Hierarchical Planning for Autonomous Parking in Dynamic Environments

Wang, Yebin; Hansen, Emma; Ahn, Heejin

TR2024-034 April 04, 2024

Abstract

This paper investigates planning for autonomous parking in a dynamic environment where moving obstacles are present. To fulfill fast planning, we employ a divide-and- conquer approach where path planning with static obstacles and safe motion planning with moving obstacles are solved sequentially. We develop a bi-directional improved A-search guided tree algorithm to achieve fast path planning by proposing two modifications to node selection and node expansion of the A* algorithm. First, with the simultaneous construction of two trees rooted at the initial configuration and goal configuration, respectively, the arrival costs of both trees are shared to better estimate the cost-to-go, which improves node selection. Second, by partitioning motion primitives into prioritized modes to facilitate mode selection, node expansion grows the tree toward a more finely tuned direction. For safe motion planning, we define conflict areas as segments of the path that overlap or intersect with moving obstacles' paths and then develop scheduling algorithms to assign time intervals during which the ego vehicle can occupy each conflict area. Particularly, to improve throughput and lower computational complexity, we divide large conflict areas into small areas and establish that, in certain scenarios, the original scheduling problem can be decoupled into sub-problems involving the subsets of conflict areas. Simulation verifies the effectiveness of the proposed architecture and algorithms.

IEEE Transactions on Control Systems Technology 2024

Hierarchical Planning for Autonomous Parking in Dynamic Environments

Yebin Wang, *Senior Member, IEEE*, Emma Hansen, and Heejin Ahn, *Member, IEEE*

***Abstract*—This paper investigates planning for autonomous parking in a dynamic environment where moving obstacles are present. To fulfill fast planning, we employ a divide-and-conquer approach where path planning with static obstacles and safe motion planning with moving obstacles are solved sequentially. We develop a bi-directional improved A-search guided tree algorithm to achieve fast path planning by proposing two modifications to node selection and node expansion of the A* algorithm. First, with the simultaneous construction of two trees rooted at the initial configuration and goal configuration, respectively, the arrival costs of both trees are shared to better estimate the cost-to-go, which improves node selection. Second, by partitioning motion primitives into prioritized modes to facilitate mode selection, node expansion grows the tree toward a more finely tuned direction. For safe motion planning, we define conflict areas as segments of the path that overlap or intersect with moving obstacles' paths and then develop scheduling algorithms to assign time intervals during which the ego vehicle can occupy each conflict area. Particularly, to improve throughput and lower computational complexity, we divide large conflict areas into small areas and establish that, in certain scenarios, the original scheduling problem can be decoupled into sub-problems involving the subsets of conflict areas. Simulation verifies the effectiveness of the proposed architecture and algorithms.**

I. Introduction

PLANNING, originating in robotics [1], has been recently studied in the context of autonomous vehicles [2]. This work strives to achieve fast planning for autonomous parking in a dynamic environment that not only is tight on space but also contains moving obstacles. The problem is challenging in several perspectives. First, planning in a static environment has been recognized as PSPICE-hard [3]. Second, tightness implies that the region containing feasible paths is small relative to the whole space; thus, finding a feasible path could be time-consuming. Third, autonomous driving imposes more stringent real-time requirements than robotic applications.

Well-established results for planning in static environments include optimal control [4]–[6]; mixed-integer linear programming (MILP) [7], [8]; potential fields [9]; sampling-based algorithms such as probabilistic roadmaps (PRM) [10], rapidly-exploring random trees (RRT) [11], RRT* and PRM* [12], particle RRT [13], anytime RRT [14], [15]; and A* [16] and its variants [17]–[20]. Optimal control is well-suited for simple environments but results in hardly tractable optimization problems for general scenarios. The mixed-integer formulation ameliorates the numerical stability of optimization solvers but is accompanied by an increase in computational load. Potential fields are susceptible to being trapped in local minimums and are not complete. Sampling-based methods [10]–[12], [21] construct a sparse representation of the collision-free configuration space and partially address the curse of dimensionality, with guaranteed probabilistic completeness in many circumstances. Deterministic algorithms such as A* and D* achieve resolution-completeness and optimality guarantees under certain circumstances [22]. For sampling-based methods and A*, a key pathway to real-time applications is quickly growing a graph towards a goal configuration [2], [23].

Planning work for dynamic environments has been limited. Having recognized how unlikely it is to find a global trajectory reaching the target within a given computation budget, [33] proposes a partial motion planner to obtain a local trajectory. Similarly, results in [34] exploit receding horizon control to determine a local path and motion plan simultaneously. These methods are quick, but lack completeness. Alternatively, one can focus on completeness, while sacrificing run-time. D* [1], [24] deals with slow-moving obstacles by repairing or replanning the path. RRT-based results [25] handle moving obstacles at the steering level, where any local trajectory undergoes a collision check against moving obstacles. Works [21], [26]–[28] follow a state-time notion, where the time is treated as an additional state, and construct graphs at all time instants. The construction of such a spatial-temporal graph appears to be computationally prohibitive and constitutes the key hurdle to real applications. In [29], the authors introduce safe time intervals associated with configurations occupied by moving obstacles to narrow down the search space and accelerate the construction of the spatial-temporal graph. Following the decomposition idea, works [30]–[32] propose a dynamic roadmap, where a roadmap is constructed offline from static obstacles and is updated online based on moving obstacles.

A variety of planning methods have been tailored to autonomous parking. Early works [35], [36] focus on real-time motion planning for specific parking scenarios in static environments, and it is not obvious how to generalize to dynamic environments. Optimal control-based results [37]–[40], although dealing with moving obstacles and being general, are not suitable for real-time applications because of their reliance on good initial guesses and are computationally taxing. Sampling-based methods and A* have resulted in considerable progress in general parking, e.g. RRT-based [41],

Y. Wang is with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139, USA. (email: yebinwang@ieee.org)

E. Hansen is a PhD student in the Department of Mathematics at the University of British Columbia, Vancouver, BC, Canada. This work was done while she was an intern with MERL. (email: ehansen@math.ubc.ca)

H. Ahn was a visiting research scientist at MERL and is now an Assistant Professor at the School of Electrical Engineering, KAIST. (email: heejin.ahn@kaist.ac.kr)

[42], heuristic-based search [43], and A* search-based [20], [44]. Instrumented with biased sampling strategies or heuristics, small trees can be quickly constructed to meet computational and memory constraints imposed by vehicle platforms. However, most of these results either do not consider moving obstacles or fail to meet real-time specifications. Existing planning solutions for autonomous parking need improvements to meet customer demands in multiple aspects: computational efficiency, applicability to general parking tasks, and capability of dealing with moving obstacles.

Contributions: This work adopts the divide-and-conquer approach in [30], [45] to fulfill real-time planning for autonomous parking in environments containing moving obstacles. We perform planning by solving two sub-problems: path planning with static obstacles and safe motion planning with moving obstacles. We now discuss the noticeable differences between our approach and the method proposed in [45]. Taking the motion planning stage as an example, motion planning in [45] is recast into path planning over a directional visibility graph (DVG). The DVG is constructed based on the monotonicity of time and maximum velocity constraints, and forbidden zones in the time dimension (forbidden zones characterize the intersections between the robot's path and obstacles' trajectories). Our work resorts to a hierarchical framework [46]: an optimization-based scheduler on the top identifies all possible conflict areas (CAs) based on the path of the ego vehicle and trajectories of moving obstacles, and schedules when to enter CAs, and an underlying analytical motion planner generates the motion according to the schedules.

Regarding path planning, a Bi-directional Improved A-search Guided Tree (BIAGT) is developed to fulfill real-time path computation in static environments. It improves A* in two aspects. First, by partitioning motion primitives (MPs) into multiple modes and assigning each mode a priority, a node is expanded according to mode selection. During the expansion, only the MPs of the mode with the highest priority will be applied. Second, we propose an alternative approach to estimate the cost-to-go, which is notoriously difficult to reckon with due to its dependence on the environment and vehicle dynamics [47]. By simultaneously constructing a start tree and a goal tree rooted at the initial configuration and goal configuration, respectively, the cost-to-go of nodes on one tree could be better estimated by incorporating the arrival costs of nodes on the other tree. The rationale behind this mechanism is that the other tree's arrival cost contains obstacle information around the goal of the current tree. BIAGT demonstrates effectiveness in most scenarios, especially if obstacles are near the goal. It enjoys the same completeness property as A* and typically produces a smaller tree. Unfortunately, to bring in computational advantages, the cost-to-go has to be overestimated, which implies a less favorable path than A*. BIAGT could perform worse if the arrival cost of the other tree provides misleading information, and how to detect and avoid this pitfall is left for future work.

Given the path computed by BIAGT, we follow the hierarchical framework developed in [46] to solve safe motion planning in dynamic environments, where CAs are treated as resources that are shared by moving obstacles and the ego vehicle. A scheduler determines the timing for the vehicle to enter CAs by solving an MILP problem, and an underlying algorithm computes the motion according to the scheduling decision. It is noteworthy that work [46] facilitates intersection management, where CAs are typically small. In garage parking, CAs could arise from intersections and overlapping of paths, where the latter could yield large CAs. A plain adoption of results in [46] could lead to conservative solutions, amounting to low operation efficiency. We propose two customizations, where the first improves the garage throughput by splitting a large, connected CA into small concatenated CAs, and the second addresses computational complexity by decoupling the scheduling problem into separate sub-problems. It is worth pointing out that the scheduler and the underlying trajectory generation compute the reference trajectory of the ego vehicle. Thus, the scheduler has full access to the reference trajectories of moving obstacles. This assumption is acceptable for garage systems with centralized management. Meanwhile, assuming that the control systems of both the ego vehicle and the moving obstacles execute their reference trajectories precisely, the scheduler may run only once before the ego vehicle starts moving. To ensure safety during movement, the trajectory tracking control module reacts to tracking errors such that the ego vehicle does not violate the safety constraints imposed during scheduling.

This approach has strong ties to a variety of existing methods. For instance, BIAGT uses similar observations to [2], [47], [48]: it is crucial and effective to encode obstacle information into the estimated cost-to-go. The prioritized node expansion and sharing arrival costs of both trees to improve the cost-to-go estimate have been proposed in [20], where both ideas are implemented in individual algorithms. The idea of splitting obstacles into static and dynamic categories and treating them differently can be found in [30], [45], [49], [50]. Differences between the literature and our approach are also pronounced. For example, dynamic obstacles are handled reactively and locally in [49], [50]. This treatment necessarily incurs performance loss, whereas our work takes all information into account for a global solution before the ego vehicle starts moving.

The remainder of this paper is organized as follows. The planning problem and system architecture are provided in Section II. Section III presents BIAGT and the performance analysis. Section IV elaborates on the hierarchical solution to the safe motion planning problem. Simulation results are included in Section V, followed by conclusions in Section VI.

II. Preliminaries

A. Motivating Application

Imagine how an autonomous vehicle can safely enter or leave a garage as shown in Fig. 1, where parking lots are arranged to admit one-way traffic. A garage management server (GMS) processes service requests from vehicles and verifies motion plans of vehicles. A sensing system provides real-time high-fidelity mapping and localization service. A communication system facilitates information exchange between infrastructure (GMS and sensing) and vehicles.

Take incoming traffic as an example. A vehicle at configuration $\boldsymbol{q}_0$ requests parking. GMS assigns and sends a target lot $\boldsymbol{q}_f$ to the vehicle, along with a static map $\mathcal{W}_s$ of the garage and trajectories of moving obstacles $\mathcal{R}_o$. The vehicle generates and submits the motion plan to GMS for approval. The vehicle, assisted by infrastructure, is responsible for computing a control sequence to reach $\boldsymbol{q}_f$.

Remark 2.1: An autonomous vehicle relies on the mapping and localization service of infrastructure for planning and control because the quality of onboard sensors is presumably inferior to that of the infrastructure. A high-quality mapping and localization service lessens uncertainties that the planner and vehicle controller combat, and improves system safety and operational efficiency. Large uncertainties lead to conservative plans with slow execution. □

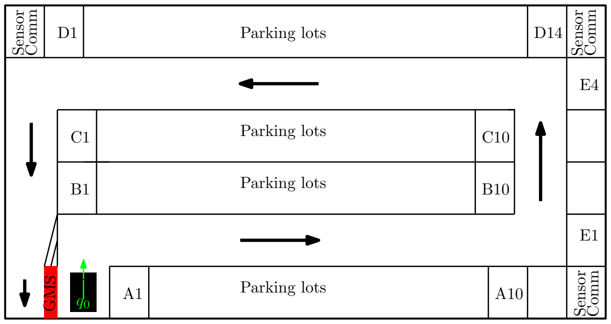


Fig. 1. Garage facility layout.

B. Planning Problems

Consider a robot with the following dynamic model

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}), \tag{1}$$

where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^{n_x}$ is the state, $\boldsymbol{u} \in \mathbb{R}^{n_u}$ the control, and $f(\cdot,\cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ a smooth vector field. A *feasible trajectory* is a collision-free solution $\boldsymbol{x}(t)$ of (1) with initial conditions and $\boldsymbol{u}(t)$, i.e., the robot moving along it does not intersect with obstacles in the environment.

Consider the following fifth-order model of a front-wheel drive vehicle [51]

$$\begin{aligned} \dot{x} &= \cos(\theta) v \\ \dot{y} &= \sin(\theta) v \\ \dot{\theta} &= \tan(\phi)\frac{v}{L} \\ \dot{s} &= |v| \\ \dot{v} &= a, \end{aligned} \tag{2}$$

where (x, y) are the coordinates of the midpoint of the rear wheel axis, θ the vehicle orientation, s the longitudinal distance, v the velocity along the vehicle orientation, ϕ the steering angle, a the acceleration, and L the distance between (x, y) and the midpoint of the front wheel axis. The control input is $\boldsymbol{u} = (a, \phi)^\top$ where a and ϕ are subject to physical constraints $|a| \leq a_{\max}$ and $|\phi| \leq \phi_{\max}$, respectively, i.e., the domain of admissible control is $\mathbb{U} \triangleq [-a_{\max}, a_{\max}] \times [-\phi_{\max}, \phi_{\max}]$. The vehicle has the minimum turning radius $R = L/\tan(\phi_{\max})$.

This work investigates the following planning problem.

Problem 2.2: Consider the vehicle model (2) with $\boldsymbol{x} = (x, y, \theta, s, v)^\top$ and $\boldsymbol{u} = (a, \phi)^\top \in \mathbb{U}$. Given a dynamic map $\mathcal{W}_d$ comprising a static map $\mathcal{W}_s$ and trajectories of moving obstacles $\mathcal{R}_o$, an initial state $\boldsymbol{x}_0$, and a goal state $\boldsymbol{x}_f$, find a feasible trajectory $\mathcal{R}(t) : \mathbb{R} \to \mathcal{X}, \forall t \in [0, t_f]$ such that $\mathcal{R}(0) = \boldsymbol{x}_0, \mathcal{R}(t_f) = \boldsymbol{x}_f$.

Our approach follows the divide-and-conquer approach to solve Problem 2.2, by defining and tackling the following path planning and safe motion planning problems separately.

Path planning: Path planning hinges on concepts such as configuration and configuration space [23]. A *configuration* $\boldsymbol{q} \in \mathbb{R}^{n_c}$ of system (1) is a complete specification of the position of every point in the system. The *configuration space*, denoted by $\mathcal{C} \subset \mathbb{R}^{n_c}$, represents all possible configurations. The state space $\mathcal{X}$ typically has a higher dimension than $\mathcal{C}$. A collision-free configuration space, $\mathcal{C}_{\text{free}}$, is the set of collision-free configurations. Because both vehicles and obstacles can occupy the same space, it is non-trivial to represent $\mathcal{C}_{\text{free}}$ analytically. Since $\boldsymbol{q} = (x, y, \theta)^\top$ uniquely determines the locations of all points on the vehicle, the configuration space is $\mathcal{C} \subset \mathbb{R}^2 \times \mathbb{S}^1$, where $\mathbb{S}^1$ denotes the one-dimensional sphere. The corresponding kinematic model of the vehicle is [52]

$$\begin{aligned} \dot{x} &= \cos(\theta) v \\ \dot{y} &= \sin(\theta) v \\ \dot{\theta} &= \frac{v\phi_n}{R}, \end{aligned} \tag{3}$$

where $\phi_n = \tan(\phi)/\tan(\phi_{\max})$ is the normalized steering angle. Kinematic model (3) has control input $\boldsymbol{u}_p = (v, \phi_n)^\top$ belonging to the domain $\mathbb{U}_p \triangleq [-v_{\max}, v_{\max}] \times [-1, 1]$. A *feasible path* $\mathcal{P}$ is referred to as a collision-free solution $\boldsymbol{q}(t)$ of (3) with $\boldsymbol{q}(0) = \boldsymbol{q}_0$ and an admissible input.

The path planning problem is stated as follows:

Problem 2.3: Consider the vehicle kinematic model (3) with $\boldsymbol{u}_p = (v, \phi_n)^\top \in \mathbb{U}_p$. Given static map $\mathcal{W}_s$, an initial configuration $\boldsymbol{q}_0 \in \mathcal{C}_{\text{free}}$, and a goal configuration $\boldsymbol{q}_f \in \mathcal{C}_{\text{free}}$, find a feasible path $\mathcal{P}$ which starts at $\boldsymbol{q}_0$ and ends at $\boldsymbol{q}_f$.

Remark 2.4: Path $\mathcal{P}$ can be re-parameterized in terms of s, where $s(\boldsymbol{q}(t)) = \int_0^t |v(\tau)| \mathrm{d}\tau = \int_0^t \sqrt{\dot{x}^2(\tau) + \dot{y}^2(\tau)} \mathrm{d}\tau$ is the length of the arc from $\boldsymbol{q}_0$ to $\boldsymbol{q}(t) \in \mathcal{P}$. This is because $s(t) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is monotonic and its inverse $t(s)$ exists. □

Motion planning: Safely maneuvering the ego vehicle along path $\mathcal{P}$ involves future trajectories of moving obstacles and the following longitudinal dynamics of the ego vehicle

$$\begin{aligned} \dot{s} &= v \\ \dot{v} &= a, \end{aligned} \tag{4}$$

which has control input $\mathbf{u} = a$. Denote $\mathbf{x} = (s, v)^\intercal \in \mathbb{X} \triangleq \mathbb{R}_{\geq 0} \times [0, v_{\max}]$ and $\mathbb{U}_m \triangleq [-a_{\max}, a_{\max}]$. The notation v, representing the velocity of the rear wheels in (3), is abused to depict the longitudinal velocity in the model (4).

Let $s_f(\mathcal{P}) \in \mathbb{R}_{\geq 0}$ denote the total arc-length of $\mathcal{P}$, and $s \in [0, s_f(\mathcal{P})]$ the arc-length of a point in $\mathcal{P}$ from the start position $\boldsymbol{q}_0 \in \mathcal{P}$. Assume the vehicle stands still at $\boldsymbol{x}_0, \boldsymbol{x}_f$. The safe motion planning problem is formulated as follows.

Problem 2.5: Consider the vehicle longitudinal dynamics (4) with $a(t) \in \mathbb{U}_m$. Given $\mathcal{P}$ a solution of Problem 2.3, and dynamic map $\mathcal{W}_d$, find a feasible motion $s(t), \forall t \in [0, t_f]$ along $\mathcal{P}$ such that $\mathbf{x}(0) = (0,0)^\top$ and $\mathbf{x}(t_f) = (s_f(\mathcal{P}), 0)^\top$.

Remark 2.6: The solution to Problem 2.2 can be readily constructed from $\mathcal{P}(s)$ and $s(t)$. Specifically, one can re-parameterize $\phi(s)$ as $\phi(t)$, and thus the control $\boldsymbol{u} = (a, \phi)^\top$ of the model (2) is well-defined. □

C. Planning System Architecture

The planning system architecture is illustrated by Fig. 2. Given static map $\mathcal{W}_s$, initial configuration $\boldsymbol{q}_0$, and goal configuration $\boldsymbol{q}_f$, the path planner, based on the BIAGT algorithm detailed in Section III, generates a feasible path $\mathcal{P}$ as a solution of Problem 2.3; then based on $\mathcal{P}$ and dynamic map $\mathcal{W}_d$, the motion planner, based on the safe motion planning algorithm detailed in Section IV, determines the longitudinal velocity profile $s(t)$ as a solution of Problem 2.5. Then Remark 2.6 is applied to construct the solution $\mathcal{R}(t)$ to Problem 2.2. An underlying tracking controller commands the vehicle to track $\mathcal{R}(t)$ faithfully.

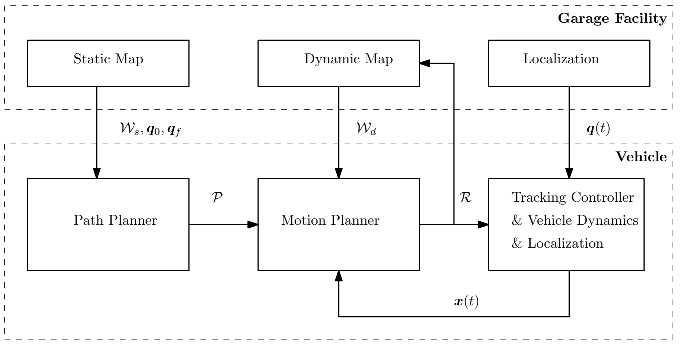


Fig. 2. Planning system architecture. A path planner computes a feasible path $\mathcal{P}$ based on $\mathcal{W}_s, \boldsymbol{q}_0, \boldsymbol{q}_f$; a motion planner determines the longitudinal velocity profile $s(t)$ along $\mathcal{P}$ based on $\mathcal{W}_d$, and constructs reference trajectory $\mathcal{R}(t)$. The tracking controller, not a part of the planning module, ensures that the motion of the ego vehicle follows $\mathcal{R}(t)$ precisely.

A change in $\mathcal{W}_s$, for example when static obstacles turn into moving obstacles or vice versa, could invalidate the original path. To circumvent the appeal for path replanning, the following assumption is introduced.

Assumption 2.7: Static map $\mathcal{W}_s$ remains valid until the parking task is accomplished.

A simple device to meet Assumption 2.7 is taking $\mathcal{W}_s$ as a super-set of all possible static maps, i.e., $\mathcal{W}_s$ can be defined by marking all parking lots occupied (static obstacles) except the one corresponding to $\boldsymbol{q}_f$. This seemingly conservative treatment is meaningful because it is a good practice to avoid maneuvering over other unoccupied lots during parking.

Remark 2.8: The planning architecture, adopted in [30], [45], differs from the decomposition-based approach [53] where path planning deals with spatial constraints induced by static obstacles, and motion planning handles temporal constraints induced by system dynamics. Here, motion planning deals with dynamics as well as moving obstacles. □

III. Path Planning in Static Environments

This section presents the BIAGT algorithm to solve Problem 2.3. It differs from A* in both node expansion and node selection. Inspired by the idea that node priorities facilitate node selection and the desire to avoid expanding all nodes, we prioritize control actions which avoid expanding all actions during node expansion. A node can't be selected for expansion unless all actions have been applied, and during expansion, only the subset of control actions with the highest priority is applied. The second improvement is to estimate the cost-to-go by exchanging arrival cost information between two trees, e.g. the arrival costs of nodes on one tree are used to estimate the cost-to-go of nodes on the other tree.

TABLE I
Notations

Notation	Description	Notation	Description
$\mathcal{C}$	configuration space	$\mathcal{X}$	state space
$\boldsymbol{q}$	configuration	$\boldsymbol{x}$	system state
$\boldsymbol{x}_0$	initial state	$\boldsymbol{x}_f$	final state
$\boldsymbol{q}_0$	initial configuration	$\boldsymbol{q}_f$	final configuration
$\boldsymbol{u}$	control of (2)	$\mathbb{U}$	control space of (2)
$\boldsymbol{u}_p$	control of (3)	$\mathbb{U}_p$	control space of (3)
$\mathbf{x}$	state of (4)	$\mathbb{X}$	state space of (4)
$\mathbf{u}$	control of (4)	$\mathbb{U}_m$	control space of (4)
$\mathcal{W}_s$	static map	$\mathcal{W}_d$	dynamic map
$\mathcal{T}_s$	start tree	$\mathcal{T}_g$	goal tree
$\mathcal{T}_A$	tree selected to grow	$\mathcal{T}_B$	tree non-selected
$\mathcal{V}$	node set	$\mathcal{E}$	edge set
$\mathcal{M}$	motion primitives	$\mathcal{M}_i$	MPs of ith mode
$\mathcal{P}$	ego vehicle path	$\mathcal{R}$	ego vehicle motion
$\mathcal{P}_o$	obstacle path	$\mathcal{R}_o$	obstacle motion
t_f	final time	$\mathbf{Q}$	weighting matrix
$\mathcal{B}$	set of conflict areas	$\mathbb{B}_\epsilon(\boldsymbol{q})$	ϵ-neighbor of $\boldsymbol{q}$
n_o	num. of moving obstacles	$n_\mathcal{B}$	num. of CAs
$d(\cdot,\cdot)$	distance function	$F(\boldsymbol{q})$	F-value of $\boldsymbol{q}$
$g(\cdot,\cdot)$	arrival cost	$h(\cdot,\cdot)$	estimated cost-to-go

Notation: A node is referred to as a collision-free configuration $\boldsymbol{q}_i$. An edge $E(\boldsymbol{q}_i, \boldsymbol{q}_j)$ represents a feasible path between two nodes: $\boldsymbol{q}_i, \boldsymbol{q}_j$. A tree is a union of a node set $\mathcal{V}$ and an edge set $\mathcal{E}$, i.e., $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. A tree has a root node $\mathcal{T}.\boldsymbol{q}_0$ and a target node $\mathcal{T}.\boldsymbol{q}_f$. Start tree $\mathcal{T}_s$ has $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$ as its root and target node, respectively; goal tree $\mathcal{T}_g$ has $\boldsymbol{q}_f$ and $\boldsymbol{q}_0$ as its root node and target node, respectively. For a finite set $\mathcal{V}$, $|\mathcal{V}|$ is the number of elements. Given $\epsilon > 0$, an ϵ-neighboring ball of $\boldsymbol{q}_i$ is defined as $\mathbb{B}_\epsilon(\boldsymbol{q}_i) \triangleq \{\boldsymbol{q} | d(\boldsymbol{q}, \boldsymbol{q}_i) \leq \epsilon, \forall \boldsymbol{q} \in \mathcal{C}\}$, where $d(\cdot,\cdot)$ is a distance function, e.g. , a weighted 2-norm: $\|\boldsymbol{q}_i - \boldsymbol{q}_j\|_\mathbf{Q} = ((\boldsymbol{q}_i - \boldsymbol{q}_j)^\top \mathbf{Q} (\boldsymbol{q}_i - \boldsymbol{q}_j))^{1/2}$ where $\mathbf{Q}$ is a positive definite matrix with appropriate dimensions. Node $\boldsymbol{q} \in \mathcal{V}$ is assigned an F-value:

$$F(\boldsymbol{q}) = g(\mathcal{T}.\boldsymbol{q}_0, \boldsymbol{q}) + h(\boldsymbol{q}, \mathcal{T}.\boldsymbol{q}_f), \tag{5}$$

where $g(\mathcal{T}.\boldsymbol{q}_0, \boldsymbol{q})$ represents the arrival cost, or g-value, from $\mathcal{T}.\boldsymbol{q}_0$ to $\boldsymbol{q}$, and $h(\boldsymbol{q}, \mathcal{T}.\boldsymbol{q}_f)$ denotes the estimated cost-to-go, or h-value, from $\boldsymbol{q}$ to $\mathcal{T}.\boldsymbol{q}_f$. Note that $F(\boldsymbol{q})$ is an estimated cost of a potential path from $\mathcal{T}.\boldsymbol{q}_0$ to $\mathcal{T}.\boldsymbol{q}_f$ which passes through $\boldsymbol{q}$. Tree $\mathcal{T}$ maintains a priority queue Q of nodes to be expanded. All nodes in Q are ordered according to their F-values. Tree $\mathcal{T}$ is dead if $\mathcal{T}.Q$ is empty.

A. Motion Primitive, Mode, and Priority

Let the set of admissible control input signals to the kinematic model (3) be $\mathcal{U}_p = \{\boldsymbol{u}_p : \boldsymbol{u}_p(t) \in \mathbb{U}_p,\ \boldsymbol{q}(\boldsymbol{q}_0, t, \boldsymbol{u}) \in \mathcal{C},\ t \in [0, \infty)\}$, where $\boldsymbol{q}(\boldsymbol{q}_0, t, \boldsymbol{u})$ is the solution of (3) with initial conditions $\boldsymbol{q}(0)=\boldsymbol{q}_0$ and control input $\boldsymbol{u}$. An MP is defined by an admissible control input $\boldsymbol{u}$ and an associated travel distance l. Assume that the set of MPs, denoted by $\mathcal{M}$, has been pre-computed. Set $\mathcal{M}$ is partitioned into m subsets

$$\mathcal{M}=\bigcup_{1\le k\le m}\mathcal{M}_k$$
$$\mathcal{M}_i\cap\mathcal{M}_j=\emptyset,\ \forall 1\le i\ne j\le m,$$

where $\mathcal{M}_i$ represents the ith mode. For example, $\mathcal{M}$ can be simply categorized into two subsets: MPs moving forward and MPs moving backward. Alternatively, $\mathcal{M}$ can be split into $|\mathcal{M}|$ subsets, each of which contains one MP. At node $\boldsymbol{q}$, one assigns priority $p_{\boldsymbol{q}}^{\mathcal{M}_i}$ to mode $\mathcal{M}_i$. Mode $\mathcal{M}_i$ is *tried* if MPs in $\mathcal{M}_i$ have been applied. The set of indices of untried modes is denoted by $\mathcal{I}_u\subset\mathcal{I}\triangleq\{1,\dots,m\}$. If $\mathcal{I}_u=\mathcal{I}$, then $\boldsymbol{q}$ is fresh; if $\mathcal{I}_u=\emptyset$, we say $\boldsymbol{q}$ is fully expanded.

Remark 3.1: Modes can be inferred from physical insights or human experiences or more rigorously from the minimum principle, which allows us to derive the complete set of optimal arcs and the corresponding optimal controls as in [54]. The optimal controls can be clustered into modes. □

Remark 3.2: Partitioning $\mathcal{M}$ into modes and prioritizing modes allows mode selection, an analogy of the prioritizing nodes to enable node selection in A*. □

B. BIAGT Algorithm

BIAGT, described by Algorithms 1-2, grows trees $\mathcal{T}_s$ and $\mathcal{T}_g$ towards $\boldsymbol{q}_f$ and $\boldsymbol{q}_0$, respectively. It succeeds if any tree $\mathcal{T}$ reaches its end-game ball $\mathbb{B}_\epsilon(\mathcal{T}.\boldsymbol{q}_f)$, or both trees are connected. BIAGT contains 5 key parameters: ϵ is the radius of the end-game ball; K is the maximum number of iterations; μ is the threshold of distance indicating that two trees are in proximity; δ is the radius of $\mathbb{B}_\delta(\boldsymbol{q})$ defining the volume that node $\boldsymbol{q}$ exclusively occupies; γ is the radius of $\mathbb{B}_\gamma$ used in cost-to-go estimation.

In lines 2-3, both $\mathcal{T}_s$ and $\mathcal{T}_g$ are initialized, where the edge set and the priority queue are empty. At the kth iteration, `SelectTree` picks a tree $\mathcal{T}_A\in\{\mathcal{T}_s,\mathcal{T}_g\}$. If $\mathcal{T}_A.Q$ is empty, then BIAGT returns failure; otherwise, $\boldsymbol{q}_{\text{best}}$, the node with the lowest F-value in $\mathcal{T}_A.Q$, is identified (line 10). If $\boldsymbol{q}_{\text{best}}\notin\mathbb{B}_\epsilon(\mathcal{T}_A.\boldsymbol{q}_f)$, `Expand` conducts prioritized node expansion at $\boldsymbol{q}_{\text{best}}$ (line 12); otherwise BIAGT returns success. Node $\boldsymbol{q}_{\text{best}}$ is removed from $\mathcal{T}_A.Q$ if it is fully expanded.

`SelectTree`: chooses tree $\mathcal{T}_A$ according to the following rules. Case 1 (both $\mathcal{T}_s$ and $\mathcal{T}_g$ die): it returns null. Case 2 (one tree dies): the alive tree is selected. Case 3 (both trees are alive): if the distance between $\mathcal{T}_s$ and $\mathcal{T}_g$ is greater than μ, i.e., $d(\boldsymbol{q}_i,\boldsymbol{q}_j)>\mu,\forall\boldsymbol{q}_i\in\mathcal{T}_s.\mathcal{V},\forall\boldsymbol{q}_j\in\mathcal{T}_g.\mathcal{V}$, both trees are selected alternatively; otherwise, the tree having a lower estimated cost-to-go is selected.

`Expand`: expands node $\boldsymbol{q}_{\text{best}}$ according to $p_{\boldsymbol{q}_{\text{best}}}^{\mathcal{M}_i}, i\in\mathcal{I}_u$, as given by Algorithm 2. In line 3, `GetCurrentMode` determines the mode $\mathcal{M}_c$ with the highest priority by solving

$$\mathcal{M}_c=\text{argmin}_{i\in\mathcal{I}_u}\,p_{\boldsymbol{q}_{\text{best}}}^{\mathcal{M}_i}.$$

Applying $\text{MP}_k\in\mathcal{M}_c$ at $\boldsymbol{q}_{\text{best}}$ yields configuration $\boldsymbol{q}_k$ and path $\mathcal{P}_k$ from $\boldsymbol{q}_{\text{best}}$ to $\boldsymbol{q}_k$ (line 5). If $\mathcal{P}_k$ is collision-free and $\boldsymbol{q}_k$ is

Algorithm 1: BIAGT

```
1  input $q_0, q_f, \mathcal{M}, \epsilon, K, \mu, \delta, \gamma$;
2  $\mathcal{T}_s \leftarrow (\{q_0\},\emptyset), \mathcal{T}_s.q_0 = q_0, \mathcal{T}_s.q_f = q_f, \mathcal{T}_s.Q.$Push$(q_0)$;
3  $\mathcal{T}_g \leftarrow (\{q_f\},\emptyset), \mathcal{T}_g.q_0 = q_f, \mathcal{T}_g.q_f = q_0, \mathcal{T}_g.Q.$Push$(q_f)$;
4  $k \leftarrow 1$, $flag \leftarrow$ false;
5  while $k \le K$ and not $flag$ do
6      $(\mathcal{T}_A, \mathcal{T}_B) \leftarrow$ SelectTree$(\mathcal{T}_s, \mathcal{T}_g)$;
7      if $\mathcal{T}_A.Q.$Empty then
8          break;
9      else
10         $q_{\text{best}} = \mathcal{T}_A.Q.$Pop where
             $F(q_{\text{best}}) \le F(q), \forall q \in \mathcal{T}_A.Q$;
11         if $q_{\text{best}} \notin \mathbb{B}_\epsilon(\mathcal{T}_A.q_f)$ then
12             Expand$(\mathcal{T}_A, q_{\text{best}}, \mathcal{T}_B, \mathcal{M})$;
13         else
14             $flag \leftarrow$ true;
15     $k \leftarrow k+1$;
16 return $(\mathcal{T}_s, \mathcal{T}_g, flag)$;
```

δ-distance away from any node in $\mathcal{V}_A$ (lines 6), $\boldsymbol{q}_k$ is added to $\mathcal{T}_A$ (lines 7-12). By checking whether $\boldsymbol{q}_k$ is μ-distance away from any node in $\mathcal{V}_B$, its F-value is updated distinctively in line 9. Finally, $p_{\boldsymbol{q}_{\text{best}}}^{\mathcal{M}_c}$ is updated based on $F(\boldsymbol{q}_k)$.

Algorithm 2: Expand in BIAGT

```
1  input $\mathcal{T}_A, q_{\text{best}}, \mathcal{T}_B, \mathcal{M}$;
2  $(\mathcal{V}_A, \mathcal{E}_A) \leftarrow \mathcal{T}_A$, $(\mathcal{V}_B, \mathcal{E}_B) \leftarrow \mathcal{T}_B$;
3  $\mathcal{M}_c \leftarrow$ GetCurrentMode$(q_{\text{best}})$;
4  for MP$_k \in \mathcal{M}_c, 1 \le k \le |\mathcal{M}_c|$ do
5      $(q_k, \mathcal{P}_k) \leftarrow$ Simulate$(q_{\text{best}}, \text{MP}_k)$;
6      if $\min_{q\in\mathcal{V}_A} d(q_k, q) \ge \delta$ and CollisionFree$(\mathcal{P}_k)$
        then
7          if $\sim T_A.CloseToOtherTree$ and
             $\min_{q\in\mathcal{V}_B} d(q_k, q) \le \mu$ then
8              $T_A.CloseToOtherTree \leftarrow$ true;
9          UpdateCost$(q_k)$;
10         $\mathcal{V}_A \leftarrow \mathcal{V}_A \bigcup \{q_k\}, \mathcal{E}_A \leftarrow \mathcal{E}_A \bigcup E(q_{\text{best}}, q_k)$;
11         $\mathcal{T}_A.Q.$Push$(q_k)$;
12         UpdateModePriority$(F(q_k))$;
13 $\mathcal{T}_A \leftarrow (\mathcal{V}_A, \mathcal{E}_A)$;
14 return $\mathcal{T}_A$;
```

`UpdateCost`$(\boldsymbol{q}_k)$: calculates $F(\boldsymbol{q}_k)$ by checking the set:

$$\mathcal{V}_{\text{near}}(\boldsymbol{q}_k)\triangleq\{\boldsymbol{q}|\boldsymbol{q}\in\mathbb{B}_\gamma(\boldsymbol{q}_k)\bigcap\mathcal{V}_B\}.$$

If $\mathcal{V}_{\text{near}}=\emptyset$, `UpdateCost` computes $F(\boldsymbol{q}_k)$ in the same manner as A*; otherwise the following cost-to-go is used

$$h(\boldsymbol{q}_k,\mathcal{T}_A.\boldsymbol{q}_f)=\min_{\boldsymbol{q}_i\in\mathcal{V}_{\text{near}}(\boldsymbol{q}_k)}\{h(\boldsymbol{q}_k,\boldsymbol{q}_i)+g(\mathcal{T}_B.\boldsymbol{q}_0,\boldsymbol{q}_i)\},$$

where $g(\mathcal{T}_B.\boldsymbol{q}_0,\boldsymbol{q}_i)$ is accessed through $\mathcal{T}_B$. `UpdateCost` is illustrated by Fig. 3, where the start tree is in olive, the goal tree is in black, a circular obstacle is in gray, and the dots represent nodes. Suppose that the expansion of $\boldsymbol{q}_{\text{best}}\in\mathcal{T}_s.\mathcal{V}$ gives two admissible child nodes: $\boldsymbol{q}_{k1},\boldsymbol{q}_{k2}$. The green circle and magenta circle represent $\mathbb{B}_\gamma(\boldsymbol{q}_{k1})$ and $\mathbb{B}_\gamma(\boldsymbol{q}_{k2})$, respectively. We have $\mathcal{V}_{\text{near}}(\boldsymbol{q}_{k1})=\mathcal{V}_{\text{near}}(\boldsymbol{q}_{k2})=\{\boldsymbol{q}_g\}$ and thus

$$F(\boldsymbol{q}_{k1})=g(\boldsymbol{q}_0,\boldsymbol{q}_{k1})+h(\boldsymbol{q}_{k1},\boldsymbol{q}_g)+g(\boldsymbol{q}_f,\boldsymbol{q}_g)$$

$$F(\boldsymbol{q}_{k2}) = g(\boldsymbol{q}_0, \boldsymbol{q}_{k2}) + h(\boldsymbol{q}_{k2}, \boldsymbol{q}_g) + g(\boldsymbol{q}_f, \boldsymbol{q}_g).$$

Differently, A* updates F-values as follow

$$F_{A*}(\boldsymbol{q}_{k1}) = g(\boldsymbol{q}_0, \boldsymbol{q}_{k1}) + h(\boldsymbol{q}_{k1}, \boldsymbol{q}_f)$$
$$F_{A*}(\boldsymbol{q}_{k2}) = g(\boldsymbol{q}_0, \boldsymbol{q}_{k2}) + h(\boldsymbol{q}_{k2}, \boldsymbol{q}_f),$$

where $h(\boldsymbol{q}_{k1}, \boldsymbol{q}_f)$ and $h(\boldsymbol{q}_{k2}, \boldsymbol{q}_f)$ do not account for obstacles. With obstacle information being encoded in $g(\boldsymbol{q}_f, \boldsymbol{q}_g)$, $F(\boldsymbol{q}_{k2})$ and $F(\boldsymbol{q}_{k1})$ should give better estimate than $F_{A*}(\boldsymbol{q}_{k2})$ and $F_{A*}(\boldsymbol{q}_{k1})$, albeit not always. Consistently, tree $\mathcal{T}_s$ is more likely to grow toward $\mathcal{T}_g$; in contrast, A* tends to grow $\mathcal{T}_s$ by adding nodes in blue (toward the obstacle).

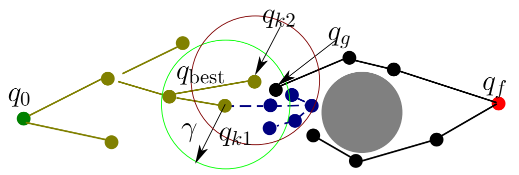


Fig. 3. Schematics to estimate cost-to-go of node $\boldsymbol{q}_{k1}$ on start tree (in olive) by using the arrival cost of nodes on goal tree (in black). The circular obstacle is in gray, and the dots represent nodes. The cost-to-go of $\boldsymbol{q}_{k1}$ is the sum of the length of the shortest Reeds-Shepp path [52] between $\boldsymbol{q}_{k1}$ and $\boldsymbol{q}_g$ and the arrival cost of $\boldsymbol{q}_g$ from $\boldsymbol{q}_f$.

`UpdateModePriority`: if $\boldsymbol{q}_{\text{best}}$ is fresh, $p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}}(0)$ takes the same value as that of its parent; otherwise, $p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}}$ is updated according to $F(\boldsymbol{q}_k)$. Assume that applying MPs in $\mathcal{M}_c$ gives $\mathcal{V}_c$: a set of nodes with $|\mathcal{V}_c| \geq 1$. Deterministic rules are exemplified below to attain repeatable planning outcomes. Similar to node priority, it is natural to define $p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}} \in \mathbb{R}_{\geq 0}$ according to the F-values of nodes in $\mathcal{V}_c$, i.e., $p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}} = \min_{\boldsymbol{q}_k \in \mathcal{V}_c} F(\boldsymbol{q}_k)$. Another simple rule is given by

$$p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}} = \begin{cases} 0, & \text{if } \min_{\boldsymbol{q}_k \in \mathcal{V}_c} F(\boldsymbol{q}_k) < F(\boldsymbol{q}_{\text{best}}) \\ 1, & \text{otherwise.} \end{cases}$$

We could also adopt the following rules

$$p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}} = \alpha p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}}(0) + (1-\alpha)\frac{1}{|\mathcal{V}_c|}\sum_{\boldsymbol{q}_k \in \mathcal{V}_c} F(\boldsymbol{q}_k),$$

where $\alpha > 0$ is the step size to be tuned. The aforementioned priority update rules ensure the boundedness of $p^{\mathcal{M}_c}_{\boldsymbol{q}_{\text{best}}}$.

The number of modes and associated nodes requires a careful design to balance exploration and exploitation, which entails physical insights into the specific problem. If $\mathcal{M}$ is divided into $|\mathcal{M}|$ modes, i.e., each mode contains one MP, BIAGT is reduced to greedy search.

C. Completeness

BIAGT does not incur a loss of feasibility compared to A*, i.e., it is resolution-complete under the same assumption. Assume that A* along with $\mathcal{M}$ is resolution complete. Then BIAGT is complete if all modes can be visited if necessary. This property is related to `GetCurrentMode` and the boundedness of mode priority. Since `GetCurrentMode` locates the mode with the highest priority among untried modes, any mode can be visited as long as the priority is finite. We further assume that K is arbitrarily large, neglecting the memory and computation time restriction.

Proposition 3.3: BIAGT is resolution-complete.

Proof: Let us consider the completeness for a special case of BIAGT, where only one tree is constructed. As long as the special BIAGT is complete, then the general BIAGT, growing two trees, is also complete.

Sufficiency is shown by contradiction, i.e., assume that A* is complete, but BIAGT is not. That is, for some pair $(\boldsymbol{q}_0, \boldsymbol{q}_f)$, A* constructs tree $\mathcal{T}_{A*}$ and gives a set of feasible paths $\mathcal{P}$ connecting $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$, albeit BIAGT fails to return any feasible path. Take an arbitrary path $\mathcal{P}_t \in \mathcal{P}$ and assume that it passes through a set of nodes $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N\}$ in $\mathcal{T}_{A*}$. BIAGT fails only if at least one node $\boldsymbol{q}_j \in \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N\}$ either does not belong to $\mathcal{T}_{BIAGT}$, or it is in $\mathcal{T}_{BIAGT}$ but not selected for expansion, or not expanded with the correct mode.

Case $\boldsymbol{q}_j \in \mathcal{T}_{BIAGT}$. The F-value of $\boldsymbol{q}_j$ is finite because it is collision-free and accessible from $\boldsymbol{q}_0$ through a finite number of nodes $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}\}$. Because the K-layer reachable tree $\mathcal{T}$ contains a finite number of nodes and $\mathcal{T}_{BIAGT} \subset \mathcal{T}_K$, there are a finite number of nodes in $\mathcal{T}_{BIAGT}$ with F-value less than $F(\boldsymbol{q}_j)$. Hence, node $\boldsymbol{q}_j$ will be selected for expansion in finite time. Also, $\boldsymbol{q}_j$ will be removed from the Q queue only if all modes are tried and $\boldsymbol{q}_{j+1}$ (child of $\boldsymbol{q}_j$ as a result of applying a certain primitive in $\mathcal{M}$ to $\boldsymbol{q}_j$) is added to $\mathcal{T}_{BIAGT}$. That is to say: if $\boldsymbol{q}_j \in \mathcal{T}_{BIAGT}$, so does its child $\boldsymbol{q}_{j+1}$. By induction, we have $\{\boldsymbol{q}_j, \ldots, \boldsymbol{q}_n\}$ belongs to $\mathcal{T}_{BIAGT}$, which implies the existence of path $\mathcal{P}_t$ in $\mathcal{T}_{BIAGT}$. A contradiction is concluded.

Case $\boldsymbol{q}_j \notin \mathcal{T}_{BIAGT}$. In this case, it is necessary that node $\boldsymbol{q}_{j-1} \notin \mathcal{T}_{BIAGT}$. This is because, if $\boldsymbol{q}_{j-1} \in \mathcal{T}_{BIAGT}$, it will be selected for expansion; all primitives will be applied in the worst case; and $\boldsymbol{q}_j$ will be generated and added to $\mathcal{T}_{BIAGT}$. By induction, we conclude that $\{\boldsymbol{q}_0, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$ does not belong to $\mathcal{T}_{BIAGT}$. This leads to a contradiction, because $\boldsymbol{q}_0 \in \mathcal{T}_{BIAGT}$. Hence, we have $\boldsymbol{q}_j \in \mathcal{T}_{BIAGT}$ for all $1 \leq j \leq N$, and thus tree $\mathcal{T}_{BIAGT}$ contains path $\mathcal{P}_t$. ■

Remark 3.4: BIAGT requires an overestimate of the cost-to-go, which implies sub-optimality. With $h(\boldsymbol{q}_k, \boldsymbol{q}_f)$ being the lower bound of the cost-to-go, the node selection of BIAGT is the same as in A*,

$$\begin{aligned} F(\boldsymbol{q}_{\text{best}}) &= g(\boldsymbol{q}_0, \boldsymbol{q}_{\text{best}}) + h(\boldsymbol{q}_{\text{best}}, \boldsymbol{q}_f) \\ &\leq g(\boldsymbol{q}_0, \boldsymbol{q}_{\text{best}}) + c(\boldsymbol{q}_{\text{best}}, \boldsymbol{q}_k) + h(\boldsymbol{q}_k, \boldsymbol{q}_f) \\ &= F(\boldsymbol{q}_k), \end{aligned}$$

and `Expand` likely applies all MPs in $\mathcal{M}$ at $\boldsymbol{q}_{\text{best}}$. In other words, BIAGT necessitates an inflated heuristic cost to show computational benefits: $\rho h(\boldsymbol{q}_k, \boldsymbol{q}_f)$ with $\rho > 1$. The result is that BIAGT is sub-optimal, although in practice it might produce a smaller tree and perform faster than A*. Interested readers are referred to [20] for simulation validation. Note that a similar principle has been exploited to achieve anytime planning [17]. □

D. Discussions and Improvements

In A*, each node gets one chance to expand. This means all MPs must be applied during node expansion. Recalling why A* outperforms breadth-first search by prioritized node selection, the idea of applying all MPs is apparently neither necessary nor efficient. BIAGT weakens this limitation by allowing mode selection: a node can be expanded multiple times, and only the MPs in the selected mode are applied each time. BIAGT is incentivized by the realization that vehicle paths typically consist of several arcs, and each arc is formed by a sequence of MPs in the same mode.

The efficacy of BIAGT depends on whether it can correctly

(I) pick the best node $\boldsymbol{q}_{\text{best}}$ to expand; and
(II) determine the best mode $\mathcal{M}_c$ to apply.

As a variant of A*, node selection of BIAGT is purely determined by how accurate the estimated cost-to-go is. Allowing $\mathcal{T}_g$ and $\mathcal{T}_s$ to exchange arrival costs facilitates early accommodation of distant obstacles into cost-to-go estimates. How early distant obstacles are accommodated is controlled by parameter μ. Mode selection is dictated by how the mode priority is inferred in `UpdateModePriority`. This is difficult because the correct mode is largely influenced by the overall layout of obstacles. The priority updating rules recursively update the mode priority based on local obstacle information and global cost-to-go, and work well only if the cost-to-go captures global information.

Parameters δ and l are related to memory consumption and completeness. For embedded platforms, it is crucial that the nodes of the tree form a sparse distribution over $\mathcal{C}_{\text{free}}$. Works [2], [48] utilize pre-defined state lattices to prevent over-exploration (densely populated nodes), which compromises path quality and results in a large array to mark the state lattice. BIAGT fulfills the sparsity requirement by rejecting $\boldsymbol{q}_k$ within δ-distance from the same tree. The distance check results in less memory but extra computation, which is non-negligible for a big tree. There is no clear-cut way to determine what value δ should take, and thus, tuning δ may involve trial and error. Roughly speaking, δ defines the resolution used to partition $\mathcal{C}_{\text{free}}$ and is related to resolution-completeness. The smaller δ is, the larger the tree, whereas a larger δ results in a smaller tree but at a higher probability of losing completeness. Impacts of l are similar to δ. A fixed δ, implying uniform node density, potentially wastes computation and memory resources. In fact, when the vehicle moves in a relatively open space, feasibility is barely affected by trying aggressive MPs and actively rejecting node candidates, and thus, one can afford to use a large δ and long MPs; in a tight space, feasibility is likely at risk due to aggressive rejection, and thus a smaller δ and shorter MPs are appropriate. It is highly desirable to adjust parameters δ and l during tree construction to balance computation efficiency, memory, and completeness. The details about how to adjust δ and l are omitted because it is not the focus of this work.

IV. Safe Motion Planning in Dynamic Environments

Path planning assumes static environments to lower the computational complexity. This section presents a hierarchical motion planner to deal with dynamic environments with safety guarantees at a reasonable computational burden. At the scheduling level, the results of [46] are employed with slight modifications for parking. Based on the scheduling decision, analytical formulae are applied to generate vehicle trajectories.

A. Conflict Area

We recite concepts previously used in [46] for the sake of completeness. To characterize potential collisions with moving obstacles, the CA (where the paths of vehicle/obstacles intersect or overlap) is introduced below.

Definition 4.1: Given the path of the ego vehicle $\mathcal{P}$ and the occupancy set defined by the length and width of the ego vehicle $\mathcal{S}$, and those of moving vehicles $\mathcal{P}_o$ and $\mathcal{S}_o$ for $o = 1, \ldots, n_o$, where n_o is the number of all moving obstacles, the conflict area $\mathcal{B} \subset [0, s_f(\mathcal{P})]$ is defined as

$$\mathcal{B} := \{s \in [0, s_f(\mathcal{P})] : \exists o \in \{1, \ldots, n_o\}, s_o \in [0, s_f(\mathcal{P}_o)] \text{ such that } (\mathcal{P}(s) \oplus \mathcal{S}) \cap (\mathcal{P}_o(s_o) \oplus \mathcal{S}_o) \neq \emptyset\}, \quad (6)$$

where $\oplus$ is the Minkowski sum.

If $\mathcal{P}$ coincides with the paths of obstacles more than once, $\mathcal{B}$ may not be a connected set and can be written as the union of connected sets. More precisely, we let

$$\mathcal{B} = \bigcup_{i=1}^{n_{\mathcal{B}}} \mathcal{B}_i,$$

where $\mathcal{B}_i$ is a connected set in $\mathbb{R}$. We say that there are $n_{\mathcal{B}}$ conflict areas and refer to each conflict area (by default along the vehicle path) as CA$_i$. For notional simplicity, let

$$\inf \mathcal{B}_i = a_i, \qquad \sup \mathcal{B}_i = b_i,$$

and say $\mathcal{B}_i$ has a range $[a_i, b_i]$. CAs are illustrated in Fig. 4.

One can define CAs for each vehicle based on the paths and sizes of vehicles and obstacles. Notably, one segment of the vehicle path could intersect with multiple obstacle paths, and we should be careful to ensure any point on the vehicle path belongs to only one CA.

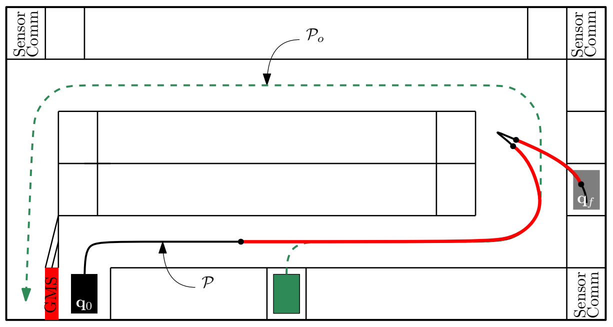


Fig. 4. Red segments are the CAs by Definition 4.1. Ego vehicle (solid black box) plans a path $\mathcal{P}$ in solid line connecting $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$, while an obstacle (solid green box) executes a path $\mathcal{P}_o$ in dash green. This results in two disconnected CAs, represented by red solid lines.

B. Reformulation of Problem 2.5

Safe motion planning amounts to avoiding the overlap with moving obstacles inside the same CA$_i$ at any time, which can be formulated as a temporal constraint on an input signal $\mathbf{u} \in \mathcal{U}_m$. Here, $\mathcal{U}_m$ is defined as the set of admissible control input signals to the longitudinal dynamics (4), that is, $\mathcal{U}_m \triangleq \{\mathbf{u} : \mathbf{u}(t) \in \mathbb{U}_m,\ \mathbf{x}(\mathbf{x}_0, t, \mathbf{u}) \in \mathbb{X},\ t \in [0, \infty)\}$, where $\mathbf{x}(\mathbf{x}_0, t, \mathbf{u})$ is a solution of (4) with control input $\mathbf{u}$.

We formulate such a temporal collision-avoidance constraint in terms of the time intervals during which moving obstacles and the ego vehicle occupy the same CA$_i$. Based on the motion trajectories of all moving obstacles, one computes time intervals $\mathcal{I}_{o,\mathcal{B}_i} \subset \mathbb{R}_{\geq 0}$ during which obstacle o occupies CA$_i$ by

$$\mathcal{I}_{o,\mathcal{B}_i} := \{t \in \mathbb{R}_{\geq 0} : \mathcal{R}_o(t) \in \mathcal{B}_i\} \tag{7}$$

where $\mathcal{R}_o$ is the trajectory of obstacle o. Also, for the ego vehicle and for each CA$_i$, define $t_{\mathcal{B}_i} \subset \mathbb{R}_{\geq 0}$ as a time interval during which the ego vehicle is inside the CA$_i$, that is, $\{t : s(t, \mathbf{u}) \in \mathcal{B}_i\}$. We can simply use t_{a_i} to denote the time when the vehicle enters $\mathcal{B}_i$. Then, to avoid collisions inside each CA$_i$, we should impose

$$t_{\mathcal{B}_i} \cap \mathcal{I}_{o,\mathcal{B}_i} = \emptyset \quad \text{for all } o. \tag{8}$$

Problem 2.5 is recast to the following scheduling problem:

*Problem 4.2 (*SP*):* Given a path $\mathcal{P}$ and a dynamic map $\mathcal{W}_d$ (which gives the value of $\mathcal{I}_{o,\mathcal{B}_i}$ for $1 \leq i \leq n_{\mathcal{B}}$),

$$\min_{\mathbf{t}_{1:n_{\mathcal{B}}}} \quad \mathrm{f}(\mathbf{t}_{1:n_{\mathcal{B}}}) \tag{9a}$$

$$\text{subject to} \quad t_{\min}(a_i, \mathbf{t}_{1:i-1}) \leq t_{a_i} \leq t_{\max}(a_i, \mathbf{t}_{1:i-1}) \tag{9b}$$

$$t_{\mathcal{B}_i} \cap \mathcal{I}_{o,\mathcal{B}_i} = \emptyset \quad \text{for all } o, i, \tag{9c}$$

where $\mathbf{t}_{1:n_{\mathcal{B}}} = (t_{a_1}, \ldots, t_{a_{n_{\mathcal{B}}}})$ and

$$t_{\min}(a_i, \mathbf{t}_{1:i-1}) := \min_{\mathbf{u} \in \mathcal{U}_m} \{t : s(t, \mathbf{u}) = a_i \text{ with constraint } s(t_{a_{i-1}}, \mathbf{u}) = a_{i-1}\} \tag{10a}$$

$$t_{\max}(a_i, \mathbf{t}_{1:i-1}) := \max_{\mathbf{u} \in \mathcal{U}_m} \{t : s(t, \mathbf{u}) = a_i \text{ with constraint } s(t_{a_{i-1}}, \mathbf{u}) = a_{i-1}\}. \tag{10b}$$

By definition, $t_{\min}(a_i, \mathbf{t}_{1:i-1})$ and $t_{\max}(a_i, \mathbf{t}_{1:i-1})$ are the soonest and latest, respectively, time when the vehicle can enter CA$_i$ according to the dynamics (4), given that it has entered CA$_{i-1}$ at a given time $t_{a_{i-1}}$.

In short, Problem 4.2 solves for a set of times when the vehicle enters each CA. It is a special form of scheduling problem where only one job is scheduled over $n_{\mathcal{B}}$ machines (i.e., CAs), where each machine should be kept idle during specific time intervals $\mathcal{I}_{o,\mathcal{B}_i}$. In this paper, we use the approximate solution developed in [46]. The idea is to restrict the problem such that it consists of one decision variable t_{a_1} and several CA-related constraints, which are expressed as (linear) functions of t_{a_1}. The approximate problem can be formulated as MILP. Readers are referred to [46] for more details.

Given the approximate solution $\mathbf{t}^*_{1:n_{\mathcal{B}}} = (t^*_{a_1}, \ldots, t^*_{a_{n_{\mathcal{B}}}})$, we can compute an input signal $\mathbf{u} \in \mathcal{U}_m$, such that

$$s(t^*_{a_i}, \mathbf{u}) = a_i, \quad \text{for all } i = 1, \ldots, n_{\mathcal{B}}.$$

To find such an input signal $\mathbf{u}$, we analytically solve the above equations subject to velocity and acceleration constraints. For example, the conventional trapezoidal velocity profile can be readily generated based on arc length, initial and final velocities, and travel time.

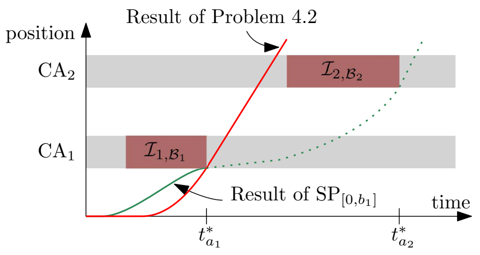


Fig. 5. Decoupling example when $a_2 - b_1 \geq \ell$. With $\mathrm{f}(\mathbf{t}_{1:n_2}) = t_{a_1} + t_{a_2}$, the solution of SP$_{[0,b_1]}$ ensures the existence of a feasible solution of Problem 4.2; that is, there exists a dotted trajectory where $(t^*_{a_1}, t^*_{a_2})$ is a feasible solution of Problem 4.2. However, the solution of SP$_{[0,b_1]}$ does not guarantee optimality because its resulting trajectory may not be part of a trajectory resulting from the solution of Problem 4.2. Proposition 4.4 provides a condition that guarantees the optimality.

C. Decomposed CA

Ranges of CAs affect the efficiency of the approximate solution. We categorize CAs into two types: intersection and overlap. Intersection type means that paths of two vehicles cross with each other, and the CA typically spans over a small region. Overlap type characterizes the scenario shown in Fig. 1 where infrastructure admits one-way traffic. All vehicles move in the same direction and likely have paths overlapping with each other. Take two outgoing vehicles as an example, where vehicle B is behind vehicle A. Hence, based on spatial constraint, the corresponding CA covers the entire path of vehicle A. With this CA definition, solving Problem 4.2 for vehicle B ends up with the solution that vehicle B cannot start moving until vehicle A drives out of the garage. This is not acceptable because of its remarkably low throughput.

We propose to decompose the overlap CA into a series of CAs which are exclusive and possess the nature of accounting for the temporal constraint. The downside of this treatment is the increase in the number of decision variables and the resulting increase in computational complexity of Problem 4.2. Note that the resulting approximate problem requires solving for one decision variable t_{a_1}, and thus the increase of computational complexity is less severe than Problem 4.2.

D. Decoupled Approach

As the number of CAs grows, the approximate algorithm tends to be more restrictive because it adjusts only t_{a_1} to avoid collision in the subsequent CAs. Moreover, its computation time increases exponentially. To address these issues, we propose to decouple the problem into subproblems and solve them sequentially.

Let SP$_{[0,s_0]}$ be Problem 4.2 restricted to the CAs that are located before s_0, i.e., $b_i \leq s_0$. This effectively changes the range of i to $1, \ldots, \iota$ if $b_\iota \leq s_0$ is the closest CA to s_0, and therefore the decision variable becomes $\mathbf{t}_{1:\iota} = (t_{a_1}, \ldots, t_{a_\iota})$. We can solve the problem SP$_{[0,b_\iota]}$, instead of Problem 4.2, ignoring the subsequent CAs, in the following cases.

(I) Suppose that two consecutive CAs ι and $\iota+1$ are more than distance ℓ apart (i.e., $a_{\iota+1} - b_\iota \geq \ell$), and ℓ satisfies

$$\ell \geq \text{minimum distance required to stop for any velocity} = \min_t \{s(t) - s(0) : v(t) = 0,\ \forall |v(0)| \leq v_{\max}\}.$$

The condition is not too restrictive because the maximum velocity is low during parking maneuvers.

(II) Suppose the path consists of a position $\mathcal{P}(s_r)$ where the vehicle reverses its direction, and s_r is not an entry of any $\mathcal{B}_i$ and $b_\iota \leq s_r$ is the closest conflict area.

One common and interesting feature of these two cases is that $t_{\max}(a_{\iota+1}, \mathbf{t}_{1:\iota}) = \infty$ for any feasible $\mathbf{t}_{1:\iota}$.

The decoupled approach preserves feasibility and, furthermore, optimality under some conditions.

Proposition 4.3: Let $\mathbf{t}^*_{1:\iota}$ be the solution of $\text{SP}_{[0,b_\iota]}$. Then, there exists $\mathbf{t}^*_{\iota+1:n_\mathcal{B}}$ such that the concatenated vector $\mathbf{t}^*_{1:n_\mathcal{B}}$ is feasible for Problem 4.2.

Proof: (Sketch) Problem 4.2 has the constraints of $\text{SP}_{[0,b_\iota]}$, the constraints of $\text{SP}_{[b_\iota,s_f]}$, and

$$t_{\min}(a_{\iota+1}, \mathbf{t}_{1:\iota}) \leq t_{a_{\iota+1}} \leq t_{\max}(a_{\iota+1}, \mathbf{t}_{1:\iota}), \qquad (11)$$

where $\mathbf{t}_{1:\iota}$ is any feasible solution of $\text{SP}_{[0,b_\iota]}$. Because $t_{\max}(a_{\iota+1}, \mathbf{t}_{1:\iota}) = \infty$ by definition of ι, and $\mathcal{I}_{o,\mathcal{B}_i}$ is finite, there exists $\mathbf{t}^*_{\iota+1:n_\mathcal{B}}$ satisfying the constraints of $\text{SP}_{[b_\iota,s_f]}$ and (11). ■

The example illustrated in Fig. 5 shows that the optimality cannot be guaranteed by the decoupled approach; think about the decomposition of $\text{SP}_{[0,b_1]}$ and $\text{SP}_{[b_1,s_f]}$ in the example by assuming that $t_{\max}(a_2, t_1) = \infty$ for any t_1. However, under some conditions, optimality can be guaranteed.

Proposition 4.4: If $\max v(t^*_\iota)$ is equal to $v_{\max}$, then there exists $\mathbf{t}^*_{\iota+1:n_\mathcal{B}}$ such that $\mathbf{t}^*_{1:n_\mathcal{B}}$ is optimal for Problem 4.2.

Proof: (Sketch) Suppose $\mathbf{t}^*_{\iota+1:n_\mathcal{B}}$ is the optimal solution for $\text{SP}_{[b_\iota,s_f]}$ with constraint (11), which exists by Proposition 4.3. The condition $\max v(t^*_\iota) = v_{\max}$ implies that for any feasible solution $\mathbf{t}_{1:\iota}$ of $\text{SP}_{[0,b_\iota]}$,

$$t_{\min}(a_{\iota+1}, \mathbf{t}^*_{1:\iota}) \leq t_{\min}(a_{\iota+1}, \mathbf{t}_{1:\iota}).$$

So the constraint (11) is no more restrictive than the one in $\text{SP}_{[0,s_f]}$, and thus, $\mathbf{t}^*_{1:n_\mathcal{B}}$ is the optimal solution. ■

V. Case Studies

In the simulation, the proposed solution is applied to Problem 2.2 for a garage as shown in Fig. 1. The garage contains four rows of perpendicular parking lots, $\{A_1, \ldots, A_{10}, B_1, \ldots, B_{10}, C_1, \ldots, C_{10}, D_1, \ldots, D_{14}\}$ and four parallel parking lots $\{E_1, \ldots, E_4\}$, all being abstracted as rectangles. Each parking lot has a dimension of 2.5m $\times$ 6m, the lane width is 6m, and the garage size is 40m$\times$36m. All vehicles and moving obstacles have a length of 4.655m, a width of 1.810m, and a minimum turning radius of 4.132m. The following two scenarios are considered:

(I) Case 1: one obstacle vehicle drives out of A_5, followed by the ego vehicle driving into E_2; The obstacle moves at a speed $1 + v_n(t)$m/sec with $v_n(t)$ uniformly distributed over $[0, 0.01]$.

(II) Case 2: two obstacle vehicles leave A_5 and D_{10}, respectively, followed by the ego vehicle driving out of E_2. Obstacle 1 moves at a speed $1 + v_{n_1}(t)$m/sec and obstacle 2 moves at a speed $0.5 + v_{n_2}(t)$m/sec with $v_{n_1}(t), v_{n_2}(t)$ uniformly distributed over $[0, 0.01]$.

The ego vehicle has $v_{\max} = 1.5$m/sec, and $u_1 \in [-1, 1]$m/sec^2. The configurations of the entrance, the exit, A_5, E_2, and D_{10} are given as follows

$$\begin{aligned} \boldsymbol{q}_{\text{entr}} &= [9.5, 1.5, \pi/2]^\top \\ \boldsymbol{q}_{\text{exit}} &= [3, 4, 3\pi/2]^\top \\ \boldsymbol{q}_{A_5} &= [23.75, 1.5, \pi/2]^\top \\ \boldsymbol{q}_{E_2} &= [38.75, 13.5, \pi/2]^\top \\ \boldsymbol{q}_{D_{10}} &= [26.25, 34.5, 3\pi/2]^\top. \end{aligned}$$

All simulations are conducted in Matlab®2020b, where the path planning Algs. 1-2 can be implemented in Matlab without using any toolbox, and the safe motion planning algorithms are coded based on Matlab Optimization Toolbox. Please note extensive simulation validation for path planning has been conducted in [20] to verify computational efficiency against classical A* algorithm, and thus omitted here.

For both cases, we compare three algorithms: regular, decomposed, and decoupled, where regular solves Problem 4.2 without sub-division of CAs, decomposed solves Problem 4.2 with sub-division of CAs, and decoupled solves a sequence of sub-motion planning problems as in Proposition 4.3. All three are bench-marked against a baseline: minimum time optimal control method (MTOC), which generates time-optimal motion without considering moving obstacles. We assess algorithms in terms of task time, energy consumption, and computation time, where task time and energy consumption indicate the time and energy taken for the vehicle to accomplish the task, respectively.

Tables II-III summarize the comparison results, where *MILP time* indicates the time taken to solve one instance of the MILP as a result of Problem 4.2 or $\text{SP}_{[0,s_0]}$, and *MILP calls* counts the total number of MILPs solved during the task. It is noteworthy that, ideally, the MILP can be called once for all three algorithms. However, we keep solving the MILP during the motion execution for the flexibility to adjust the schedule, aiming to address stochastic uncertainties in obstacle velocities. Also, numerical values in both tables are normalized based on the regular algorithm result. Particularly, in Table III, the energy consumption for the regular algorithm without considering idling is 1, and all others are normalized against it. It is worth mentioning that for both cases, it takes less than 0.02sec to solve one MILP instance on a computer with Intel® i7-4790K and 32GB RAM, and thus, the safe motion planning can run in real-time if the number of moving obstacles is small.

From both tables, MTOC outperforms the other three algorithms, achieving the shortest task time, lowest computational complexity, and minimal energy consumption. This is expected because MTOC, ignoring the moving obstacles, returns a motion plan that does not involve waiting. It involves a purely analytical formula for motion planning, and the MILP time and calls are zero.

From Table II, both the decomposed and decoupled algorithms shorten task time, compared to the regular, but the percentage of improvement depends on the task. MILP times are about the same for Case 1, whereas the decomposed takes longer to solve for Case 2. It is understood that for Case 1,

TABLE II
COMPARISON ON TIME DIMENSION

Alg.	Case	Task time	MILP time	MILP calls
MTOC	1	0.6770	0	0
regular	1	1	1	1
decomposed	1	0.7179	0.9749	1
decoupled	1	0.7160	0.9838	1.2624
MTOC	2	0.2493	0	0
regular	2	1	1	1
decomposed	2	0.2670	1.1977	0.1734
decoupled	2	0.2606	1.0047	0.1753

the regular, decomposed, and decoupled algorithms involve one CA, four CAs, and two CAs, respectively, which is not significantly different; for Case 2, the three algorithms use two CAs, 17 CAs, and two CAs respectively, and the decomposed algorithm has to deal with many more constraints. Regarding MILP calls, the comparison conveys a mixed message because it depends on the task, which becomes clear by examining the figures below. The main takeaway is that the decomposed algorithm lands the least calls.

For the energy consumption results in Table III, column Ignoring idle and column Including idle are obtained by calculating the following integral

$$E(\mathbf{u}) = \int_0^T \max(c^2, a^2)\mathrm{d}t, \tag{12}$$

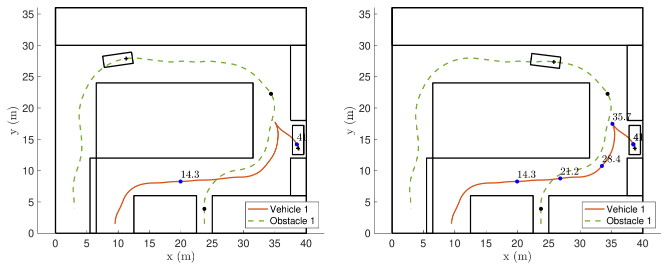
with $c = 0$ and $c = 0.2$, respectively. Here, c denotes the power during idle, a is the acceleration of the vehicle, and T is the task time. Among these three algorithms, the decomposed results in the most efficient task execution for both cases. Energy benefit brought by the decoupled algorithm varies over cases. If one ignores energy consumption during idle, the following motion by the regular algorithm is more efficient than the decoupled because the latter involves more stop-and-go during the task. The consideration of idling energy consumption could change the whole story because the decoupled solution may finish tasks earlier, idle less, and reduce the incurred energy consumption. Whether the energy savings due to reduced idle time outweigh the penalty incurred by extra braking is task-dependent. We remark that the energy consumption model in the evaluation is overly simplified for illustrative purposes.

TABLE III
COMPARISON ON ENERGY CONSUMPTION DIMENSION

Alg.	Case	Ignoring idle	Including idle
MTOC	1	0.8611	0.9503
regular	1	1	1.1379
decomposed	1	0.8827	0.9719
decoupled	1	1.1108	1.1946
MTOC	2	1.0962	1.2467
regular	2	1	1.7393
decomposed	2	0.9560	1.1140
decoupled	2	1.3315	1.4781

Comprehensive results are reported in Figs. 6-9 for Case 1, and in Figs. 10-12 for Case 2, respectively. All figures, unless explicitly specified, adopt the following color code: solid black, dash blue, and dash green represent the regular, decomposed, and decoupled algorithms, respectively; blue dots denote the beginnings and ends of CAs along the vehicle path; black dots denote the beginnings and ends of CAs along obstacle paths. Fig. 6 defines the garage layout, the paths, and CAs for both regular and decomposed algorithms. The layout for the decoupled algorithm is similar to Fig.6(b), except that it involves only the first two CAs at the beginning of the task and thus is omitted. Fig. 7 shows that the decoupled solution leads to the decomposed. This is not always true though, and we have witnessed in simulations that the decomposed motion does catch up. This discrepancy is largely due to the stochastic nature of the obstacles' speed. Nevertheless, a consistent observation is that the task time difference between the decomposed and decoupled solutions is subtle, and the former always lags in the beginning. This is anticipated because the decoupled algorithm determines the schedule based on a subset of CAs.

Examining Figs. 7-8 offers a clear picture of the schedules and movements of the vehicle. The regular algorithm schedules the vehicle to enter the large CA at [14.3,41]m after the obstacle clears it; the decomposed schedules the vehicle into CA$_1$ when it is safe to finish the rest of the path without further waiting, i.e., the vehicle is guaranteed to clear all remaining CAs with the MTOC planner safely; and the decoupled schedules the vehicle into CA$_1$ when it can safely clear the foreseeable CAs. The number of foreseeable CAs for a given vehicle state is tunable. In simulation, the foreseeable CAs are defined as the set comprised of two CAs ahead and behind the vehicle.



(a) Regular (b) Decomposed

Fig. 6. Case 1: paths and CAs. Red solid: vehicle path; green dash: obstacle path. (a) A large CA along the vehicle path ranges from 14.3m to 41m, abbreviated as [14.3,41]m, indicating the beginning and end of the CA relative to the vehicle's initial position. (b) The large CA, ranging [14.3,41]m, is decomposed into four CAs: CA$_1$-[14.3, 21.2]m, CA$_2$-[21.2, 28.4]m, CA$_3$-[28.4,35.7]m, and CA$_4$-[35.7,41]m along the vehicle path. The four CA$_i$, $1 \leq i \leq 4$ correspond to four CAs along the obstacle path, CA$_1^O$-[-2.4,8.7]m, CA$_2^O$-[8.7,15.2]m, CA$_3^O$-[15.2,21.6]m, CA$_4^O$-[21.6,26.4]m, where numerical values are relative to obstacle's initial position. CA$_i^O, 1 \leq i \leq 4$ are not shown to avoid confusion.

Fig. 9 visualizes the obstacle and the vehicle positions at moments when the vehicle is about to enter a CA. The horizontal bar on the top shows CA$_i^O, 1 \leq i \leq 4$, and the lower bar plots CA$_i, 1 \leq i \leq 4$. The black, blue, yellow, and red pentagrams show the beginning of the 1st, 2nd, 3rd, and 4th CA, respectively. The black, blue, yellow, and red dots denote obstacle positions when the vehicle is at the pentagram in black, blue, yellow, and red, respectively. Apparently, no obstacle and vehicle share the same CA at any time. The

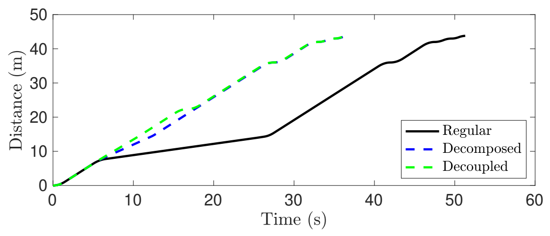


Fig. 7. Case 1: vehicle position trajectories.

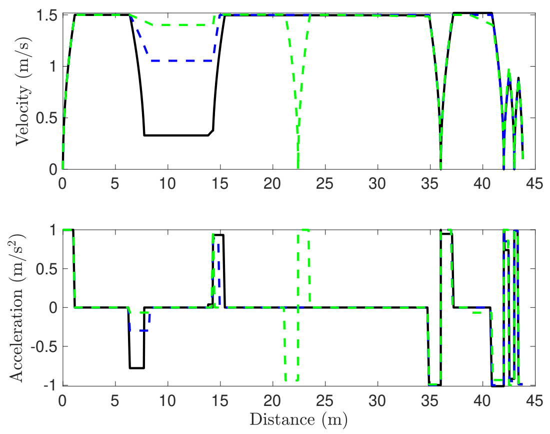


Fig. 8. Case 1: vehicle velocity & acceleration profiles along its path. The vehicle slows down before entering CA$_1$. Three algorithms give different schedules for the vehicle to pass 14.3m. The regular algorithm is the most conservative because the vehicle has to wait for the obstacle to clear the large CA [14.3, 41]m; the decomposed is the second most conservative, where the vehicle can pass 14.3m if it is safe to execute the rest path without further waiting; and the decoupled is the least conservative, where the vehicle is allowed to pass 14.3m as long as no collision happens in the two CAs ahead. Hence, velocity in solid black slows down most, and green dashes barely slow down before CA$_1$. Afterward, velocity profiles for both regular and decomposed algorithms do not contain extra waiting before any CA, whereas the velocity from the decoupled algorithm goes to zero in the middle of CA$_2$, to avoid entering CA$_3$ before the obstacle clears.

vehicle slows down after entering CA$_2$ to avoid entering CA$_3$ before the obstacle clears CA$_3^O$.

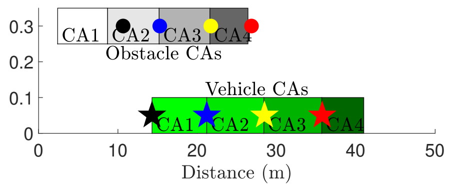


Fig. 9. Decoupled algorithm for Case 1: positions of vehicle and obstacles when the former is about passing a CA.

Case 2 involves 17 CAs for the decomposed and decoupled algorithms, which makes it difficult to make a legible visual presentation. Hence, Fig. 10 only plots the two CAs used in the regular algorithm. Temporal position trajectories are depicted in Fig. 11, where two observations can be made: first, consistent with Table II and Case 1, the regular algorithm ends up with a long waiting time before entering CA$_2$, and thus finishes the task late; second, the decoupled algorithm initially leads the decomposed, but slows down at $t \approx 40$sec and thus is overtaken by the latter. Fig. 12 elucidates how the vehicle moves along the path according to different algorithms. Since the vehicle starts with a parallel parking maneuver involving several cusps, the velocity profiles look messy before CA$_2$. Profiles in the first 10m are magnified and shown in magenta boxes. By looking at the upper magenta box, we noticed similar patterns observed in Case 1, i.e., the regular algorithm issues a complete stop before CA$_2$, the decoupled does not wait at all, and the decomposed adopts a slow acceleration to avoid entering CA$_2$ too early. Also, the decoupled solution waits at $x \approx 50$m, right after entering CA$_5$, ranging [49.2,56.2]m, to stay outside of CA$_6$ before the first obstacle clears it. We remark that the vehicle does not need to come to a full stop to avoid entering CA$_6$ while the first obstacle still possesses it. Sporadic spikes in acceleration plots are induced by time discretization during motion generation.

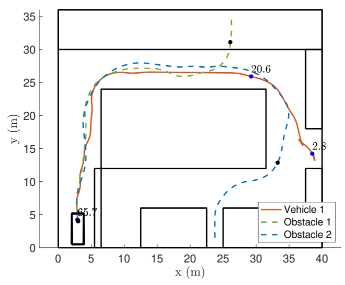


Fig. 10. Case 2: paths and CAs. Red solid: vehicle path; green dash: the first obstacle path; blue dash: the second obstacle path. In the regular algorithm, the vehicle has a large overlap CA$_1$ with the first obstacle, ranging [20.6, 65.7]m, and another CA$_2$ with the second obstacle, ranging [2.8,65.7]m.

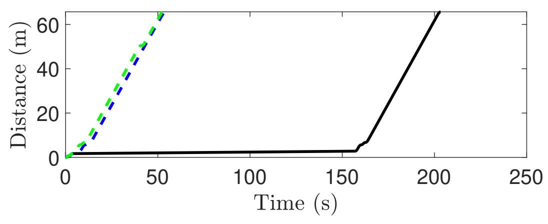


Fig. 11. Case 2: vehicle position trajectories.

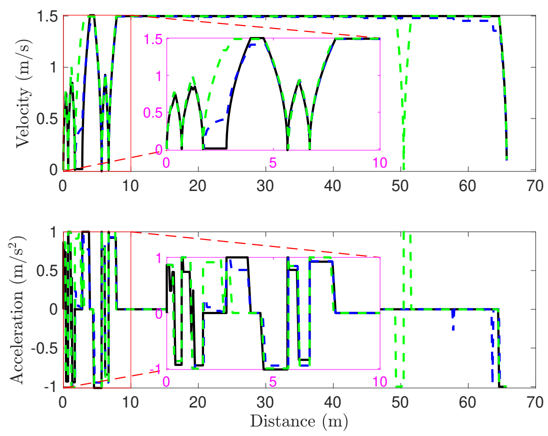


Fig. 12. Case 2: vehicle velocity & acceleration profiles along its path.

VI. Conclusions

This work presented a hierarchical planning method to fulfill fast planning for automated parking in dynamic environments. BIAGT, a variant of A*, was developed to solve path planning in static environments and is adapted for efficient, safe motion planning in dynamic environments by adopting the scheduling framework [46]. Major differences between BIAGT and A* are the construction of estimated cost-to-go and the newly added mode selection step. Analysis showed that BIAGT could lead to a smaller tree and lighter computation load, albeit losing the optimality guarantee. This is because the mode selection takes effect only if the cost-to-go is overestimated. A basic implementation of the scheduling framework [46] for parking scenarios could lead to unsatisfactory throughput and computation complexity, where the low throughput is caused by overlapping CAs, and computation complexity corresponds to the number of CAs. Decomposed and decoupled algorithms were developed to address these two limitations. The simulation confirmed the effectiveness of the proposed architecture and algorithms.

References

[1] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-93-20, 1993.

[2] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot Res.*, vol. 29, no. 5, pp. 485–501, 2010.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[4] J. T. Betts, "Survey of numerical methods for trajectory optimization," *J. Guid. Control Dynam.*, vol. 21, no. 2, pp. 193–207, Mar.-Apr. 1998.

[5] G. Elnagar, M. A. Kazemi, and M. Razzaghi, "The pseudospectral legendre method for discretizing optimal control problems," *IEEE Trans. Automat. Control*, vol. 40, no. 10, pp. 1793–1796, Oct. 1995.

[6] Y. Zhao, Y. Wang, M.-C. Zhou, and J. Wu, "Energy-optimal collision-free motion planning for multiaxis motion systems: An alternating quadratic programming approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 1, pp. 327–338, Jan. 2019.

[7] A. Richards, T. Schouwenaars, J. P. How, and E. Feron, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *J. Guid. Control Dynam.*, vol. 25, no. 4, pp. 755–763, July-August 2002.

[8] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. 2012 ICRA*, 2012, pp. 477 – 483.

[9] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot Res.*, vol. 5, no. 1, pp. 417–431, 1986.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[11] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[13] N. A. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *Proc. 2007 ICRA*, 2007, pp. 1617–1624.

[14] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. 2006 IROS*, 2006, pp. 5369–5375.

[15] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. 2006 ICRA*, 2006, pp. 2366–2371.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[17] M. Likhachev, G. Gordon, and S. Thrun, *ARA*: Anytime A* with probable bounds on sub-optimality*. MIT Press, 2003, ch. Advances in Neural Information Processing Systems.

[18] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[19] S. Koenig and M. Likhachev, "D* lite," *AAAI*, vol. 15, 2002.

[20] Y. Wang, "Improved A-search guided tree construction for kinodynamic planning," in *Proc. 2019 ICRA*, 2019, pp. 5530–5536.

[21] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[22] B. R. Donald and P. Xavier, "Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds," *Algorithmica*, vol. 14, pp. 443–479, 1995.

[23] H. M. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA: MIT Press, 2005.

[24] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An anytime, replanning algorithm," in *Proc. 2005 ICAPS*, 2005, pp. 262–271.

[25] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[26] K. Fujimura, "Time-minimum routes in time-dependent networks," *IEEE Trans. Robot. Automat.*, vol. 11, no. 3, pp. 343–351, 1995.

[27] T. Fraichard, "Trajectory planning in a dynamic workspace: A state-time space approach," *Advanced Robotics*, vol. 13, no. 1, pp. 75–94, 1998.

[28] J. P. van den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 885–897, 2005.

[29] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *Proc. 2011 ICRA*, 2011, pp. 5628–5635.

[30] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. J. Robot Res.*, vol. 21, no. 12, pp. 999–1030, 2002.

[31] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *Proc. 2004 ICRA*, 2004, pp. 4399–4404.

[32] L. Jaillet and T. Siméon, "Motion planning using dynamic roadmaps," in *Proc. 2004 IROS*, 2004, pp. 1606–1611.

[33] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proc. 2005 IROS*, 2005, pp. 2210–2215.

[34] T. Mercy, R. V. Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Trans. Contr. Syst. Technol.*, vol. 26, no. 6, pp. 2182–2189, Nov. 2018.

[35] I. E. Paromtchik and C. Laugier, "Motion generation and control for parking an autonomous vehicle," in *Proc. 1996 ICRA*, Apr. 1996, pp. 3117–3122.

[36] H. Vorobieva, S. Glaser, N. Minoiu-Enache, and S. Mammar, "Automatic parallel parking in tiny spots: path planning and control," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 1, pp. 396–410, 2015.

[37] K. Kondak and G. Hommel, "Computation of time optimal movements for autonomous parking of non-holonomic mobile platforms," in *Proc. 2001 ICRA*, Seoul, Korea, May. 2001, pp. 2698–2702.

[38] G. Lini, A. Piazzi, and L. Consolini, "Multi-optimization of η^3 for autonomous parking," in *Proc. 50th CDC*, 2011, pp. 6367–6372.

[39] B. Li and Z. Shao, "A unified motion lanning method for parking an autonomous vehicle in the presence of irregularly placed obstacles," *Knowledge-Based Systems*, vol. 86, pp. 11–20, 2015.

[40] B. Li, Y. Zhang, and Z. Shao, "Spatio-temporal decomposition: a knowledge-based initialization strategy for parallel parking motion optimization," *Knowledge-Based Systems*, vol. 107, pp. 179–196, Sept. 2016.

[41] L. Han, Q. H. Do, and S. Mita, "Unified path planner for parking an autonomous vehicle based on RRT," in *Proc. 2011 ICRA*, 2011, pp. 5622–5627.

[42] Y. Wang, D. K. Jha, and Y. Akemi, "A two-stage RRT path planner for automated parking," in *Proc. of IEEE Conf. on Automation Science and Engineering*, 2017, pp. 496–502.

[43] C. Chen, M. Rickert, and A. Knoll, "Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering," in *2015 IEEE Intelligent Vehicles Symposium*, Jun. 2015, pp. 1148–1153.

[44] S. Dai and Y. Wang, "Long-horizon motion planning for autonomous vehicle parking incorporating incomplete map information," in *Proc. 2021 ICRA*, 2021, pp. 8135–8142.

[45] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot Res.*, vol. 5, no. 3, pp. 72–89, 1986.

[46] H. Ahn and D. Del Vecchio, "Safety verification and control for collision avoidance at road intersections," *IEEE Trans. Automat. Control*, vol. 63, no. 3, pp. 630–642, Mar. 2018.

[47] E. Glassman and R. Tedrake, “A quadratic regulator-based heuristic for rapidly exploring state space,” in *Proc. 2010 ICRA*, 2010, pp. 5021–5028.

[48] M. Likhachev and D. Ferguson, “Planning long dynamically feasible maneuvers for autonomous vehicles,” *Int. J. Robot Res.*, vol. 28, no. 8, pp. 933–945, 2009.

[49] P. Fiorini and Z. Shiller, “Motion planning in dynamic environment using velocity obstacles,” *Int. J. Robot Res.*, vol. 17, no. 7, pp. 760–772, 1998.

[50] B. Müller, J. Deutscher, and S. Grodde, “Continuous curvature trajectory design and feedforward control for parking,” *IEEE Trans. Contr. Syst. Technol.*, vol. 15, no. 3, pp. 541–553, 2007.

[51] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, *Guidelines in nonholonomic motion planning for mobile robots*. Springer, 1998.

[52] J. A. Reeds and L. A. Shepp, “Optimal paths for a car that goes both forwards and backwards,” *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[53] K. G. Shin and N. D. Mckay, “Minimum-time control of robotic manipulators with geometric path constraints,” *IEEE Trans. Automat. Control*, vol. AC-30, no. 6, pp. 531–541, Jun. 1985.

[54] Y. Wang, K. Ueda, and S. A. Bortoff, “A Hamiltonian approach to compute an energy efficient trajectory for a servomotor system,” *Automatica*, vol. 49, no. 12, pp. 3550–3561, Dec. 2013.