A Dynamic Programming Algorithm for Grid-based Formation Planning of Multiple Vehicles

Tsz-Chiu Au[1]

***Abstract*— A common operation in multirobot systems is to generate a motion plan for multiple robots such that the robots can move in formation to achieve some desired effects. For example, in autonomous parking lots, a group of vehicles can be asked to move to another location when they block another vehicle that needs to leave the parking lot. In this paper, we present a novel grid-based planning approach for motion planning that minimizes the makespan of moving multiple vehicles from one location to another in a safe manner. Unlike most existing multirobot planning algorithms, our algorithm uses dynamic programming to compute a nearly-optimal motion plan for a large group of vehicles in polynomial time with the help of a given set of intermediate vehicle patterns. Our experimental results show that our algorithm is much faster than an exact algorithm but does not increase the minimum makespans tremendously.**

I. INTRODUCTION

In many multirobot systems, robots have to move in formation as quickly as possible while avoiding collisions. A typical example is drone light shows, in which a group of drones displays a sequence of light patterns in the sky by changing from one drone pattern to another repeatedly. There has been work on controlling a team of robots to form and maintain a robot pattern while the robots move together [1], [2]. Some works have considered the multi-UAV formation reconfiguration problem in which the control rules can transform the initial formation configuration into a final configuration [3]. However, long-term planning is needed for more complicated formation reconfiguration. In general, motion planning is NP-hard [4], [5]. There is no efficient algorithm to solve the multirobot planning problem unless we make some assumptions to simplify the problem. For example, Yu and LaValle presented an optimal formation control on graphs in which agents take exactly one time-step to move from one vertex to an adjacent vertex [6], [7]. However, the general case remains difficult.

In this paper, we present a new approach that relies on a given set of intermediate robot patterns to speed up the planning process. This approach is suitable for situations in which the intermediate robot patterns are easily obtainable. The idea of this approach stems from our study of autonomous parking lots (APLs), a type of high-density parking (HDP) [8], [9], [10], [11]. Unlike conventional parking methods that reserve more than half of the space for driveways and sidewalks, HDP reduces the size of driveways by putting vehicles close to each other. An autonomous vehicle in the parking lot can be asked to move autonomously if it blocks another vehicle that needs to leave the parking lot. One basic operation in APLs is to relocate a *block* of vehicles (See Fig. 1). The efficiency of APLs can be improved by minimizing the makespan (i.e., the length) of the motion plan so that more vehicles can get in or out of the parking lot.

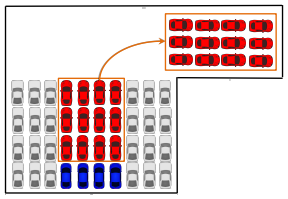

Fig. 1: Move a block of vehicles to the right in an autonomous parking lot.

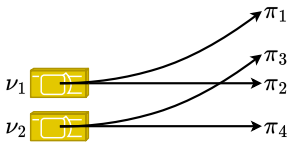

Fig. 2: Local motion plans.

The task of minimizing the makespan is nontrivial since each vehicle can have more than one local plan generated by the local planner, and our algorithm has to choose the best set of local plans such that the makespan of the entire multi-vehicle motion plan can be minimized. For example, in Fig 2, each vehicle has two feasible local plans. If the vehicle ν_1 chooses π_2 and ν_2 chooses π_3, ν_2 has to delay its execution of π_3 since π_2 and π_3 cannot be overlapped during execution. However, if ν_1 chooses π_1 instead, both π_1 and π_3 can be executed simultaneously, and the makespan can be reduced. Fig. 2 highlights the fact that ad hoc execution of local plans (i.e., a vehicle chooses a local plan right before the execution) can fail to minimize the makespan. A planning algorithm is needed to avoid overlapping local motion plans such that the makespan can be shortened.

Another way to reduce the makespan is to choose local plans in which vehicles can move faster. However, vehicles in APLs are close to each other, and hence it is dangerous to drive at high speed. We define a *safety buffer* around a vehicle such that no other vehicle should present at any time (see Fig. 3). The safety buffer is *dynamic* since the shape of a safety buffer depends on the current speed and the current heading of the vehicle. Typically, the shape depends on the amount of space a vehicle needs for an emergency stop in the current driving direction. We utilize the reservation system in autonomous intersection management (AIM) for handling safety buffers [12]. Our *grid-based* motion planning algorithm integrates the reservation system into the planning algorithm such that the safety constraint can be enforced.

In summary, the contributions of this paper are:

- We define the formation planning problem for multiple vehicles given a sequence of intermediate patterns.

[1]Department of Computer Science and Engineering, Ulsan National Institute of Science and Technology, South Korea. chiu@unist.ac.kr

- We define a grid-based reservation system for handling vehicles' dynamic safety buffers for emergency stops.
- We present a polynomial time algorithm for solving the formation planning problem with respect to vehicles' safety buffers.
- We conducted experiments in simulation to compare our algorithm with an optimal graph-search algorithm.

This paper is organized as follows. After the related work section in Sec. II, we define our vehicle formation planning problem in Sec. III, and then present our algorithm in Sec. IV. Finally, we present our experimental results in Sec. V and conclude this paper in Sec. VI.

II. RELATED WORK

Many existing works on multirobot planning specifically focus on controlling a team of robots to maintain a pattern (e.g., [1], [2]). Wei et al. presented a set of integer programming and dynamic programming models for scheduling longitudinal trajectories of a chain of vehicles based on space-time lattices [13]. Our work can be considered as a generalization of longitudinal car-following models to 2D and higher dimensions. Leader-follower models are popular models for formation control (e.g., [14], [15]). But these works typically aim for maintaining a formation by control rules and do not optimize for the makespan. The multi-UAV formation reconfiguration is the task of transforming an initial formation configuration into a final configuration [3]. Our work belongs to this line of research, but we rely on the assumption that the sequence of intermediate formations is given, which is not found in the literature.

A feature of our approach is the discretization of space and time. This feature is similar to occupancy grids [16], but our work is closely related to the grid-based reservation system in AIM [12], [17]. Wu et al. proposed a data structure called stacked reservation grid (SRG) for motion planning [18]. Lattice-based motion planners go one step further to discretize the state of robots [19], [20].

Our bilevel optimization is similar to hierarchical motion planning. For example, Vukosavljev et al. introduced hierarchical motion primitives to solve the motion planning problem for a large collection of agents in a modular framework for motion planning [21], [22]. Grymin et al. presented a hierarchical approach for primitive-based motion planning and control of autonomous vehicles using a library of pre-specified motion primitives [23]. Currently, our system has a two-level hierarchy only, but it is possible to extend the hierarchy for more elaborated scenarios (e.g., coalition planning for several different groups of vehicles).

III. VEHICLE FORMATION PLANNING PROBLEM

We shall consider motion planning for a finite set $\mathcal{V} = \{\nu_1, \nu_2, \ldots, \nu_n\}$ of vehicles in a finite 2D workspace. We discretize the workspace into a $N_x \times N_y$ grid, where $\{C_1, C_2, \ldots, C_{N_x \cdot N_y}\}$ is the set of all cells in the grid. We discretize the timeline into a sequence of time intervals: T_0, T_1, …, T_k where T_k is the horizon, $T_i = [t_i, t_{i+1})$ and $t_i = i \times D$ for a given constant D and $0 \leq i < k$.

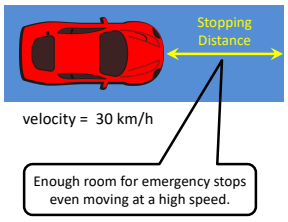

Fig. 3: Dynamic safety buffer.

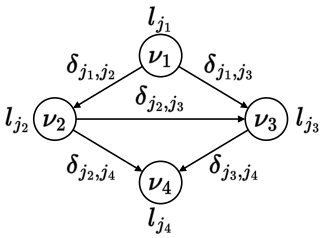

Fig. 4: A formation planning graph.

For simplicity, we shall assume $D = 1$, such that $t_i = i$. Following the notations in AIM [12], [24], the space-time is discretized into a set of *tiles*, each of which is a pair (C_i, T_j). A tile is *occupied* by a vehicle if the tile intersects with the vehicles' dynamic safety buffer, as defined below, during the execution of a motion plan for the vehicle. In our algorithm, collision avoidance is achieved by preventing two vehicles from occupying the same tile.

Let $\rho = (x, y, \theta)$ be a *pose* of a vehicle ν, where (x, y) is the coordinate of the center of ν in the workspace and θ is the heading of ν. Following the simple car model in [25], let (x, y, θ, v, ϕ) be a configuration of a vehicle ν, where (x, y, θ) is a pose of ν, v is the velocity of ν, and ϕ is the steering angle.

Each vehicle can have a different size and shape. Moreover, each vehicle ν has a *static safety buffer* $\partial_0(\nu)$, which is a region slightly larger than the shape of the vehicle. $\partial_0(\nu)$ is the safety buffer when a vehicle is stopped. A *dynamic* safety buffer $\partial(\nu, s)$ is the one that depends on the configuration $s = (x, y, \theta, v, \phi)$ of ν. Typically, $\partial(\nu, s)$ is larger than $\partial_0(\nu)$ when $v > 0$ since a moving car needs more space for safety. Thus, a dynamic safety buffer is an extension of the static safety buffer such that the vehicle can have enough space to make an emergency stop (e.g., the blue region in Fig. 3). A dynamic safety buffer can be obtained by increasing the length of the static safety buffer in the heading direction and fanning out slightly in the direction of the angular velocity so that the vehicle can stop inside the dynamic safety buffer when it is asked to stop immediately. In this paper, all safety buffers are dynamic unless we state otherwise.

A *local motion plan* π for a vehicle ν is a sequence of control commands for controlling ν to move along a trajectory. Given two poses ρ_1 and ρ_2 for ν, there is a *local planner* for ν that can generate a finite set $\mathsf{LocalPlan}_\nu(\rho_1, \rho_2)$ of *distinct* local motion plans quickly. That is, $\mathsf{LocalPlan}_\nu(\rho_1, \rho_2) = \{\pi_1, \pi_2, \ldots, \pi_k\}$ such that ν can change its pose from ρ_1 to ρ_2 using any one of these local motion plans, assuming the velocity of ν at ρ_1 and ρ_2 are zero. The *footprint* of a local motion plan π is the set $\mathsf{Tile}(\pi)$ of tiles that intersects with the safety buffer of ν when ν traverses on the trajectory during the execution of π. For collision avoidance, the footprints of two local motion plans for two different vehicles cannot overlap, even during an emergency stop.

Given a set $\mathcal{V}$ of n vehicles, a *formation* F for $\mathcal{V}$ is a set $\{\rho_1, \rho_2, \ldots, \rho_n\}$ of poses, where ρ_i is a pose for $\nu_i \in \mathcal{V}$. Given two formations $F_1 = \{\rho_i^1\}_{1 \leq i \leq n}$ and $F_2 = \{\rho_i^2\}_{1 \leq i \leq n}$ for $\mathcal{V} = \{\nu_i\}_{1 \leq i \leq n}$ where ρ_i^1 and ρ_i^2 are the poses for ν_i, a

formation plan for F_1 and F_2 is $\Pi = \{\pi_i\}_{1\leq i\leq n}$ where π_i is a local motion plan such that $\pi_i \in \mathsf{LocalPlan}_{\nu_i}(\rho_i^1, \rho_i^2)$, for all $\nu_i \in \mathcal{V}$. However, we cannot directly execute a formation plan to transform one formation into another since the footprints of some of the local motion plans in Π may cross each other (see Fig. 2). Hence, some of the motion plans have to be delayed so that the footprints of the two vehicles do not overlap. A *schedule* for $\Pi = \{\pi_i\}_{1\leq i\leq n}$ is $\Gamma = \{t_i\}_{1\leq i\leq n}$, where t_i is the time delay of the execution of π_i for $1 \leq i \leq n$. Before t_i, ν_i remains at its pose ρ_i. Let $\mathsf{Tile}(\pi_i, t_i)$ be the footprint π_i after delaying its execution for time t_i relative to the start time t_0 of the execution of Π (i.e., π_i will be executed at time $t_0 + t_i$). A schedule Γ for Π is *valid* if and only if $\mathsf{Tile}(\pi_{i_1}, t_{i_1}) \cap \mathsf{Tile}(\pi_{i_2}, t_{i_2}) = \emptyset$ for $1 \leq i_1 < i_2 \leq n$. We call (Π, Γ) a *timed formation plan* for $\mathcal{V}$, where Γ is a valid schedule for Π. The *makespan* of (Π, Γ) is $\mathsf{makespan}(\Pi, \Gamma) = \max_{1\leq i\leq n}\{t_i + |\pi_i|\}$, which is the time difference between t_0 and the end time of the last action execution, where $|\pi_i|$ denotes the length of the execution of the plan π_i.

Given a sequence $\langle F_1, F_2, \ldots, F_m\rangle$ of formations for a set $\mathcal{V}$ of n vehicles, a *meta-formation plan* is $\Pi^{\mathsf{meta}} = \{(\Pi_j, \Gamma_j)\}_{1\leq j\leq m-1}$, where (Π_j, Γ_j) is a timed formation plan that transforms F_j to F_{j+1} for $1 \leq j \leq m-1$. Π^{meta} also needs a *meta-schedule* $\Gamma^{\mathsf{meta}} = \langle t_j^{\mathsf{meta}}\rangle_{1\leq j\leq m-1}$ to determine when a timed formation plan in Π^{meta} should start. Γ^{meta} is *valid* if and only if the footprints of all local motion plans in the timed formation plan (Π_j, Γ_j) do not overlap with the footprints of all local motion plans in $(\Pi_{j+1}, \Gamma_{j+1})$ for $1 \leq j \leq m-2$. More precisely, Γ^{meta} is valid if and only if $\mathsf{Tile}(\pi_{i_1}, t_{i_1}) \cap \mathsf{Tile}(\pi_{i_2}, t_{i_2}) = \emptyset$ for every $\pi_{i_1} \in \Pi_j$ and every $\pi_{i_2} \in \Pi_{j+1}$, where $1 \leq j \leq m-2$. We call $(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$ a *timed meta-formation plan* for $\mathcal{V}$ given $\langle F_1, F_2, \ldots, F_m\rangle$. The *makespan* of $(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$ is $\mathsf{makespan}(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}}) = t_{m-1}^{\mathsf{meta}} + \mathsf{makespan}(\Pi_{m-1}, \Gamma_{m-1})$.

In summary, the problem statement of our vehicle formation planning problem is defined as follows:

Definition 1: Given

1) a set $\mathcal{V} = \{\nu_1, \nu_2, \ldots \nu_n\}$ of vehicles,
2) a sequence $\langle F_1, F_2, \ldots, F_m\rangle$ of formations where $F_j = \{\rho_i^j\}_{1\leq i\leq n}$ in which ρ_i^j is the pose of ν_i in F_j, for $1 \leq j \leq m$, and
3) a set $\mathsf{LocalPlan}_{\nu_i}(\rho_i^j, \rho_i^{j+1})$ of local motion plans for all $\nu_i \in \mathcal{V}$ and $1 \leq j \leq m-1$,

find a timed meta-formation plan $(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$ such that $\mathsf{makespan}(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$ is minimized, where

1) $\Gamma^{\mathsf{meta}} = \langle t_j^{\mathsf{meta}}\rangle_{1\leq j\leq m-1}$ is a *valid* schedule of the meta-formation plan Π^{meta}, and
2) $\Pi^{\mathsf{meta}} = \{(\Pi_j, \Gamma_j)\}_{1\leq j\leq m-1}$, where $\Gamma_j = \{t_i^j\}_{1\leq i\leq n}$ is a *valid* schedule of a formation plan $\Pi_j = \{\pi_i^j\}_{1\leq i\leq n}$, for *some* local motion plan $\pi_i^j \in \mathsf{LocalPlan}_{\nu_i}(\rho_i^j, \rho_i^{j+1})$

IV. FORMATION PLANNING ALGORITHMS

According to Definition 1, the solution to the formation planning problem hinges on choosing 1) a local motion plan π_i^j from the set of all possible local motion plans generated by the local planner for ν_i, for each pose in every formation transformation; 2) a valid schedule Γ_j for every formation plan Π_j constructed by the chosen local motion plans; and 3) a valid meta-schedule Γ^{meta}. In this section, we present a bilevel optimization algorithm in which 1) the lower level minimizes the makespan of the timed formation plan for each transformation of formations by choosing the local motion plans for every vehicle, and 2) the upper level minimizes the makespan of the timed meta-formation plan. It turns out that both the lower level optimization and the upper local optimization can be solved by the same dynamic programming algorithm that returns a nearly-optimal solution in polynomial time.

A. Formation Planning Graphs

The lower level optimization aims to address this problem: given two formations F_1 and F_2 for $\mathcal{V}$ and $\Pi_i = \mathsf{LocalPlan}_{\nu_i}(\rho_i^1, \rho_i^2)$ where $\rho_i^1 \in F_1$ and $\rho_i^2 \in F_2$ for all $\nu_i \in \mathcal{V}$, find 1) a set $\{\pi_i\}_{1\leq i\leq n}$ of local motion plans where $\pi_i \in \Pi_i$ for all $\nu_i \in \mathcal{V}$, and 2) a valid schedule $\Gamma = \{t_i\}_{1\leq i\leq n}$ such that $\mathsf{makespan}(\Pi, \Gamma)$ is minimized. First, we formulate this problem as a graph search problem as follows. We construct an undirected graph $(\mathcal{V}, E')$ by having a vertex ν for each vehicle $\nu \in \mathcal{V}$ and inserting an undirected edge into E' for each pair (ν_1, ν_2) of vehicles if the footprints of ν_1 and ν_2 can *potentially* overlap if they are executed simultaneously (i.e., there exist $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$ such that $\mathsf{Tile}(\pi_1, 0) \cap \mathsf{Tile}(\pi_2, 0) \neq \emptyset$). This condition can be checked by enumerating all pairs of plans in $\Pi_1 \times \Pi_2$. Then we convert the undirected graph $(\mathcal{V}, E')$ into a directed acyclic graph $(\mathcal{V}, E)$ as follows: for each vertex $\nu \in \mathcal{V}$ whose in-degree is zero, we conduct a depth-first search in $(\mathcal{V}, E')$ using ν as the root. When the depth-first search visits a child node ν_2 of ν_1 for the first time, it should check whether $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$ such that π_2 can be possibly executed after π_1. This check can be done by checking whether there exists a start time $t \in [0, |\pi_1|)$ of π_2 such that the footprints of π_1 and π_2 does not overlap when π_1 starts at time 0 and π_2 starts at time t. If this check fails, the depth-first search backtracks at ν_1. If the depth-first search reaches all vertices in $\mathcal{V}$ and there is no cycle (i.e., no vertex is visited twice), we insert a directed edge (ν, ν') into E' whenever ν is a parent of ν' during the search. If the search fails to reach all vertices or a cycle is found, we repeat the depth-first search for another vertex until we construct $(\mathcal{V}, E)$ that is a connected, directed acyclic graph (DAG). If no connected DAGs can be constructed, our algorithm fails since our algorithm can only work with connected DAGs. The reason is that the direction of a directed edge (ν, ν') denotes the requirement that the chosen local motion plan for ν has to be executed before the chosen local motion plan for ν'. This ordering is inconsistent if the graph is cyclic.

Each vertex ν_i in $\mathcal{V}$ is associated with a set $L_i = \{l_j\}_{1\leq j\leq|\Pi_i|}$ of non-negative numbers, where $l_j = |\pi_j|$ is the length of the local motion plan π_j for every $\pi_j \in \Pi_i$. Likewise, each edge (ν_{i_1}, ν_{i_2}) in E is associated with a table Δ_{i_1,i_2} of non-negative numbers, where $\delta_{j_1,j_2} \in \Delta_{i_1,i_2}$ is the

minimum time delay of the local motion plan $\pi_{j_2} \in \Pi_{i_2}$ after the execution of the local motion plan $\pi_{j_1} \in \Pi_{i_1}$ such that the footprints of π_{j_1} and π_{j_2} do not overlap (i.e., $\mathsf{Tile}(\pi_{j_1}, 0) \cap \mathsf{Tile}(\pi_{j_2}, \delta_{j_1,j_2}) = \emptyset$ and $\mathsf{Tile}(\pi_{j_1}, 0) \cap \mathsf{Tile}(\pi_{j_2}, t) \neq \emptyset$ for $0 \leq t < \delta_{j_1,j_2}$). We can compute δ_{j_1,j_2} by increasing δ_{j_1,j_2} from 0 to $|\pi_{j_1}|$ until the footprints of π_{j_1} and π_{j_2} does not overlap. Note that the size of Δ_{i_1,i_2} is $|\Pi_{i_1}| \times |\Pi_{i_2}|$.

Let $(\mathcal{V}, E, \mathbb{L}, \mathbb{\Delta}, \nu_{\mathsf{root}})$ be a *formation planning graph*, where $\mathbb{L} = \{L_i\}_{1 \leq i \leq n}$, $\mathbb{\Delta} = \{\Delta_{i_1,i_2}\}_{(\nu_{i_1},\nu_{i_2}) \in E}$, and $\nu_{\mathsf{root}} \in \mathcal{V}$ is the root of the DAG $(\mathcal{V}, E)$. A *solution* to a formation planning graph is a set $\Pi^{\mathsf{sol}} = \{\pi_{j_i}\}_{1 \leq i \leq n \text{ and } \pi_{j_i} \in \Pi_i}$ of local motion plans. Note that there is one local motion plan π_{j_i} from each Π_i in Π^{sol}. In the rest of this paper, we assume a formation planning graph is given and focus on minimizing the makespan of the given graph.

B. Solution's Makespan

Given a solution Π^{sol}, the formation planning graph can be reduced to $(\mathcal{V}, E, \mathbb{L}', \mathbb{\Delta}', \nu_{\mathsf{root}})$, where $\mathbb{L}' = \{l_{j_i}\}_{1 \leq i \leq n \text{ and } l_{j_i} \in L_i \text{ and } \pi_{j_i} \in \Pi^{\mathsf{sol}}}$, and $\mathbb{\Delta}' = \{\delta_{j_{i_1},j_{i_2}}\}_{(\nu_{i_1},\nu_{i_2}) \in E \text{ and } \delta_{j_{i_1},j_{i_2}} \in \Delta_{i_1,i_2} \text{ and } \pi_{j_{i_1}},\pi_{j_{i_2}} \in \Pi^{\mathsf{sol}}}$. For simplicity, let $\lambda_i = l_{j_i}$ and $\vartheta_{i_1,i_2} = \delta_{j_{i_1},j_{i_2}}$. Thus, in the reduced formation planning graph, there is one number λ_i for each vertex ν_i, and there is one number ϑ_{i_1,i_2} for each edge (ν_{i_1}, ν_{i_2}).

Let $\tau = \langle \nu_{i_1}, \nu_{i_2}, \ldots, \nu_{i_h} \rangle$ be a path that connects the root $\nu_{\mathsf{root}} = \nu_{i_1}$ to ν_{i_h} in a reduced formation planning graph. The lower bound of the *minimum delay* of τ is

$$D(\tau) = \sum_{1 \leq k \leq h-1} \vartheta_{i_k,i_{k+1}}.$$

If τ is a *critical path* (i.e., all chosen local motion plans on τ start at the lowerest possible delay), $D(\tau)$ is the actual minimum delay of the execution of the chosen local motion plan $\pi_{j_{i_h}} \in \Pi^{\mathsf{sol}}$ for ν_{i_h}. In other words, the time delay of $\pi_{j_{i_h}}$ in a valid schedule Γ^* is at least $D(\tau)$. Otherwise, it is just a lower bound of the minimum delay.[2]

A schedule $\Gamma^* = \{t_i^*\}_{1 \leq i \leq n}$ is *optimal* if and only if it is valid and there does not exist another valid schedule $\Gamma = \{t_i\}_{1 \leq i \leq n}$ such that $t_i < t_i^*$ for some $\nu_i \in \mathcal{V}$. The following theorem states that $t_i^* = \max_{\tau \in \Upsilon_i} D(\tau)$.

Theorem 1: Given a solution Π^{sol} to a formation planning graph $(\mathcal{V}, E, \mathbb{L}, \mathbb{\Delta}, \nu_{\mathsf{root}})$, the optimal valid schedule is

$$\Gamma^* = \{t_i^*\}_{1 \leq i \leq n} \tag{1}$$

where $t_i^* = \max_{\tau \in \Upsilon_i} D(\tau)$ and Υ_i is the set of all paths from ν_{root} to ν_i.

Sketch of Proof. Without loss of generality, the indices of the vertices are ordered by a topological sort of the DAG $(\mathcal{V}, E)$, and let $\nu_1 = \nu_{\mathsf{root}}$. Since the length of the path to ν_1 is zero, $t_1^* = 0$ which is optimal for $\pi_1 \in \Pi^{\mathsf{sol}}$. Suppose t_i^* is optimal for $1 \leq i \leq k$. Let $t_{k+1}^* = \max_{\tau \in \Upsilon_{k+1}} D(\tau) = \max_{i' \in \mathsf{parent}(k+1)} \{\vartheta_{i',i} + t_{i'}^*\}$, where $\mathsf{parent}(k+1)$ is the set of indexes of the parents of ν_{k+1} in the DAG. Since all indexes in $\mathsf{parent}(k+1)$ are less than or equal to k, $t_{i'}^*$ are optimal by the induction hypothesis. Therefore, t_{k+1}^* is also optimal. By induction, Γ^* is optimal. □

The above proof shows that t_i^* can be computed recursively by this equation:

$$t_i^* = \max_{i' \in \mathsf{parent}(i)} \{\vartheta_{i',i} + t_{i'}^*\}, \tag{2}$$

for $1 < i \leq n$ and $t_1^* = 0$. Then we can use this equation to compute $\mathsf{makespan}(\Pi^{\mathsf{sol}}, \Gamma^*) = \max_{1 \leq i \leq n} \{\lambda_i + t_i^*\}$ recursively.

C. Optimal Solution

An *optimal* solution $\bar{\Pi}^{\mathsf{sol}}$ of a formation planning graph $(\mathcal{V}, E, \mathbb{L}, \mathbb{\Delta}, \nu_{\mathsf{root}})$ is one that minimizes the makespan: for all Π^{sol}, $\mathsf{makespan}(\Pi^{\mathsf{sol}}, \Gamma^*) \geq \mathsf{makespan}(\bar{\Pi}^{\mathsf{sol}}, \bar{\Gamma}^*)$, where Γ^* and $\bar{\Gamma}^*$ are the optimal schedules for Π^{sol} and $\bar{\Pi}^{\mathsf{sol}}$, respectively. More precisely, the minimum makespan can be computed by the following equation:

$$\begin{aligned} &\min_{\Pi^{\mathsf{sol}}} \left\{ \mathsf{makespan}(\Pi^{\mathsf{sol}}, \Gamma^*) \right\} \\ &\quad = \min_{j_1 \in \Pi_1, \ldots, j_n \in \Pi_n} \left\{ \max_{1 \leq i \leq n} \{l_{j_i} + t_i^*\} \right\}, \end{aligned} \tag{3}$$

where $j_i \in \Pi_i$ is a shorthand for $\pi_{j_i} \in \Pi_i$, for $1 \leq i \leq n$. However, this equation requires an enumeration of all possible solutions and then evaluates the makespan of each solution via Eq. 2. This brute-force approach is quite slow. Therefore, we want to factorize the equation to simplify the calculation. First of all, we expand the max term using Eq. 1:

$$\begin{aligned} &\min_{j_1 \in \Pi_1, \ldots, j_n \in \Pi_n} \left\{ \max_{1 \leq i \leq n} \{l_{j_i} + t_i^*\} \right\} \\ &\quad = \min_{j_1 \in \Pi_1, \ldots, j_n \in \Pi_n} \left\{ \max_{1 \leq i \leq n} \left\{ l_{j_i} + \max_{\tau \in \Upsilon_i} D(\tau) \right\} \right\} \\ &\quad = \min_{j_1 \in \Pi_1, \ldots, j_n \in \Pi_n} \left\{ \max_{1 \leq i \leq n} \max_{\tau \in \Upsilon_i} \{l_{j_i} + D(\tau)\} \right\} \end{aligned} \tag{4}$$

where Υ_i be the set of all paths from ν_{root} to ν_i. The max terms refer to an enumeration of all possible paths starting from ν_{root} in the graph. This enumeration provides some opportunities for factorizing the equation. For example, the minimum makespan of the formation planning graph in Fig. 4 is $\min_{j_1 \in \Pi_1, j_2 \in \Pi_2, j_3 \in \Pi_3, j_4 \in \Pi_4} \{\max\{l_{j_1}, (\delta_{j_1,j_2} + l_{j_2}), (\delta_{j_1,j_3} + l_{j_3}), (\delta_{j_1,j_2} + \delta_{j_2,j_3} + l_{j_3}), (\delta_{j_1,j_2} + \delta_{j_2,j_4} + l_{j_4}), (\delta_{j_1,j_2} + \delta_{j_2,j_3} + \delta_{j_3,j_4} + l_{j_4}), (\delta_{j_1,j_3} + \delta_{j_3,j_4} + l_{j_4})\}\}$, where each term refers to one path starting from the root ν_1 in the graph. We factorize the common prefixes of these terms: $\min_{j_1 \in \Pi_1, j_2 \in \Pi_2, j_3 \in \Pi_3, j_4 \in \Pi_4} \{\max\{l_{j_1}, (\delta_{j_1,j_2} + \max\{l_{j_2}, (\delta_{j_2,j_3} + \max\{l_{j_3}, \delta_{j_3,j_4} + l_{j_4}\}, (\delta_{j_2,j_4} + l_{j_4})\}, \delta_{j_1,j_3} + \max\{l_{j_3}, (\delta_{j_3,j_4} + l_{j_4})\}\}\}$. Then we can push some of the min operators inside the max terms: $\min_{j_1 \in \Pi_1} \{\max\{l_{j_1}, \min_{j_3 \in \Pi_3, j_4 \in \Pi_4} \{\max\{\min_{j_2 \in \Pi_2} \{(\delta_{j_1,j_2} + \max\{l_{j_2}, (\delta_{j_2,j_3} + \max\{l_{j_3}, (\delta_{j_3,j_4} + l_{j_4})\}, (\delta_{j_2,j_4} + l_{j_4})\}\}, \delta_{j_1,j_3} + \max\{l_{j_3}, (\delta_{j_3,j_4} + l_{j_4})\}\}\}\}\}$. Compared to Eq. 3, this equation is much faster in calculating the minimum makespan. However, this equation has two

[2] $D(\tau)$ is the lower bound of the minimum delay of $\pi_{j_{i_h}}$ for the given formation planning graph only. When more than one formation planning graph can be constructed for a given set of vehicles, it is possible that $D(\tau)$ for one formation planning graph is smaller than $D(\tau)$ for another formation planning graph.

drawbacks: 1) some of the min operators cannot be pushed deep inside the min terms, and 2) it is hard to write down an equation to describe where we should insert the min operators in the max terms. In fact, when most paths in the graph are intertwined, the min operators cannot be pushed inside the min terms, and therefore the running time of the calculation remains exponential to the number of vertices.

D. Dynamic Programming

It would be ideal if we could push all min operators closer to the terms that use them. For example, $\min_{j_1\in\Pi_1}\{\max\{l_{j_1}, \min_{j_2\in\Pi_2}\{(\delta_{j_1,j_2} + \max\{l_{j_2}, \min_{j_3\in\Pi_3}\{(\delta_{j_2,j_3} + \max\{l_{j_3}, \min_{j_4\in\Pi_4}\{(\delta_{j_3,j_4} + l_{j_4})\}\}\}, \min_{j_4\in\Pi_4}\{(\delta_{j_2,j_4} + l_{j_4})\}\}\}, \min_{j_3\in\Pi_3}\{\delta_{j_1,j_3} + \max\{l_{j_3}, \min_{j_4\in\Pi_4}\{(\delta_{j_3,j_4} + l_{j_4})\}\}\}\}\}$. But this calculation is incorrect since this allows a vehicle to choose different local motion plans for different min operators. For example, the optimal value of j_4 in the term $\min_{j_4\in\Pi_4}\{(\delta_{j_2,j_4} + l_{j_4})\}$ can be different from the the optimal value of j_4 in $\min_{j_4\in\Pi_4}\{(\delta_{j_3,j_4} + l_{j_4})\}$. But ν_4 cannot choose two different local motion plans in Π_4 simultaneously.

This equation, however, is not totally useless. In fact, the result of this calculation is a *lower bound* of the actual minimum makespan since this equation addresses a *relaxed* problem in which the constraint that a vehicle can choose one and only one local motion plan is relaxed. If we randomly select one of the chosen local motion plans during the calculation of this equation, the quality of the solution is suboptimal but not too far from the optimal solution in many formation planning graphs according to our experiments. More importantly, this equation can be computed in polynomial time by dynamic programming, making it a practical solution to our formation planning problem.

In general, the recursive equation for this calculation is:

$$P(i_1, i_2, j_{i_1}) = \min_{j_{i_2}\in\Pi_{i_2}} \left\{ \delta_{j_{i_1},j_{i_2}} + \max\left\{ l_{j_{i_2}}, \max_{i_3\in\mathsf{children}(i_2)} \{P(i_2, i_3, j_{i_2})\} \right\} \right\}, \tag{5}$$

where $i_2 \in \mathsf{children}(i_1)$ (i.e., ν_{i_2} is a child of ν_{i_1}). In this calculation, we have to add a dummy vertex ν_0 and a dummy edge (ν_0, ν_1) before the root vertex $\nu_{\mathsf{root}} = \nu_1$ in the graph. ν_0 has one local motion plan π_{j_0} of zero length, such that $\delta_{j_0,j_1} = 0$ for all $\pi_{j_1} \in \Pi_1$. Then the calculation starts with $P(0, 1, 1)$, and the result is a lower bound of the actual minimum makespan.

Based on Eq. 5, we devised Algorithm 1 to calculate a lower bound of the minimum makespan of a formation planning graph by dynamic programming. The running time of Algorithm 1 is $O(H \times n^2)$ where $H = \max_{1\le i\le n} |\Pi_i|$, assuming the maximum number of local plans for a vehicle and the number of children vertices of a vertex in a graph are constants. To construct a solution Π^{sol}, we can modify the algorithm to keep track of the chosen value $j_{i_2} \in \Pi_{i_2}$ in the min operator in Eq. 5. Let Π'_{i_2} be the set of all $\pi_{j_{i_2}}$ for every $\nu_{i_2} \in \mathcal{V}$ that contributes to the result of Algorithm 1. We can construct a solution Π^{sol} by randomly choosing a local motion plan from Π'_{i_2} for every $\nu_{i_2} \in \mathcal{V}$. Then we can use Eq. 2 to compute the corresponding schedule Γ^*. Note that $\mathsf{makespan}(\Pi^{\mathsf{sol}}, \Gamma^*)$ differs from the lower bound of the minimum makespan returned by Algorithm 1. In fact, $\mathsf{makespan}(\Pi^{\mathsf{sol}}, \Gamma^*)$ can be larger than the actual minimum makespan since the choice of local motion plans from Π'_{i_2} can be suboptimal.

Algorithm 1 Calculate the lower bound of the minimum makespan by dynamic programming.

procedure CalcMinMakespanLB($\mathcal{V}, E, \mathbb{L}, \mathbb{\Delta}, \nu_{\mathsf{root}}$)
  Conduct a topological sort of $(\mathcal{V}, E)$ and set the vertex indices in the ascending order.
  Add ν_0 to V; Add (ν_0, ν_1) to E; Add $L_0 = \{0\}$ to $\mathbb{L}$; Add $\Delta_{0,1} = \{\delta_{0,j_1} = 0\}_{\pi_{j_1}\in\Pi_1}$ to $\mathbb{\Delta}$. Let $\Pi_0 = \{\pi_1\}$
  Create a $n \times n \times H$ table P, where H is the maximum number of local motion plans for a vehicle.
  for $i_1 = n - 1$ **down to** 0 **do**
    for all $\pi_{j_{i_1}} \in \Pi_{i_1}$ **do**
      for all $i_2 \in \mathsf{children}(i_1)$ **do**
        Compute $P(i_1, i_2, j_{i_1})$ by Eq. 5
  return $P(0, 1, 1)$

E. The Upper Level Optimization

The upper level optimization aims to find an optimal timed meta-formation plan $(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$ for a sequence $\langle F_1, F_2, \ldots, F_m\rangle$ of formations such that $\mathsf{makespan}(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$ is minimized. We can model this problem as a formation planning graph $(\mathcal{V}, E, \mathbb{L}, \mathbb{\Delta}, \nu_{\mathsf{root}})$ in which 1) $\mathcal{V} = \{F_1, F_2, \ldots, F_m\}$, 2) $E = \{(F_i, F_{i+1})\}_{1\le i<m}$, 3) $\mathbb{L} = \{L_i\}_{1\le i\le m}$ where $L_i = \{\mathsf{makespan}(\Pi_i^{\mathsf{sol}}, \Gamma_i^*)\}$ where Π_i^{sol} is the solution of Algorithm 1 for the formation planning graph for F_i and F_{i+1}, for $1 \le i < m$, and 4) $\mathbb{\Delta} = \{\Delta_{i,i+1}\}_{1\le i<m}$ where $\Delta_{i,i+1} = \{\delta_{1,1}^{i,i+1}\}$ and $\delta_{1,1}^{i,i+1}$ is the minimum time delay between Π_i^{sol} and Π_{i+1}^{sol}, and 5) $\nu_{\mathsf{root}} = F_1$. Then we can use Algorithm 1 to compute a nearly optimal timed meta-formation plan $(\Pi^{\mathsf{meta}}, \Gamma^{\mathsf{meta}})$. The running time of the algorithm is $O(m^2)$ since there is only one timed formation plan for each pair of adjacent formations. Together with the lower level optimization, our approach's total running time is $O(m \times H \times n^2 + m^2)$.

F. Automatic Generation of Intermediate Formations

Our approach relies on the availability of a sequence of intermediate formations. In some applications, it is easy to identify the intermediate formations that are helpful. For example, in autonomous parking lots, we can precompute a set of locations based on the geometrical shape of a parking lot and force vehicles to go through these locations in formation. However, in drone light shows, the intermediate formations for complicated formation transformation may not be obvious. In the latter situations, we can generate intermediate formations automatically using sampling-based motion planning algorithms such as RRT [26], [27], RRT* [28], and PRM [29]. If we keep the shape of the formation the same in all intermediate formations, we can consider all robots in a formation as one robot and use

Algorithm 1 as the local planner for these sampling-based algorithms. Obviously, the sequence of intermediate formations generated by this method may not be optimal since we can do better by allowing formations to have different shapes. In the future, we will study formation shapeshifting when applying sampling-based motion planning algorithms for the automatic generation of intermediate formations.

V. EXPERIMENTAL EVALUATION

We conducted two experiments to evaluate our approach. In Experiment 1, we compared Algorithm 1 to a graph-search algorithm that returns a motion plan with minimum makespan. In Experiment 2, we tested our algorithm in confined environments such as autonomous parking lots. Our experiments were conducted in a simulator we developed for traffic simulation. Both the simulator and the algorithms were implemented in C++23 with the SDL2 library.

In Experiment 1, we developed a graph-search algorithm that returns an optimal formation plan with minimal makespan. The graph-search algorithm is based on Eq. 4 with some tricks to push some but not all min operators inside the max terms. It is a complete algorithm that guarantees to return an optimal solution that is the same as a brute-force search's solution, but the algorithm is around three times faster than a naive implementation of a brute-force search. Unfortunately, the running time of the graph-search algorithm remains exponential to the number of vehicles. Due to space limitation, we cannot present the details of the graph-search algorithm. For more information, please examine the source code of the graph-search algorithm and the brute-force search we released on GitHub.[3]

We considered two different types of formation: rectangular grids and triangular grids. We generated formation pairs by the following steps. For each formation of size n, we systematically put n vehicles in a grid such that they are close to each other. The vehicles' headings are the same. After generating an initial formation, we duplicated the formation at a random location in the workspace with a different orientation to generate the final formation. Then we check whether there are local motion plans for every vehicle to move from the location in the initial formation to the location in the final formation. If there is a vehicle that has no local motion plan, we reject the pair of initial and final formations and generate another pair. In the end, the number of local motion plans for each vehicle is either 1 or 2. We randomly generated 100 formation pairs generated according to the above procedure.

Second, we converted the formation pairs into formation planning graphs based on a discretization of the workspace and the timeline. The size of a cell in a grid is $1m \times 1m$, and the length of the time interval of a tile is $0.04s$. The size of the static safety buffer of all vehicles is $6m \times 3m$, but the size of the dynamic safety buffer increases with the vehicle's speed. We ran Algorithm 1 and the graph-search algorithm with every formation graph and measured the running times of the algorithms and the makespans of the formation plans. The results are shown in Fig 5 and Fig 6.

[3] https://github.com/chiuau/multiplan

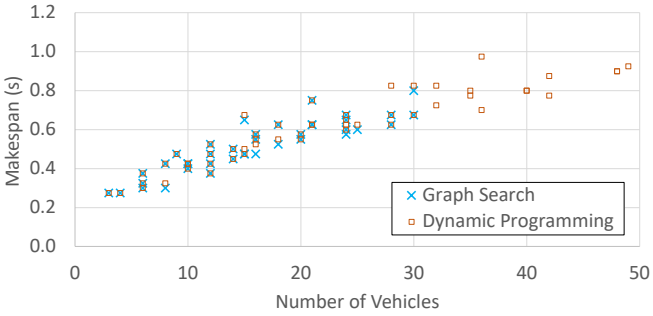

Fig. 5: Average makespan vs. the number of vehicles

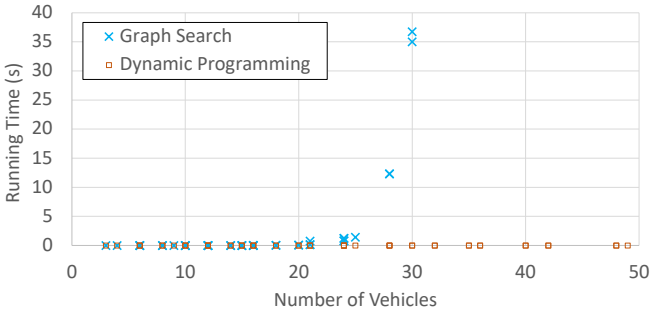

Fig. 6: Running time vs. the number of vehicles

TABLE I: The average makespans in Experiment 2.

	Right-Shift	Left-Turn	Reverse
Avg. makespan (in sec.)	7.13	11.57	35.47

As can be seen in Fig. 5, the makespans of the formation plans generated by both algorithms are mostly the same. It means that even though Algorithm 1 cannot guarantee to find an optimal solution, it often returned an optimal solution when n, the number of vehicles, is small. When n is larger than 30, the graph-search algorithm took too much time to find an optimal solution (see Fig. 6), but Algorithm 1 can still return solutions within a few milliseconds. According to the trend in Fig. 5, we believe that when n is large, the solutions returned by Algorithm 1 are not optimal but remain close to the optimal solution. More importantly, since Algorithm 1 can return a suboptimal solution quickly, Algorithm 1 is more useful than the graph-search algorithm when n is large.

In Experiment 2, we used our algorithm to generate motion plans for autonomous vehicles in a parking lot. We considered three maneuvers: 1) shifting a block of vehicles to the right, 2) moving a block of vehicles forward and then making a left turn, and 3) reserving the orientation of a block of vehicles. In each maneuver, we handpicked different sequences of intermediate formations. Then we measured the average makespan of the timed meta-formation plan, and the result is shown in Table I. This experiment demonstrates the feasibility of our approach for essential vehicle maneuvers in confined areas. We found no difficulty in handpicking the intermediate formations to accomplish the maneuvers.

VI. DISCUSSION AND FUTURE WORK

We have presented a new dynamic programming algorithm for planning a group of vehicles to move from one formation to another with respect to their dynamic safety buffer. This grid-based approach relies on a sequence of intermediate formations that can be either given or generated automatically. Our algorithm can generate a schedule of local motion plans for many vehicles with a suboptimal makespan within a few milliseconds, which is much faster than a complete graph search algorithm. The speed of the algorithm is very important in some applications such as autonomous parking lots. Our experimental results show that the makespan of the motion plans generated by this algorithm is not far off when compared with an exact algorithm. The bilevel optimization procedure stitches the motion plans together to create a motion plan that moves a group of vehicles along the sequence of intermediate formations efficiently.

The advantage of our approach is that our algorithm can run extremely fast and can scale better than existing multirobot motion planning algorithms as the number of vehicles increases. Moreover, our grid-based approach, which converts vehicles' trajectories into sets of tiles for collision detection, can easily handle non-holonomic constraints of vehicles' motions as well as the dynamic safety buffer for emergency stops. Although we focus on motion planning in 2D workspaces, our approach should work in other multirobot systems in 3D environments. Nonetheless, our approach has several limitations. First, vehicles are required to stop completely at their designated locations in intermediate formations, causing non-smooth rides. In the future, we intend to utilize lattice-based motion planning to generate smooth trajectories between formations. Second, our approach only works with directed acyclic formation planning graphs. In some cases, valid motion plans exist even if there are cycles in the graph. Hence, one future work is to make our algorithm work with undirected cyclic graphs.

ACKNOWLEDGMENTS

This work has been taken place at UNIST and was supported by NRF (2022R1A2C101216812).

REFERENCES

[1] M. Turpin, N. Michael, and V. Kumar, "Trajectory design and control for aggressive formation flight with quadrotors," *Autonomous Robots*, vol. 33, pp. 143–156, 2012.

[2] H. Wang and M. Rubenstein, "Shape formation in homogeneous swarms using local task swapping," *IEEE Transactions on Robotics*, vol. 36, no. 3, pp. 597–612, 2020.

[3] H. Duan, Q. Luo, Y. Shi, and G. Ma, "Hybrid particle swarm optimization and genetic algorithm for multi-uav formation reconfiguration," *IEEE Computational Intelligence Magazine*, vol. 8, no. 3, pp. 16–27, 2013.

[4] J. H. Reif, "Complexity of the movers problem and generalizations," in *IEEE Symposium on Foundations of Computer Science*, 1979.

[5] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects: Pspace-hardness of the "warehouseman's problem"," *International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[6] J. Yu and S. M. LaValle, "Distance optimal formation control on graphs with a tight convergence time guarantee," in *IEEE Conference on Decision and Control*, 2012.

[7] ——, "Shortest path set induced vertex ordering and its application to distributed distance optimal formation path planning and control on graphs," in *IEEE Conference on Decision and Control*, 2013.

[8] J. Timpner, S. Friedrichs, J. van Balen, and L. Wolf, "k-stacks: High-density valet parking for automated vehicles," in *IEEE Intelligent Vehicles Symposium*, 2015.

[9] H. Banzhaf, F. Quedenfeld, D. Nienhüser, S. Knoop, and J. M. Zöllner, "High density valet parking using k-deques in driveways," in *IEEE Intelligent Vehicles Symposium*, 2017, pp. 1413–1420.

[10] J. Azevedo, P. M. D'orey, and M. Ferreira, "High-density parking for automated vehicles: A complete evaluation of coordination mechanisms," *IEEE Access*, vol. 8, pp. 43 944–43 955, 2020.

[11] T.-C. Au, "Gridlock-free autonomous parking lots for autonomous vehicles," in *IEEE/RSJ International conference on Intelligent Robots and Systems*, 2021, pp. 4881–4887.

[12] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research (JAIR)*, March 2008.

[13] Y. Wei, C. Avcı, J. Liu, B. Belezamo, N. Aydın, P. Li, and X. Zhou, "Dynamic programming-based multi-vehicle longitudinal trajectory optimization with simplified car following models," *Transportation Research Part B: Methodological*, vol. 106, pp. 102–129, 2017.

[14] H. Rezaee, T. Parisini, and M. M. Polycarpou, "Resiliency in dynamic leader–follower multiagent systems," *Automatica*, vol. 125, no. 4, 2021.

[15] L. Dou, S. Cai, X. Zhang, X. Su, and R. Zhang, "Event-triggered-based adaptive dynamic programming for distributed formation control of multi-uav," *Journal of the Franklin Institute*, vol. 359, no. 8, pp. 3671–3691, 2022.

[16] S. Thrun and A. Bücken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1996, pp. 944–950.

[17] T.-C. Au, C.-L. Fok, S. Vishwanath, C. Julien, and P. Stone, "Evasion planning for autonomous vehicles at intersections," in *IEEE/RSJ International conference on Intelligent Robots and Systems*, 2012, pp. 1541–1546.

[18] F. Wu, D. Wang, M. Hwang, C. Hao, J. Lu, T. Darrell, and A. Bayen, "Motion planning in understructured road environments with stacked reservation grids," in *ICRA Workshop on Perception, Action, Learning (PAL)*, 2020.

[19] M. Cirillo, T. Uras, and S. Koenig, "A lattice-based approach to multi-robot motion planning for non-holonomic vehicles," in *IEEE/RSJ International conference on Intelligent Robots and Systems*, 2014, pp. 232–239.

[20] M. Cirillo, "From videogames to autonomous trucks: A new algorithm for lattice-based motion planning," in *IEEE Intelligent Vehicles Symposium*, 2017, pp. 148–153.

[21] M. Vukosavljev, A. P. Schoellig, and M. E. Broucke, "Hierarchically consistent motion primitives for quadrotor coordination," arXiv, 2019.

[22] M. Vukosavljev, Z. Kroeze, A. P. Schoellig, and M. E. Broucke, "A modular framework for motion planning using safe-by-design motion primitives," *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1233–1252, 2019.

[23] D. J. Grymin, C. B. Neas, and M. Farhood, "A hierarchical approach for primitive-based motion planning and control of autonomous vehicles," *Robotics and Autonomous Systems*, vol. 62, no. 2, pp. 214–228, 2014.

[24] T.-C. Au, N. Shahidi, and P. Stone, "Enforcing liveness in autonomous traffic management," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2011, pp. 1317–1322.

[25] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[26] ——, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept, Iowa State University, Tech. Rep. TR 98-11, 1998.

[27] S. M. LaValle and J. James J. Kuffner, "Rapidly-exploring random trees: progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[28] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[29] L. E. Kavaki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 1996.