot shortest trajectory from a state to a goal state using hybrid A* for evaluating all the motion primitives is both time-intensive and incomplete. For these reasons, we consider employing the distance heuristic function discussed earlier as well as the action cost, denoted as

$$Q_i'(v) = -\texttt{DistH}(v, g_i) - \lambda \texttt{Cost}(p_i(t), v) \tag{2}$$

where λ is a weight and $\texttt{Cost}(p_i(t), v)$ returns the action cost from state $p_i(t)$ to v. We use the action cost similar to [6] where backward motions, turning, and changes of directions will receive additional penalty cost. However, it's important to note that the distance heuristic is not a "reachable" heuristic in the context of PBCR. A heuristic is labeled as "reachable" for PIBT-like algorithms if, while ignoring inter-robot collisions, a solitary robot is guaranteed reach its goal state by consistently choosing the "best" action with the maximum Q value at each timestep. In discrete MRPP, the single-agent shortest path length qualifies as a "reachable" heuristic. We mention that while Manhattan distance is a reliable "reachable" heuristic when the map is obstacle-free, this is not guaranteed when obstacles are present. In scenarios with obstacles, a robot directed by the Manhattan distance heuristic might become entrapped in local minima, obstructing its path to the goal state. Our case showed DistH is not a "reachable" heuristic either. Despite being admissible, it lacks the required precision. Notably, the assertion that a greedy motion primitive should yield the highest Q value among the possibilities isn't consistently upheld when using $Q_i(v) = -\texttt{DistH}(v, g_i)$. This discrepancy easily leads the robot into a state of stagnation. Furthermore, we found that this greedy heuristic could also lead to deadlocks. An example can be seen in Fig. 3.

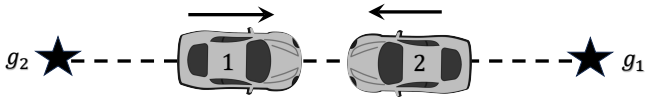


Fig. 3. An illustrative case highlighting the potential occurrence of deadlocks due to the exclusion of the count-based heuristic is as follows: In this scenario, two robots, labeled as robot 1 and robot 2, travel in opposite directions along a linear path. When robot 2 yields to robot 1 in PBCR, a predicament arises. In this situation, the motion primitives FS, FR, and FL lead to collisions, compelling robot 2 to consistently opt for the motion primitive BS. This choice is driven by BL and BR, involving additional turning penalties. A dynamic shift occurs in priorities upon robot 1's successful arrival at its designated goal state. Robot 1 is assigned a lower priority than robot 2 and consequently yields to the latter. Similarly, robot 1 consistently selects BS using the greedy strategy. Consequently, an unending cycle emerges, entangling robot 1 and robot 2 in an indefinite sequence of movements

We present a novel count-based exploration heuristic to address the challenge, inspired by [32]. This heuristic is specifically designed to surmount the issue of local minima arising from inaccuracies in distance heuristics and deadlocks and to foster exploration of uncharted territories. To achieve this, we incorporate supplementary penalties for states that have been visited, aiming to encourage robots to break free from local minima and venture into unexplored regions. Each state $v = (x, y, \theta)$ is mapped to its nearest discrete state $\widetilde{v} = (\lfloor \frac{x}{\delta x} \rfloor, \lfloor \frac{y}{\delta y} \rfloor, \lfloor \frac{\theta}{\delta \theta} \rfloor)$. This mapping facilitates the tallying of occurrences using a hash table $\mathcal{H}_i$ for each robot, where δx, δy, and $\delta \theta$ delineate the state-space resolution.

When a robot i traverses state v at a given timestep t, the count $\mathcal{H}_i(\widetilde{v})$ is incremented by one. In addition, we introduce bonus rewards to incentivize the selection of the greedy motion primitive. The resultant Q-function is:

$$Q_i(v) = Q_i'(v) + \alpha Gr(v) - \beta \mathcal{H}_i(\widetilde{v}) \tag{3}$$

Here, α and β represent positive weight parameters. If the state v results from a greedy motion primitive, $Gr(v) = 1$;

otherwise, $Gr(v) = 0$. To accommodate the requirement that robots should remain at their goal states upon arrival, we refrain from applying penalties to states near the goal states. As a result, when $v = g_i$ (the goal state of robot i), we set β to zero. We mention that such a Q function could be "reachable" if we properly choose the weight parameters. Moreover, by implementing the count-based heuristic, which compels robots to explore distinct motion patterns for collision avoidance, the deadlock issues can be nicely achieved.

IV. Centralized ECCR

We enhance the CL-CBS algorithm by substituting the conventional low-level spatiotemporal hybrid state A* planner with focal hybrid state A* search [5]. This refined approach is named ECCR. In comparison to CL-CBS, the ECCR method guides the low-level planner to discover trajectories within a bounded suboptimality ratio while encountering fewer potential conflicts with other robots. This reduction in conflicts subsequently results in a significantly lower number of high-level expansions.

For the purpose of adapting ECCR to lifelong scenarios, characterized by the necessity for frequent online replanning, we incorporate the windowed version of focal search in the low-level planning process [33], [34]. Specifically, during path planning, we only address conflicts that arise within a window of ω steps. This adjustment effectively decreases the runtime of the ECCR algorithm during the planning phase. Replanning occurs at intervals of every ω steps and also when robots reach their current goals and receive new tasks.

V. Evaluation

In this section, we evaluate the proposed algorithms in static scenarios and lifelong scenarios. All methods are implemented in C++. The source code can be found in https://github.com/GreatenAnoymous/CarLikePlanning. All experiments are performed on an Intel® Core™ i7-6900K CPU at 3.0GHz in Ubuntu 18.04LTS. In the simulation, the following parameters are used: $w = 2$, $l = 3$, $l_b = 2$, $\delta x = \delta y = 2$, $\delta\theta = 40.1°$, $u_m = 2$, $\phi_m = 40.1°$, $r_m = 3$, $\Delta t = r_m \delta\theta / u_m$.

A. Static Scenarios

In this section, we assess the performance of the algorithms on a grid of dimensions 100×100, both with and without obstacles. For the scenarios involving obstacles, we create 50 circular obstacles with a radius of 1 unit length and place them randomly on the map. For each value of n, we generate 50 distinct instances, ensuring that the states of the robot do not overlap with each other and with the obstacles in start and goal configurations. A time limit of 60 seconds is imposed on each instance. The success rate is determined by tallying the instances each algorithm successfully solves within the time limit. Additionally, we evaluate the average runtime (in seconds), makespan, and flowtime across the solved instances. Our evaluation includes a comparison with two reference algorithms: firstly, the prior centralized algorithm known as CL-CBS, and secondly, the decentralized SHA* algorithm as described in [6]. We emphasize that all algorithms use identical predefined motion primitives, except that GM is exclusively used for PBCR.

A suboptimality ratio of 1.5 is selected for ECCR. Three variants of PBCR are evaluated. The first, PBCR (v0), employs the Q-function described in (3) without incorporating the count-based exploration heuristic. Both PBCR (v1) and PBCR (v2) leverage the count-based exploration heuristic to resolve local minima and deadlocks. In the case of PBCR (v1), we opt to clear the hash table $\mathcal{H}_i$ and reset the associated counts to zero whenever robot i reaches its designated goal state. Conversely, for PBCR (v2), the visiting history is not cleared. For all variants of PIBT, the maximum time step is constrained to 500. And we set $\lambda = 0.3$, $\alpha = \beta = \lambda \texttt{Cost}(p_i(t), v)$.

Detailed evaluation results are presented in Fig.4 through Fig.6. Compared to CL-CBS, ECCR showcases significantly enhanced scalability and success rates due to its capability to expand a notably smaller number of high-level nodes. Notably, the solution quality of ECCR closely approaches that of CL-CBS. On the flip side, the PBCR variants excel in terms of runtime efficiency, resulting from their decentralized nature and one-step planning strategy. They can resolve instances involving 60 robots in as little as 4 seconds. Among these variants, PBCR featuring the count-based exploration heuristic achieves an elevated success rate. Conversely, without the count-based exploration heuristic, PBCR (v0) falters in instances with obstacles, revealing its limitations. However, PBCR (v2) successfully tackles 93% of cases involving 60 robots within 4 seconds, excelling in challenging scenarios—a substantial improvement over other variants and the previous decentralized SHA* approach.

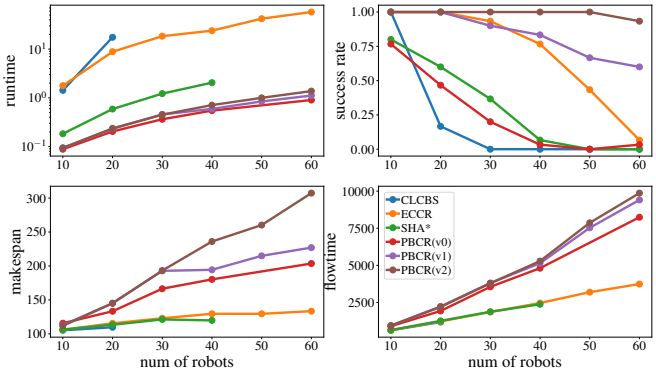


Fig. 4. Evaluation results on a 100×100 empty map for a varying number of robots.

It's crucial to emphasize that all PBCR variants do not fail due to time constraints but rather due to surpassing the maximum timestep. For those failed instances, PBCR with count-based heuristic still can guide more than 90% of the robots to their goal states as shown in Fig. 6. The contrast in success rates between PBCR (v2) and PBCR (v1) implies that retaining the visiting history until completion, instead of resetting it upon each robot's arrival at its goal state, is more effective. This arises from the fact that in the context of PBCR, when a robot reaches its designated goal, it often

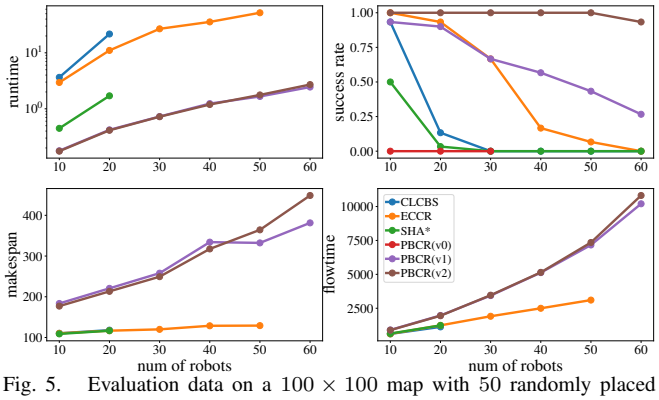
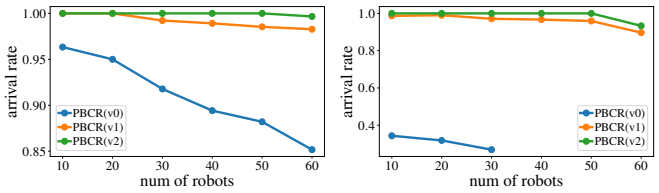


Fig. 5. Evaluation data on a 100 × 100 map with 50 randomly placed obstacles for a varying number of robots.



(a) (b)

Fig. 6. The average percentage of robots arrived at their goal state for each PBCR variant on the 100 × 100 maps for varying numbers of robots. (a) without obstacles (b) with obstacles.

has to temporarily vacate its goal position to accommodate other robots that must traverse it. This frequent shifting in and out of the goal state can potentially lead to repetitive cycles and hinder progress. Maintaining the visiting history mitigates this issue by preventing the robot from becoming trapped in an endless loop of entering and leaving the goal, consequently boosting the overall success rate. One drawback of PBCR lies in its trajectory quality. This stems from the absence of global information and the inherent nature of one-step planning, leading to longer planned trajectories compared to centralized algorithms such as ECCR and CL-CBS. This effect is particularly pronounced when dealing with a large number of robots.

B. Lifelong Scenarios

In this section, we subject both ECCR and PBCR to testing within lifelong settings. We adopt a 50 × 50 map configuration, both with and without obstacles. For scenarios involving obstacles, a set of 10 obstacles is distributed randomly. For each value of n, we randomly generate 20 unique instances. In each instance, both the initial configurations and 4000 goal states, randomly generated, are allocated to each robot. Throughout the simulation, we assume a lack of a priori knowledge on the part of the robots regarding their subsequent tasks. In each instance, we define a maximum of 5000 simulated steps and set a runtime limit of 600 seconds. Each robot has many tasks that cannot be feasibly completed within the designated maximum steps. Regarding ECCR, a suboptimality ratio of 1.5 is established, while a window size ω of 5 is employed. Given that new goals are allocated to robots upon completing current tasks, the visiting history is consistently cleared. This decision is driven by the fact that the visiting experience only relates to the robot's preceding task. PBCR (v0) is excluded from consideration owing to its poor performance in addressing issues related to deadlocks and stagnation. The outcome of our evaluations is depicted in Fig. 7. PBCR achieves significantly lower execution times than ECCR, rendering it suitable for handling large-scale scenarios and real-time planning. Nonetheless, it accomplishes a smaller number of tasks.

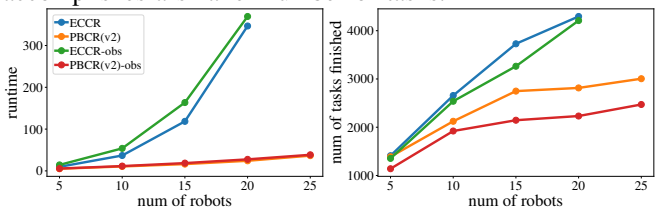


Fig. 7. Runtime and the average number of tasks finished within given timesteps in lifelong scenarios for varying number of robots. The term "obs" is an abbreviation for scenarios with obstacles.

C. Real-Robot Experiments

We employed the portable multi-robot platform, microMVP [35], for conducting real-world experiments involving car-like robots to validate the execution of our algorithmic solutions on real hardware. Whereas the microMVP robots are differential drive robots, a software layer can be imposed to simulate car-like robots, which is what we did [1]. Fig. 8 gives a visual representation of the setup. As evident from the attached video, paths generated by our planners can be successfully executed on the robots' controllers.

Fig. 8. Snapshot of a real robot experiment with 4 obstacles and 5 robots.

VI. Conclusion and Discussions

In this study, we investigate path planning for multiple car-like robots. We present two distinct algorithms tailored to the specific demands of car-like robot navigation scenarios. The first and our main contribution, PBCR, employs an effective count-based exploration heuristic. Focusing on decentralized decision-making, PBCR demonstrates a notable advancement over previous decentralized approaches. The improvement is evident through significantly heightened success rates with some manageable optimality trade-offs of yielding longer trajectories. Opportunities for improving trajectory quality within PBCR remain; in particular, our approach employs a manually designed Q-function for

[1]While achieving the same goal, the simulation makes the experiment more challenging than running directly on car-like robots.

action selection, which may be replaced with data-driven approaches for reaching optimal performance.

REFERENCES

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, pp. 9–9, 2008.

[2] R. Mason, "Developing a profitable online grocery logistics business: Exploring innovations in ordering, fulfilment, and distribution at ocado," in *Contemporary Operations and Logistics*. Springer, 2019, pp. 365–383.

[3] Q. Wan, C. Gu, S. Sun, M. Chen, H. Huang, and X. Jia, "Lifelong multi-agent path finding in a dynamic environment," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2018, pp. 875–882.

[4] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," in *International Joint Conference on Artificial Intelligence*, 2019.

[5] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[6] L. Wen, Y. Liu, and H. Li, "Cl-mapf: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints," *Robotics and Autonomous Systems*, vol. 150, p. 103997, 2022.

[7] 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.

[8] 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.

[9] P. Surynek, "An optimization variant of multi-robot path planning is intractable," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010.

[10] 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.

[11] ——, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[12] E. Erdem, D. G. Kisa, U. Öztok, and P. Schueller, "A general formal framework for pathfinding problems with multiple agents." in *AAAI*, 2013.

[13] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[14] D. Silver, "Cooperative pathfinding." *AIIDE*, vol. 1, pp. 117–122, 2005.

[15] K. Okumura, "Lacam: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, 2023, pp. 11 655–11 662.

[16] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[17] T. Guo and J. Yu, "Sub-1.5 Time-Optimal Multi-Robot Path Planning on Grids in Polynomial Time," in *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022.

[18] S. D. Han and J. Yu, "Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics," *IEEE Robotics and Automation Letters*, vol. 5, pp. 1350–1357, 2019.

[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] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 785–11 792.

[21] T. Guo and J. Yu, "Toward efficient physical and algorithmic design of automated garages," *arXiv preprint arXiv:2302.01305*, 2023.

[22] A. Okoso, K. Otaki, S. Koide, and T. Nishi, "High density automated valet parking via multi-agent path finding," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 2146–2153.

[23] J. Li, M. Ran, and L. Xie, "Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 405–412, 2020.

[24] D. Le and E. Plaku, "Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1868–1875, 2019.

[25] Y. Ouyang, B. Li, Y. Zhang, T. Acarman, Y. Guo, and T. Zhang, "Fast and optimal trajectory planning for multiple vehicles in a nonconvex and cluttered environment: Benchmarks, methodology, and experiments," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 746–10 752.

[26] 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.

[27] 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.

[28] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proceedings IEEE International Conference on Robotics & Automation*, 2008, pp. 1928–1935.

[29] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 71, no. 2, pp. 399–409, 1977.

[30] 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.

[31] J. P. Van Den Berg and 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.

[32] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[33] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 272–11 281.

[34] S. D. Han and J. Yu, "Optimizing space utilization for more effective multi-robot path planning," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 709–10 715.

[35] J. Yu, S. D. Han, W. N. Tang, and D. Rus, "A portable, 3d-printing enabled multi-vehicle platform for robotics research and education," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1475–1480.