Spatio-Temporal Lattice Planning Using Optimal Motion Primitives

Alexander Botros and Stephen L. Smith

***Abstract*—Lattice-based planning techniques simplify the motion planning problem for autonomous vehicles by limiting available motions to a pre-computed set of *primitives*. These primitives are combined online to generate complex maneuvers. A set of motion primitives t-span a lattice if, given a real number $t \geq 1$, any configuration in the lattice can be reached via a sequence of motion primitives whose cost is no more than a factor of t from optimal. Computing a minimal t-spanning set balances a trade-off between computed motion quality and motion planning performance. In this work, we formulate this problem for an arbitrary lattice as a mixed integer linear program. We also propose an A*-based algorithm to solve the motion planning problem using these primitives and an algorithm that removes the excessive oscillations from planned motions – a common problem in lattice-based planning. Our method is validated for autonomous driving in both parking lot and highway scenarios.**

Fig. 1: Motion planning using t-spanning $G3$ motion primitives. Magenta: primitives used, Red: final motion, Cyan: car footprint.

I. Introduction

Broadly, the motion planning problem is to search the configuration space of a system for a collision free path between a given start and goal [1] that optimizes desirable properties like travel time or comfort [2]. This path or *motion* can be used as a reference for a tracking controller [3] for a fixed amount of time before a new motion is planned.

In general, the motion planning problem—the focus of this work—is intractable [4] owing in part to potentially complex vehicle kinodynamic constraints that impede the calculation of two point boundary value (TPBV) problem solutions. Thus, simplifying assumptions are made. The variety in possible simplifications has given rise to several planning techniques. These typically fall into one of four categories [5]: sampling based planners, interpolating curve planners, numerical optimization approaches, and graph search based planners.

Lattice-based motion planning is an example of a graph search planner [5] and is one of the most common approaches to solving the motion planning problem [5], [6]. It works by discretizing the vehicle's configuration space into a countable set (or *lattice*) of regularly repeating configurations. Kinodynamically feasible motions between lattice configurations (or *vertices*) are pre-computed and a subset of these motions called a *control set* is selected [6]–[9]. Elements of this control set (called *motion primitives*) can be concatenated in real time to form complex maneuvers as in Figure 1, where motion primitives (magenta) are concatenated end-to-end to produce a final motion (red).

Lattice-based planners simplify the motion planning problem by limiting the set of available motions instead of simplifying their kinodynamics – guaranteeing feasibility. Appropriately selected control sets can yield theoretical guarantees on sub-optimality with respect to free-space optimal while keeping the size of the search graph small [10]. These guarantees do not require the fineness of the lattice grid structure to approach infinity, which is typically required in sampling-based techniques. Lattice-based motion planning is particularly attractive in scenarios with complex kinodynamic constraints since computationally expensive TPBV problems between lattice vertices are solved offline.

Though the process of selecting an appropriate resolution for the lattice discretization is inarguably important, this work focuses primarily on the traversal of the lattice once it has been created. Further, we do not propose a method for solving TPBV problems between lattice vertices given kinematic constraints, preferring instead to keep the methods proposed herein general and applicable to any motion constraints. Despite its many advantages, lattice-based motion planning is not without its critiques. This work presents novel solutions for three such critiques:

1) Choice of Control Set: It is observed in [11], [12] that the *number* of primitives in a control set favorably affects the quality of the resulting paths but adversely affects the run-time of an online search. The authors of [11] introduce the notion of a t-spanning control set: a control set that guarantees the existence of compound motions – motions formed by concatenating motion primitives – whose costs are no more than a factor of t from optimal. They propose to find the smallest set that, for a given value $t \geq 1$, will t-span a lattice thus optimizing a trade-off between path quality and performance. Computing such a control set is called the Minimum t-Spanning Control Set (MTSCS) problem which is known to be NP-hard [12]. Hitherto, the only known exact solver for the MTSCS problem involves a brute-force approach where each potential control set is considered [11].

2) Continuity in All States & On-lattice Propagation: Given a kinodynamically feasible motion originating from an initial configuration p_s, a new curve can be obtained from a configuration p'_s by rotating and translating the original curve

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC)

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, 200 University Ave W, Waterloo ON, Canada, N2L 3G1 (alexander.botros@uwaterloo.ca, stephen.smith@uwaterloo.ca)

if p_s and p'_s differ only in position and orientation. However, this is no longer true if p_s and p'_s differ in higher order states like curvature. For this reason, position and orientation are called variant states while higher order states are invariant [11]. When motion primitives are concatenated, invariant states may experience a discontinuous jump if all primitives are computed relative to a single starting vertex [11]. Further, for lattices with non-cardinal headings, using a single starting vertex to define all motion primitives may result in off-lattice endpoints during concatenations, requiring rounding and resulting in sub-optimal motions [13], [14]. A solution approach to this critique that we employ is to compute a control set for a lattice with several starting vertices [6], [11].

3) Excessive Oscillations: Because the set of available motions is restricted in lattice-based motion planning, compound motions comprised of several primitives may possess excessive oscillations [9] resulting in motions which are kinodynamically feasible but zig-zag unnecessarily.

A. Contributions

The solutions proposed in this paper to the challenges above are summarized here.

- *Choice of control set:* We propose a mixed integer linear programming (MILP) encoding of the MTSCS problem for state lattices with several starting vertices. This represents the first known non-brute force approach to solving the MTSCS problem. Though this formulation does not scale for large lattices, we observe that control sets need only be computed once, offline.
- *Continuity in All States:* The solution to the proposed MILP is the smallest control set that generates motions that are continuous in all states with bounded t-factor sub-optimality.
- *Reduction of Excessive Oscillations:* We provide a novel algorithm that eliminates redundant vertices along motions computed using a lattice. This algorithm is based on shortest paths in directed acyclic graphs and eliminates excessive oscillations. It runs in time quadratic in the number of motion primitives along the input motion.
- *Lattice Search Algorithm:* We present an A*-based algorithm to compute feasible motions for difficult maneuvers in both parking lot and highway settings. The algorithm accommodates off-lattice start and goal configurations up to a specified tolerance.

In our preliminary work [12], we introduced an MILP formulation for the MTSCS problem for lattices with a single starting vertex and proved the NP-completeness of this sub-problem. The novel contributions of this paper over our preliminary work include a MILP formulation of the MTSCS problem for a general lattice featuring several starting vertices, an A*-based search algorithm for use in lattice-based motion planning, and a post-processing smoothing algorithm to remove excessive oscillations from motions planned using a lattice.

B. Related Work

In this section, we review the four categories of planning techniques from [5]: sampling based, interpolating curve, numerical optimization, and graph search based (in particular lattice-based motion planning). We will outline key advantages to lattice-based motion planning as well as some of the deficiencies in the area that have motivated our contributions.

Sampling based planners work by randomly sampling configurations in a configuration space and checking connectivity to previous samples. Common examples include Asymptotically Optimal Rapidly-exploring Random Trees (RRT*) [15], and Probabilistic Roadmap (PRM) [16]. These methods use a local planner to expand or re-wire a tree to include new samples. Therefore, many TPBV problems must be solved online necessitating the use of simplified kinodynamics. Kinodynamic feasibility may not be guaranteed [5]. Sampling-based planners, while not complete, can be *asymptotically complete* with a convergence rate that worsens as the dimensionality of the problem is increased [17]. To improve the performance of RRT* in higher dimensions, the authors of [18] pre-compute a set of reachable configurations from which random samples are drawn. The pre-computed set is a lattice in which motions are computed by random sampling. Though appropriate for autonomous driving in unstructured environments like parking lots, sampling-based techniques are not widely used in highly structured environments like highways [19]. This paper proposes a planner that is appropriate in both settings.

Techniques using the interpolating curve approach include fitting Bezier curves, Clothoid curves [20], or polynomial splines [21] to a sequence of way-points. A typical criticism of clothoid-based interpolation techniques is the time required to solve TPBV problems involving Fresnel integrals [5], [22]. While TPBV problems are typically easily solved in the case of polynomial spline and Bezier curve interpolation, it is often difficult to impose constraints like bounded curvature on these curves owing to their low malleability [5], [22]. In [23], the authors develop a sampling-based planner that uses a control-affine dynamic model to facilitate solving TPBV problems. These motions are then smoothed via numeric optimization in [24]. Here, motions with non-affine non-holonomic constraints and piece-wise linear velocity profiles are developed. The authors of [24] illustrate the benefits of computing way-points using a system of similar complexity to the desired final motion. However, by the very nature of the simplification, it is possible for infeasible way-point paths to be developed. A further common critique of both interpolating curve and numerical optimization approaches is their reliance on global way-points [5], [22].

In contrast, lattice-based planners with motion primitives are capable of planning feasible trajectories [25] that do not rely on way-points [5]. In [13], [25], the authors use motion primitives to traverse a lattice with states given by position and heading. In both these works, compound motions comprised of several primitives are discontinuous in curvature – a known source of slip and discomfort [26]. To improve smoothness, the authors of [27] use the techniques of [25] to compute an initial trajectory that is then smoothed via numerical optimization. Similarly, in [28], paths from [25] are used as warm starts for an optimization-based collision avoidance algorithm for use is autonomous parking. The methods in [27], [28] require at least as much computation time as the methods proposed in [25]. In this work, we propose methods appropriate for both parking and highway driving scenarios that outperform the methods of [25] in both runtime and comfort metrics.

The choice of control set is of particular interest. Typically motion primitives are chosen to achieve certain objectives for the paths they generate. The authors of [29] maximize comfort by computing motion primitives that minimize the integral of the squared jerk over the motion. However, they limit their results to forward motion limiting applicability in parking lot scenarios. Natural behavior is the objective in [30] which introduces probabilistic motion primitives to achieve a blending of deterministic motion primitives. In this work, we illustrate a method of computing a control set of motion primitives given *any* objective. In [9] the authors consider the control set to be the entirety of the lattice which may necessitate coarser lattices for the sake of time efficiency. The notion that some motions in a lattice are sufficiently similar to others to not be included in a control set motivates the work in [31] which introduces a method of computing a control set is using the dispersion minimizing algorithm from [32].

In [7], on the other hand, the authors present the notion of using a MTSCS of motion primitives – a control set that balances a trade-off between its size and the optimality of resulting paths. These are similar to graph t-spanners first proposed in [33]. The process of computing such a set is elaborated in [34] where we compute motions between lattice vertices that minimize a user-specified cost function that is learned from demonstrations. The methods presented herein may be used in tandem with other work [34], [35] to provide an improved set of user-specific motion primitives.

In [11], the authors present a heuristic for the MTSCS problem which, though computationally efficient, does not have any known sub-optimality factor guarantees. Since a control set may be computed once, offline, and used over many motion planning problems, the time required to compute this control set is of less importance than its size.

II. Lattice Planning & Problem Statement

We begin with a review of lattice-based motion planning and outline the problems with the approach that are addressed in this paper.

Let $\mathcal{X}$ denote the configuration space of a vehicle. That is, $\mathcal{X}$ is a set of tuples — called *configurations* — whose entries are the *states* of the vehicle. A *motion* from an initial configuration $p_s \in \mathcal{X}$ to a goal $p_g \in \mathcal{X}$ is a path in $\mathcal{X}$ beginning at p_s and terminating at p_g. Let $\mathcal{M}$ denote the set of all kinodynamically feasible motions in $\mathcal{X}$ – that is, the motions that adhere to a known kinodynamic model. We assume that each motion in $\mathcal{M}$ can be evaluated using a known cost function $c : \mathcal{M} \to \mathbb{R}_{\geq 0}$. Though c is left general, we do assume that it obeys the triangle inequality. The motion planning problem is as follows.

Problem II.1 (Motion planning problem (MPP))**.** Given $\mathcal{X}$, a set of obstacles $\mathcal{X}_{\text{obs}}$, start and goal configurations $p_s, p_g \in \mathcal{X}\backslash\mathcal{X}_{\text{obs}}$, and a cost function c, compute a motion from p_s to p_g that is feasible and optimal:

1) **Feasible motions:** Each configuration in the motion is in $\mathcal{X}\backslash\mathcal{X}_{\text{obs}}$, and the motion is kinodynamically feasible.
2) **Optimal motions:** The motion minimizes c over all feasible motions from p_s to p_g.

Lattice-based planning approximates a solution to this problem by regularly discretizing $\mathcal{X}$ using a *lattice*, $L \subseteq \mathcal{X}$. Let

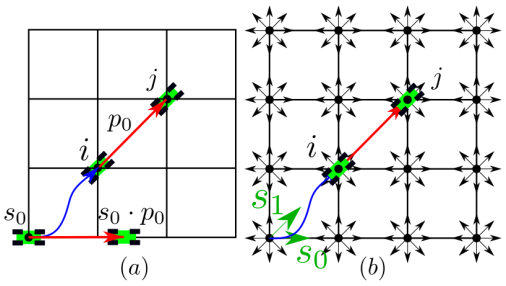

Fig. 2: (a) Config. space, lattice and start from (1). (b) Lattice and $\mathcal{X}$ as in (a) with start set $\mathcal{S} = [s_0 = (0, 0, 0), s_1 = (0, 0, \pi/4)]$. Blue motion from s_0 to $i = (1, 1, \pi/4)$, red from i to $j = (2, 2, \pi/4)$.

i, j be vertices in L, and let p be a motion that solves Problem II.1 for $p_s = i, p_g = j$, written $i \cdot p = j$. The key observation in lattice-based motion planning is that p may be applied to other vertices in L potentially resulting in another vertex in L. This motivates the use of a starting vertex $s \in L$, usually at the origin, that represents a *prototypical* lattice vertex. Offline, a feasible optimal motion is computed from s to all other vertices $i \in L - \{s\}$ in the absence of obstacles. A subset E –called a *control set* – of these motions is selected and used as an action set during an online search. The motions available at any iteration of the online search are isometric to those available at s (called *motion primitives*). In this work, we address three elements of lattice-based motion planning:

a) ***Continuity In All States****:* Care must be taken to ensure that states vary continuously over concatenations of motions. For example, suppose velocity is a state, starting vertex s is located at the origin (*i.e.,* the velocity at s is 0) and p is a motion from s to $i \in L$. If i does not have 0 velocity, then no motion originating at s can be applied to i without incurring a discontinuity in velocity. To ensure state continuity, we employ a *multi-start* lattice described in further detail in Section III.

b) ***Control Set Selection****:* As discussed in the Introduction, the choice of control set influences both the performance of an online search of the resulting graph as well as the quality of the motions computed. In Section IV, we formulate the problem of selecting a control set as a MTSCS problem in the context of a multi-start lattice. A mixed integer linear encoding of the problem is then proposed in Section V-A.

c) ***Graph Traversal****:* Motion primitives induce a search graph whose vertices are lattice vertices and whose edges are motions between vertices. In Section V-B, we present a variant of the A* algorithm for multi-start lattices. Further, in Section V-C, we propose a smoothing algorithm to remove excessive oscillations which often arise in lattice-based motion planning.

III. Multi-Start Lattices

There are examples for which a single start is insufficient to capture the full variety of motions if motions are constrained to be continuous in all states. For example, let

$$\begin{aligned} \mathcal{X} =& \mathbb{R}^2 \times [0, 2\pi], \; L = \mathbb{Z}^2 \times \{i\pi/4, i = 0, \ldots, 7\}, \\ s_0 =& (0, 0, 0), \end{aligned} \tag{1}$$

denote configurations space, lattice, and starting vertex, respectively – as in Figure 2 (a). If $i = (1, 1, \pi/4) \in L$ and

$j = (2, 2, \pi/4) \in L$, then the motion p_0 from i to j is such that $s_0 \cdot p_0 = (\sqrt{2}, 0, 0) \notin L$ if states x, y, θ vary continuously (as illustrated in Figure 2 (a)). Therefore, p_0 is not an action available at s_0 implying that the simple diagonal motion p_0 will not appear in any action set E.

We propose a solution to this problem by way of a multi-start lattice. Given a set of starts $\mathcal{S} \subset L$, vertices $i, j \in L$ and motion p from i to j, we say that $i \cdot p$ is a *valid concatenation*, and write $i \oplus p$ if there exists a start $s \in \mathcal{S}$ such that $s \cdot p \in L$. In the example in Figure 2 (b), let $\mathcal{S} = [s_0 = (0, 0, 0), s_1 = (0, 0, \pi/4)]$, then the motion p_0 from i to j is such that $s_1 \cdot p_0 = i \in L$ implying $i \oplus p_0$ is a valid concatenation.

An ideal set of starts $\mathcal{S}$ would have the property that for every pair of vertices $i, j \in L$ with motion p from i to j, the concatenation $i \cdot p$ is valid. Obtaining such a set is simple for a class of configuration spaces described here. For motion planning for autonomous cars and car-like robots, we typically use a configuration space of the form

$$\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi) \times U_1 \cdots \times U_N, \tag{2}$$

where $(x, y) \in \mathbb{R}^2$ represents the planar location of the vehicle, $\theta \in [0, 2\pi)$ the heading, and $u_i, i = 1 \dots N$ represent higher order states in a state space $u_i \in U_i \subset \mathbb{R}$. These states u_i may include curvature, curvature rate, velocity, acceleration, jerk, additional angles, etc. We make the assumption that each state U_i is bounded by physical constraints of the vehicle, passenger comfort, or speed limits. This allows us to write $U_i = [U_i^0, U_i^1] \subset \mathbb{R}$ for some upper and lower limits $U_i^0, U_i^1 \in \mathbb{R}$. For $\mathcal{X}$ in (2), we construct a lattice $L \subseteq \mathcal{X}$ by discretizing each state separately. In particular, for $\alpha, \beta \in \mathbb{R}_{>0}$, and $n_0, n_1, n_2, m_i \in \mathbb{N}$ for $i = 1 \dots N$, we let

$$\begin{aligned} L =& \{i\alpha, i = -n_0 \dots n_0\} \times \{i\beta, i = -n_1 \dots n_1\} \\ &\times \{\pi i / 2^{n_2 - 1}, i = 0 \dots 2^{n_2} - 1\} \\ &\times \prod_{i=1}^{N} \{U_i^0 + j(U_i^1 - U_i^0)/m_i\}_{j=0}^{m_i}. \end{aligned} \tag{3}$$

This lattice samples $2n_0 + 1$, and $2n_1 + 1$ values of the x, y coordinates, respectively, with spacing α, β between samples, respectively. It partitions heading values $[0, 2\pi)$ and $U_i \subset [U_i^0, U_i^1] \subset \mathbb{R}, i = 1 \dots, N$ into 2^{n_2} and $m_i + 1$ evenly spaced samples, respectively. For $\mathcal{X}, L$ from (2), (3), let

$$\begin{aligned} \mathcal{S} = \Big\{ & (0, 0, \theta, u_1, \dots u_N) \ : \\ & \theta \in \{j\pi / 2^{n_2 - 1}, j = 0, \dots, 2^{n_2 - 2} - 1\}, \\ & u_i \in \{U_i^0 + j(U_i^1 - U_i^0)/m_i, j = 0 \dots m_i\} \Big\}. \end{aligned} \tag{4}$$

This starting set has the property that for any vertices $i, j \in L$, the motion p from i to j is such that $\exists s \in \mathcal{S}$ with $s \oplus p \in L$.

IV. Selecting a Control Set

In this section, we describe our proposed approach for selecting a control set and formulate the Minimum t-Spanning Control Set (MTSCS) problem for multi-start lattices. We assume that for all $i, j \in \mathcal{X}$, there is a motion p solving Problem II.1 from i to j in the absence of obstacles and that the cost of this motion, $c(p)$, is known, non-negative, and is equal to 0 if and only if $i = j$. Further, we assume that c obeys the triangle inequality – i.e., if p_1 is a motion from $i \in \mathcal{X}$ to $j \in \mathcal{X}$, and p_2 is a motion from i to j by way of $r \in \mathcal{X}$, then $c(p_1) \leq c(p_2)$. This cost is left general, and may represent travel time, comfort, etc.

Given $(\mathcal{X}, L, \mathcal{S}, c)$ – configurations space, lattice, starting set and cost of motions, respectively, the idea behind multi-start control set motion planning is the following. For each $s \in \mathcal{S}$, pre-compute a set $\mathcal{B}_s$ of cost-minimizing feasible motions from s to vertices $i \in L - \mathcal{S}$, and let $\mathcal{B} = \bigcup_{s \in \mathcal{S}} \mathcal{B}_s$. We select a subset $E \subseteq \mathcal{B}$ to use as an action set during an online search in the presence of obstacles. We offer the following definition:

Definition IV.1 (Relative start)**.** For vertex $i \in L$, the *relative start* of i is the vertex $R(i) \in \mathcal{S}$ such that for each $j \in L$, if p_j is the motion from $R(i)$ to j then $i \oplus p_j$ is valid.

To construct a set $E \subseteq \mathcal{B}$, we choose a set E_s to be the action set for lattice vertices with relative start s, and let $E = \bigcup_{s \in \mathcal{S}} E_s$. If L and $\mathcal{S}$ are from (3), (4), respectively, then the relative start of any configuration $i = (x, y, \theta, u_1, \dots, u_N) \in L$ is $R(i) = (0, 0, \theta', u_1, \dots, u_N)$ where

$$\theta' = \frac{\pi}{2^{n_2 - 1}} \left(\frac{2^{n_2 - 1}\theta}{\pi} \mod 4 \right).$$

The set of available motions during an online search — i.e., the action set — at a configuration $i \in L$ is given by $E_{R(i)}$, and the cost of each motion $p \in E_{R(i)}$ is given by $c(p)$.

For any subset $E \subseteq \mathcal{B}$, we denote by $\bar{E}$ the set of all tuples $(i, j) \in L^2$ such that if p is the optimal feasible motion from i to j, then $p \in E_{R(i)}$:

$$\bar{E} = \{(i, j) \in L^2 : (i \oplus p = j) \Rightarrow p \in E_{R(i)}\}. \tag{5}$$

Critically, solving the motion planning problem between lattice vertices using a control set E is equivalent to computing a shortest path in the weighted directed graph $G = (L, \bar{E}_{\text{CF}}, c)$ where $\bar{E}_{\text{CF}} \subseteq \bar{E}$ is the set of all collision-free edges in $\bar{E}$, and for all $(i, j) \in \bar{E}_{\text{CF}}$, $c((i, j))$ is the cost of the optimal feasible motion from i to j. This graphical representation of motion planning motivates the following definitions given the tuple $(\mathcal{X}, L, \mathcal{S}, c, E)$:

Definition IV.2 (Path using E)**.** A *path using* E from $s \in \mathcal{S}$ to $j \in L$, denoted $\pi^E(s, j)$ is the cost-minimizing path from s to j (ties broken arbitrarily) in the weighted, directed graph $G_{\text{Free}} = (L, \bar{E}, c)$ in the absence of obstacles .

Definition IV.3 (Motion cost using E)**.** The cost of a motion *using* E from $s \in \mathcal{S}$ to $j \in L$, denoted $d^E(s, j)$ is the cost of the path $\pi^E(s, j)$ in $G_{\text{Free}} = (L, \bar{E}, c)$.

Let $i \in L - \mathcal{S}, s \in \mathcal{S}$ and p a cost minimizing motion from s to i. If $E = \mathcal{B}$, then $d^E(s, j) = d^{\mathcal{B}}(s, j) = c(p)$ implying that using $\mathcal{B}$ as a control set will result in minimal cost paths. However, if $\mathcal{B}$ is large, the branching factor at a vertex $i \in L$ during an online search may be prohibitive. We therefore wish to limit the size of $|E_{R(i)}|$ while keeping $d^E(s, j)$ close to $d^{\mathcal{B}}(s, j)$ for all $j \in L$. This motivates the following definition:

Definition IV.4 (t-Error)**.** Given the tuple $(\mathcal{X}, L, \mathcal{S}, c, E)$, the *$t$-error* of E is defined as

$$t\text{Er}(E) = \max_{\substack{s \in \mathcal{S} \\ j \in L - \mathcal{S}}} \frac{d^E(s, j)}{d^{\mathcal{B}}(s, j)}.$$

That is, the t-error of a control set E is the worst-case ratio of the distance using E from a start s to a vertex j to the cost of the optimal path from s to j over all $s \in \mathcal{S}, j \in L$. The t-error can be used to evaluate the quality of a control set E:

Definition IV.5 (t-Spanning Set)**.** Given the tuple $(\mathcal{X}, L, \mathcal{S}, c, E)$, and a real number $t \geq 1$, we say that a set E is a *t-spanner* of L (or *t-spans* L), if $t\mathit{Er}(E) \leq t$.

Our objective is to compute a control set E that optimizes a trade-off between branching factor and motion quality. This is formulated in the following problem:

Problem IV.6 (Minimum t-spanning Control Set Problem)**.**
Input: A tuple $(\mathcal{X}, L, \mathcal{S}, c)$, and a real number $t \geq 1$.
Output: A control set $E = \bigcup_{s\in\mathcal{S}} E_s$ that t-spans L where $\max_{s\in\mathcal{S}} |E_s|$ is minimized.

Using a solution E to Problem IV.6 as a control set has two beneficial properties. First, the number of collision-free neighbors of any vertex $i \in L$ in the graph G is at most $|E_{R(i)}|$ whose maximum value is minimized. Second, for every $i \in L - \mathcal{S}, s \in \mathcal{S}$, it must hold that $d^E(s,i) \leq t d^{\mathcal{B}}(s,i)$. Thus, Problem IV.6 represents a trade-off between branching factor and motion quality. As an alternative, the framework presented here can compute a t-spanning that minimizes $|E|$ instead of $\max_{s\in\mathcal{S}} |E_s|$ though this does not ensure a minimum branching factor at each vertex.

V. Proposed Methods

In this section we present a MILP formulation of Problem IV.6, and algorithms to compute and smooth motions using computed control sets.

A. Multi-Start MTSCS Problem: MILP Formulation

In [36, Theorem 5.2.1], we show that Problem IV.6 is NP-hard. Informally, there is no known polynomial time algorithm for solving any problem in this class (if one existed, then it would prove P = NP). Thus, problems in NP-hard are widely believed to be intractable beyond a certain problem size. Motivated by this, we pose the problem as a MILP, an NP-hard class of problems. Several powerful MILP solvers are available, allowing many problems to be solved to optimality. These solvers are also anytime, returning feasible solutions along with sub-optimality certificates when terminated early. Let $(\mathcal{X}, L, \mathcal{S}, c)$ denote configuration space, lattice, start set, and cost of vertex-to-vertex motions in L, respectively. For any motion $q \in \mathcal{B} = \bigcup_{s\in\mathcal{S}} \mathcal{B}_s$, let

$$S_q = \{(i,j) \ : \ i,j \in L, \ i \oplus q = j\}.$$

Thus S_q is the set of all pairs $(i,j) \in L^2$ such that q is a cost-minimizing feasible motion from i to j and $i \cdot q$ is valid. By definition of valid concatenations, there exists $s \in \mathcal{S}$ and $j' \in L$ such that q is a cost-minimizing feasible motion from s to j' (i.e., $s = R(i)$) implying that $(s, j') \in S_q$.

Let $E = \bigcup_{s\in\mathcal{S}} E_s$ be a solution to Problem IV.6. Let $G_{\text{Free}} = (L, \bar{E}, c)$ be the weighted directed graph with edges $\bar{E}$ given in (5). For each $s \in \mathcal{S}$ and each $(i,j) \in \bar{E}$, we make a copy $(i,j)^s$ of the edge. This allows us to treat edges differently depending on the starting vertex of the path to which they belong. For each $r \in L$, a path using E from s to r, $\pi^E(s,r)$, may be expressed as a sequence of edges $(i,j)^s$ where $(i,j) \in \bar{E}$. For each $s \in \mathcal{S}$ we construct a new graph

$$T^s = (L, E_T^s), \tag{6}$$

whose edges E_T^s are defined as follows: let

$$\mathcal{P}^s = \bigcup_{i\in L-\mathcal{S}} \pi^E(s,i).$$

Thus $\mathcal{P}^s$ is the set of all minimal cost paths from s to vertices $i \in L - \mathcal{S}$ in the graph G_{Free}. These paths are expressed as a sequence of edge copies $(i,j)^s$ where $(i,j) \in \bar{E}$. For each $i \in L - \mathcal{S}$, and for each $s \in \mathcal{S}$, if $\mathcal{P}^s$ contains two paths π_1^E, π_2^E from s to i, determine the last common vertex j in paths π_1^E, π_2^E, and delete the the copy of the edge in π_2^E whose endpoint is j from $\mathcal{P}^s$. The remaining edges are the set E_T^s. The graph T^s has a useful property defined here:

Definition V.1 (Arborescence)**.** From Theorem 2.5 of [37], a graph T with a vertex s is an arborescence rooted at s if every vertex in T is reachable from s, but deleting any edge in T destroys this property.

Intuitively, if T is an arborescence rooted at s, then for each vertex $j \neq s$ in T, there is a unique path in T from s to j. For each $s \in \mathcal{S}$, the graph T^s an arborescence rooted at s. This is shown in the following Lemma:

Lemma V.2 (Arborescence Lemma)**.** *Let $E = \bigcup_{s\in\mathcal{S}} E_s$ be a solution to Problem IV.6, and $G_{\text{Free}} = (L, \bar{E}, c)$. For each $s \in \mathcal{S}$, T^s (in (6)) is an arborescence rooted at s. Further, $\forall i \in L - \mathcal{S}$, $d^E(s,i)$ is the length of the path in T^s to i.*

Proof. Let $s \in \mathcal{S}$. To show that T^s is an arborescence rooted at s, observe that there is a path in T^s from s to all $i \in L$. Indeed, if E solves Problem IV.6, then E t-spans L and there must be at least one path, $\pi^E(s,i)$ using E from s to each $i \in L$ implying that $\pi^E(s,i) \in \mathcal{P}^s$. Since E_T^s deletes duplicate paths from $\mathcal{P}^s$, there must be precisely one path in T^s from s to i implying that T^s is an arborescence rooted at s. The cost of the path in T^s from s to i is defined as the cost of the path $\pi^E(s,i)$ which is $d^E(s,i)$. □

Lemma V.2 implies that E is a t-spanner of L if and only if $\forall s \in \mathcal{S}$ there is a corresponding arborescence T^s rooted at s whose vertices are L, whose edges $(i,j)^s$ are copies of members of S_q for a $q \in E$, and where the cost of the path from s to any $i \in L$ in T^s is no more than a factor of t from optimal. Indeed, the forward implication follows from Lemma V.2, while the converse holds by definition of a t-spanner. From this, we develop four criteria that represent necessary and sufficient conditions for E to t-span L:

Usable Edge Criteria: For any $q \in \mathcal{B}$, for any $s \in \mathcal{S}$, and for any $(i,j) \in S_q$, the copy $(i,j)^s$ may belong to a path in T^s from s to a vertex $r \in L$ if and only if $q \in E_{R(i)}$.

Cost Continuity Criteria: For any $s \in \mathcal{S}$, $q \in \mathcal{B}$, $(i,j) \in S_q$, and $(i,j)^s$ a copy of (i,j), if $(i,j)^s$ lies in the path in T^s to vertex j then $c(\pi^E(s,j)) = c(\pi^E(s,i)) + c(q)$. That is, the cost of the path from s to j in T^s is equal to the cost of the path from s to i plus the motion from i to j.

t-Spanning Criteria: For any vertex $j \in L - \mathcal{S}$, and $s \in \mathcal{S}$, the cost of the path in T^s from s to j can be no more than t times the cost of the direct motion from s to j.

Arborescence Criteria: The set T^s must be an arborescence for all $s \in \mathcal{S}$.

We now present an MILP encoding of these criteria. Let $|L| = n$ with all vertices enumerated $1, 2, ..., n$ with $s \in \mathcal{S}$ taking values $1, \dots, m$ for $m \leq n$. For any control set $E = \bigcup_{s\in\mathcal{S}} E_s$, define $m(n-m)$ decision variables:

$$y_q^s = \begin{cases} 1, \text{ if } q \in E_s \\ 0, \text{ otherwise.} \end{cases}, \; q = m+1, \dots, n$$

For each edge $(i,j) \in L^2$, and each $s \in \mathcal{S}$ let

$$x_{ij}^s = \begin{cases} 1 & \text{if } (i,j)^s \in T^s \\ 0 & \text{otherwise.} \end{cases}$$

That is, $x_{ij}^s = 1$ if $(i,j)^s$ (the copy of the edge (i,j) for start $s \in \mathcal{S}$) lies on a path from s to a vertex in the lattice. Let z_i^s denote the length of the path in the tree T^s to vertex i for any $i \in L$, c_{ij} the cost of the optimal feasible motion from i to j, and let $L' = L - \mathcal{S}$. The criteria above can be encoded as the following MILP:

$$\min_{K\in\mathbb{R}} \; K \tag{7a}$$
$$s.t. \; \forall s \in \mathcal{S}$$
$$\sum_{q\in L'} y_q^s \leq K, \tag{7b}$$
$$x_{ij}^{s'} - y_q^s \leq 0, \qquad \forall (i,j) \in S_q, \quad \forall q \in \mathcal{B}, \; \forall s' \in \mathcal{S}, \tag{7c}$$
$$z_i^s + c_{ij} - z_j^s \leq M_{ij}^s(1 - x_{ij}^s), \qquad \forall (i,j) \in L \times L' \tag{7d}$$
$$z_j^s \leq t c_{sj}, \qquad \forall j \in L' \tag{7e}$$
$$\sum_{i\in L} x_{ij}^s = 1, \qquad \forall j \in L' \tag{7f}$$
$$x_{ij}^s \in \{0,1\}, \qquad \forall (i,j) \in L \times L' \tag{7g}$$
$$y_q^s \in \{0,1\}, \qquad \forall q \in \mathcal{B}, \tag{7h}$$

where $M_{ij}^s = t c_{si} + c_{ij} - c_{sj}$. The objective function (7a) together with constraint (7b) minimizes $\max_{s\in\mathcal{S}} |E_s|$ as in Problem IV.6. The remainder of the constraints encode the four criteria guaranteeing that E is a t-spanning set of L:

Constraint (7c): Let q be the motion in $\mathcal{B}$ from $s \in \mathcal{S}$ to $j \in L - \mathcal{S}$. If $q \notin E_s$, then $y_q^s = 0$ by definition. Therefore, (7c) requires that $x_{ij}^{s'} = 0$ for all $(i,j) \in S_q, s' \in \mathcal{S}$. Alternatively, if $y_q^s = 1$, then $x_{ij}^{s'}$ is free to take values 1 or 0 for any $(i,j) \in S_q, s' \in \mathcal{S}$. Thus constraint (7c) encodes the Usable Edge Criteria.

Constraint (7d): Constraint (7d) takes a similar form to [38, Equation (3.7a)]. Note that $M_{ij}^s \geq 0$ for all $(i,j) \in \mathcal{B}, s \in \mathcal{S}$. Indeed, $\forall t \geq 1$, $M_{ij}^s \geq c_{si} + c_{ij} - c_{sj}$, and $c_{si} + c_{ij} \geq c_{sj}$ by the triangle inequality. Replacing M_{ij}^s in (7d) yields

$$z_i^s + c_{ij} - z_j^s \leq (t c_{si} + c_{ij} - c_{sj})(1 - x_{ij}). \tag{8}$$

If $x_{ij}^s = 1$, then $(i,j)^s$ is on the path in T^s to vertex j and (8) reduces to $z_j^s \geq z_i^s + c_{ij}$ which encodes the Cost Continuity Criteria. If, however, $x_{ij}^s = 0$, then (8) reduces to $z_i^s - z_j^s \leq t c_{si} - c_{sj}$ which holds trivially by constraint (7e) and by noting that $z_j^s \geq c_{sj}, \forall j \in L$ by the triangle inequality.

Constraint (7f): Constraint (7f) together with constraint (7d) yield the Arborescence Criteria. Indeed for all $s \in \mathcal{S}$, by Theorem 2.5 of [37], T^s is an arborescence rooted at s if every vertex in T^s other than s has exactly one incoming edge, and T^s contains no cycles. The constraint (7f) ensures that every vertex in L', which is the set of all vertices in T^s other than those in $\mathcal{S}$, have exactly one incoming edge, while constraint (7d) ensures that T^s has no cycles. Indeed, suppose that a cycle existed in T^s, and that this cycle contained vertex $i \in L'$. Suppose that this cycle is represented as

$$i \to j \to \cdots \to k \to i.$$

Recall that it is assumed that the cost of any motion between two different configurations in $\mathcal{X}$ is strictly positive. Therefore, (7d) implies that $z_i^s < z_j^s$ for any $(i,j) \in L^2$. Therefore,

$$z_i < z_j < \cdots < z_k < z_i,$$

which is a contradiction.

B. Motion Planning With a MTSCS

The previous section illustrates how to compute a control set $E = \bigcup_{s\in\mathcal{S}} E_s$ given the tuple $(\mathcal{X}, L, \mathcal{S}, c)$. We have also presented a description of how such a control set may be used during an online A*-style search of the graph $G = (L, \bar{E}_{\text{CF}}, c)$ from a start vertex $p_s \in L$ to a goal vertex $p_g \in L$. Here, the edge set $\bar{E}_{\text{CF}}$ is the set of pairs $(i,j) \in L^s$ such that there exists a collision-free motion $p \in E_{R(i)}$ from i to j where $R(i) \in \mathcal{S}$ is the relative start of i. The motion primitives in this control set could be used in any graph-search algorithm. In this section, we propose one such algorithm: an A* variant, *Primitive A* Connect* (PrAC) for path computation in G. The algorithm follows the standard A* algorithm closely with some variations described here:

Weighted fScore: In the standard A* algorithm, two costs are maintained for each vertex i: the *gScore*, $g(i)$ representing the current minimal cost to reach vertex i from the starting vertex p_s, and the *fScore*, $f(i) = g(i) + h(i)$ which is the sum of the gScore and an estimated cost to reach the goal vertex p_g by passing through i given heuristic h. We implement the weighted fScore from [39]: Given a value $\lambda \in [0,1]$, we let $a = 0.5\lambda, b = 1 - 0.5\lambda$, and we define a new cost function $f' = ag(i) + bh(i)$. If $\lambda = 1$, then both gScore and heuristic are weighted equally and standard A* is recovered. However, as λ approaches 0, more weight is placed on the heuristic which promotes *exploration* over *optimization*. While using a value $\lambda < 1$ eliminates optimality guarantees, it also empirically improves runtime performance. In practice, using $\lambda = 1$ works well for maneuvers where p_s and p_g are close together, like parallel parking, while small values of λ work well for longer maneuvers like traversing a parking lot.

Expanding Start and Goal Vertices: We employ the bi-directional algorithm from [40] with the addition of a direct motion. In detail, we expand vertices neighboring both start and goal vertices and attempt to connect these vertices on each iteration. In essence, we double the expansion routine at each iteration of A* once from p_s to p_g and once from p_g to p_s with reverse orientation. We maintain two trees, one rooted at the start vertex p_s, denoted T_s, the other at the goal, T_g whose leaves represent open sets O_s, O_g, respectively. We also maintain two sets containing the current best costs $g(p_s, i)$ to

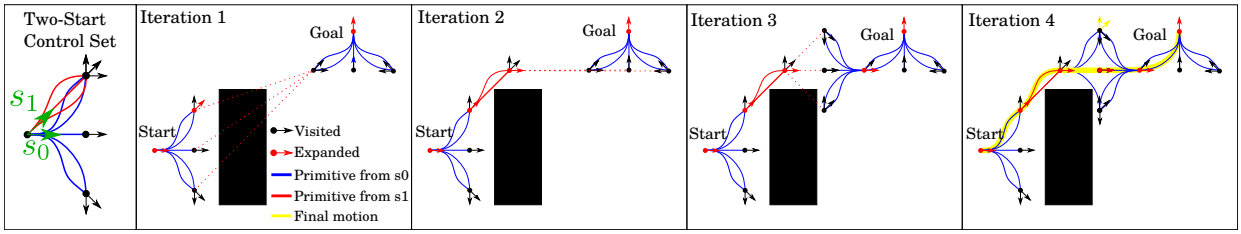

Fig. 3: Example motion planning using PrAC for a 2-start lattice.

get from p_s to each vertex $i \in T_s$, and from p_g to each vertex $j \in T_g$ (traversed in reverse), $g(p_g, j)$, respectively. Given an admissible heuristic h', we define a new heuristic h as

$$\begin{aligned} h(i) &= \min_{j \in O_g} h'(i,j) + g(p_g, j), \ \forall i \in O_s \\ h(j) &= \min_{i \in O_s} h'(j,i) + g(p_s, i), \ \forall j \in O_g \end{aligned} \tag{9}$$

On each iteration of the A* while loop, let $i \in O_s$ be the vertex that is to be expanded, let $j \in O_g$ be the vertex that solves (9) for i, and let p denote the pre-computed cost-minimizing feasible motion from i to j. We expand vertex i by applying available motions in $E_{R(i)} \cup \{p\}$ where $E_{R(i)}$ is the set of available motions at the relative start of i. The addition of p to the available motions improves the performance of the algorithm by allowing quick connections between T_s, T_g where possible. In the same iteration, we then swap T_s and T_g and perform the same steps but with all available motions in reverse. This is illustrated in Figure 3. Expanding start and goal vertices has proven especially useful during complex maneuvers like backing into a parking space.

Off-lattice Start and Goal Vertices: For a configuration $v \in \mathcal{X}$ and a lattice L given by (3), Round(v) denotes a function that returns the element of L found by rounding each state of v to the closest value of that state in L. For lattice L with start set S given by (4) and control set E obtained by solving the MILP in (7), it is likely that the start and goal configurations $p_s, p_g \in \mathcal{X}$ do not lie in L. This is particularly true in problems that necessitate frequent re-planning. While increasing the fidelity of the lattice or computing motions online between lattice vertices and p_s, p_g (e.g., [9]) can address this issue, both these methods adversely affect the performance of the planner. Instead we propose a method using lattices with graduated fidelity, a concept introduced in [8]. We compute a control set E_{off} of primitives from off-lattice configurations to lattice vertices. These can be traversed in reverse to bring lattice vertices to off-lattice configurations. The technique is summarized in Algorithm 1 which is executed offline. The algorithm takes as input L, E, and a vector of tolerances for each state $Tol = (x_{\text{tol}}, y_{\text{tol}}, \theta_{\text{tol}}, u_{1,\text{tol}}, \ldots, u_{N,\text{tol}})$. It first computes a set $Q \subset \mathcal{X}$ such that for every configuration in $v \in \mathcal{X}$ there exists a configuration $q \in Q$ that can be translated to $q' \in \mathcal{X}$ where each state of q' is within the accepted tolerance of v (Line 2). For each element of Q, we determine the lattice vertex $q' = \text{Round}(q)$, and the set of available actions at the relative start of q', $E_{R(q')}$. For each primitive in $E_{R(q')}$, we compute a motion from q to a lattice vertex close to the endpoint of p and store it in a set E_q (Lines 7-13). In Lines 9, 10 the vertex p is modified to p' that is no closer to q than p. This is to ensure that the motion added to E_q in Line 13 does not posses large loops not present the primitive p. These loops can arise if the start and end configurations of a motion are too close together. An illustrative example of this principle is given in Figure 4.

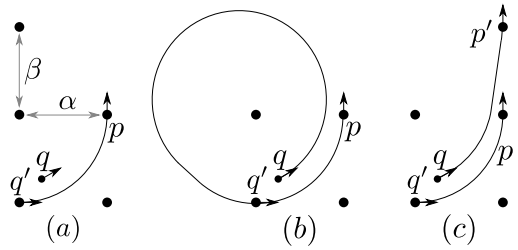

Fig. 4: Algorithm 1 Lines 9-13 Example. (a) Configuration $q \in \mathcal{X} - L$, lattice vertex $q' = \text{Round(q)}$, and primitive $p \in E_{R(q')}$. (b) Motion from q to end of p results in a loop as q, p are too close together. (c) Vertex p replaced with neighboring vertex p' and a motion from q to p' is computed.

Theorem V.3 (Completeness)**.** *If $\lambda = 1$ and $p_s, p_g \in L$, then PrAC returns the cost-minimizing path in $G = (L, \bar{E}_{CF}, c)$.*

Proof. Given the assumptions of the Theorem, we may view PrAC as standard A* where the expansion of T_g serves only to update the heuristic. That is, PrAC reduces to standard A* with a heuristic given by (9) for all $i \in O_s$. At each iteration of A*, h is improved by expanding T_g. By the completeness of A* for admissible heuristics and finite branching factor (which is the case here for finite control sets), it suffices to show that h is indeed admissible. At each iteration of A*, let i be the vertex in O_s that is to be expanded. Let π be the optimal path from i to p_g in G. Then π must pass through a vertex r in O_g. This result holds by the completeness of A* with initial configuration p_g with paths traversed in reverse. Then by the triangle inequality, $c(\pi) \geq c(\pi^{\bar{E}_{\text{CF}}}(i,r)) + g(p_g, r)$. Finally, because h' is an admissible heuristic, $c(\pi^{\bar{E}_{\text{CF}}}(i,r)) + g(p_g, r) \geq h'(i,r) + g(p_g, r) \geq \min_{j \in O_g} h'(i,j) + g(p_g, j) = h(i)$. Therefore, $h(i) \leq c(\pi)$ which concludes the proof. □

C. Motion Smoothing

Given the tuple $(\mathcal{X}, L, \mathcal{S}, c, E)$, of configuration space, lattice, cost function, and MTSCS, respectively, let $G = (L, \bar{E}_{\text{CF}}, c)$ be the weighted directed graph with edge set $\bar{E}_{\text{CF}}$ is the collision-free subset of $\bar{E}$ given in (5).

We now present a smoothing algorithm based on the *shortcut* approach that takes as input a path in G, here $\pi^E(p_s, p_g)$, between start and goal configurations p_s, p_g. This path is expressed as a sequence of edges in $\bar{E}_{\text{CF}}$. Thus,

Algorithm 1 Generate Off-Lattice Control Set

procedure OFFLATTICE(L, E, Tol)
  $E_{\text{off}} \leftarrow \emptyset$
  $[U_x^0, U_x^1], [U_y^0, U_y^1], [U_\theta^0, U_\theta^1] \leftarrow [0, \alpha], [0, \beta], [0, 2\pi - 2\theta_{\text{tol}}]$
  $N_p \leftarrow \lceil (U_p^1 - U_p^0)/2u_{p_{\text{tol}}} \rceil, p = x, y, \theta, 1, \dots N$
  $Q \leftarrow \prod_{p \in \{x,y,\theta,1,\dots,N\}} \{U_p^0 + {}^{(U_p^1 - U_p^0)j}/_{N_p}\}_{j=0}^{N_p}$
  for $q \in Q$ **do**
    $(x_q, y_q, \theta_q, u_{1,q}, \dots, u_{N,q}) \leftarrow q$
    $E_q \leftarrow \emptyset$
    $q' \leftarrow \text{Round}(q)$
    for $p \in E_{R(q')}$ **do**
      $(x_p, y_p, \theta_p, u_{1,p}, \dots, u_{N,p}) \leftarrow p$
      for $x \in \{x_p, x_p + \text{sign}(x_p - x_q)\alpha\}$ **do**
        for $y \in \{y_p, y_p + \text{sign}(y_p - y_q)\beta\}$ **do**
          $p' \leftarrow (x, y, \theta_p, u_{1,p}, \dots, u_{N,p})$
          Compute motion from q to p'
      Add motion of lowest cost over all p' to E_q
    Add E_q to E_{off}
  return E_{off}

$\pi^E(p_s, p_g) = \{(i_r, i_{r+1}), r = 1, \dots, m-1\}$ for some $m \in \mathbb{N}_{\geq 2}$ where $(i_r, i_{r+1}) \in \bar{E}_{\text{CF}}$ for all $r = 1, \dots, m-1$ and $i_1 = p_s, i_m = p_g$. Let C_π denote the set of all configurations along motions from i_r to i_{r+1} for all $(i_r, i_{r+1}) \in \pi^E(p_s, p_g)$. Algorithm 2 summarizes the proposed approach.

Algorithm 2 Smoothing Lattice Motion

1: **procedure** DAGSMOOTH($\pi^E(p_s, p_g), C_\pi, \mathcal{X}_{\text{obs}}, c, n, \Psi$)
2: $V_1 \leftarrow$SampleRandom(n, C_π)
3: $V \leftarrow \{i_r\}_{r=1}^m \cup V_1$ in the order that they appear in C_π
4: $\{i_r\}_{r=1}^{m+n} \leftarrow V$
5: **for** $i \in V$ **do**
6: dist$(i) = \infty$
7: dist$(i_1) = 0$
8: Pred(i_1) =None
9: **for** u from 1 to $n + m - 1$ **do**
10: **for** v from $u + 1$ to $m + n$ **do**
11: $p_1 =$ motion from i_u to i_v
12: $p_2 = p_1$ motion from i_v to i_u
13: $\underline{c} = \min\big(c(p_1), \Psi(c(p_2))\big)$,
14: $\underline{p} = \arg\min\big(c(p_1), \Psi(c(p_2))\big)$
15: $\overline{c} = \max\big(c(p_1), \Psi(c(p_2))\big)$,
16: $\overline{p} = \arg\max\big(c(p_1), \Psi(c(p_2))\big)$
17: **if** dist$(i_u) + \underline{c} \leq$ dist(i_v) **then**
18: **if** CollisionFree$(\underline{p}, \mathcal{X}_{\text{obs}})$ **then**
19: dist$(i_v) =$ dist$(i_u) + \underline{c}$
20: Pred$(i_v) = i_u$
21: **else**
22: **if** dist$(i_u) + \overline{c} \leq$ dist(i_v) **then**
23: **if** CollisionFree$(\overline{p}, \mathcal{X}_{\text{obs}})$ **then**
24: dist$(i_v) =$ dist$(i_u) + \overline{c}$
25: Pred$(i_v) = i_u$
  return Backwards chain of predecessors from i_m

Algorithm 2 takes as input a collision-free, dynamically feasible path π^E between start and goal configurations – such as that returned by PrAC – as well as the set of all configurations C_π, obstacles $\mathcal{X}_{\text{obs}}$, cost function of motions c, and a non-negative natural number n. It also takes a function $\Psi : \mathbb{R} \to \mathbb{R}$ which is used to penalize reverse motion. That is, given two configurations $i, j \in \mathcal{X}$ with a motion p from i to j, we say that the cost of the motion p is $c(p)$, while the cost of the reverse motion p' from j to i that is identical to p but traversed backwards is $c(p') = \Psi(c(p))$.

The set V in Line 3 represents sampled configurations along the set of configurations C_π which includes all endpoints of the edges $(i_r, i_{r+1}) \in \pi^E$, as well as n additional random configurations along motions connecting these endpoints. Configurations in V must be in the order in which they appear along π^E. For each pair (i_u, i_v) of configurations with i_v appearing after i_u in π^E, Algorithm 2 attempts to connect i_u to i_v with either a motion from i_u to i_v (Line 11) or from i_v to i_u traversed backwards, selecting the cheaper of the two if possible (Lines 17-25).

Were we to form a graph with vertices V and with edges $(i_u, i_v) \in V^2$ where i_v occurs farther along C_π than i_u and where the optimal feasible motion from i_u to i_v is collision-free, then observe that this graph would be a directed, acyclic graph (DAG). Indeed, were this graph not acyclic, then the path $\pi^E(p_s, p_g)$ would contain a cycle implying that a configuration would appear at least twice in $\pi^E(p_s, p_g)$. This is impossible by construction of the PrAC algorithm which maintains two trees rooted at p_s, p_g, respectively. This motivates the following observation and Theorem:

Observation V.4. *The nested for loop in lines 9-10 of Algorithm 2 constructs a DAG and finds the minimum-cost path from p_s to p_g in the graph.*

Theorem V.5. *Let π_1^E be the input path to Algorithm 2 between configurations p_s, p_g with cost $c(\pi_1^E)$. Let π_2^E be the path returned by Algorithm 2 with cost $c(\pi_2^E)$. Then $c(\pi_2^E) \leq c(\pi_1^E)$. Moreover, if $n = 0$, this algorithm runs in time quadratic in the number of vertices along π_1^E.*

Proof. By Observation V.4, Algorithm 2 constructs a DAG containing configurations p_s, p_g as vertices. It also solves the minimum cost path problem on this DAG. Observe further that by construction of the DAG vertex set in Line 3, all configurations along the original path π_1^E are in V. Thus π_1^E is an available solution to the minimum path problem on the DAG. This proves that the minimum cost path can do no worse than $c(\pi_1^E)$. If $n = 0$, then V is the set of endpoints of edges in π_1^E. Therefore, $V \subseteq L$. Because all motions between lattice vertices have been pre-computed in $\mathcal{B}$, Lines 11,12 can be executed in constant time, and the nested for loop in lines 9-10 will thus run in time $O(m)^2$ where m is the number of vertices on the path π_1^E. □

Algorithm 2 is similar to Algorithm 1 in [9] with one critical difference. The latter algorithm adopts a greedy approach, connecting the start vertex i_1 in the path π^E to the farthest vertex $i_k \in \pi^E$ that can be reached without collisions. This process is then repeated at vertex i_k until the goal vertex is reached. This greedy approach also runs in time quadratic in the number of vertices along the input path but the cost of paths returned will be no lower than the cost of paths returned by Algorithm 2 proposed here. A simple example is illustrated in Figure 5. Using the control set in Figure 5, an initial path is computed from $i_1 = p_s$ to $i_6 = p_g$ (red). This path remains

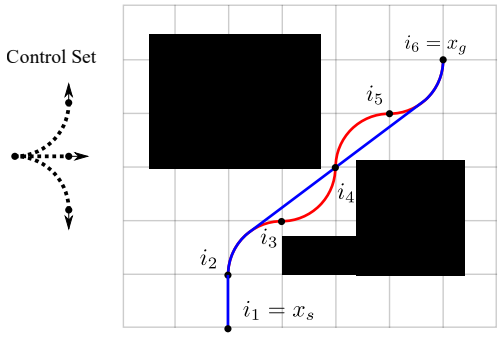

Fig. 5: Comparison of smoothing algorithms for same input path (red). Blue: path smoothed using Algorithm 2. Red: path smoothed using Algorithm 1 from [9] (no change from input path).

unchanged when smoothed using the proposed Algorithm in [9]. However, Algorithm 2 will return the less costly blue path.

VI. Experimental Results

We verify our proposed technique against two techniques in two common navigational settings: Hybrid A* [25], and the hybrid motion primitive, numerical optimization approach (denoted CGPrim) from [29]. We consider two variants of Hybrid A*: an unsmoothed version (HA*) and a smoothed version (S-HA*) that uses the smoothing approach outlined in [25]. The techniques described here were implemented in Python 3.7 (Spyder). Results were obtained using a desktop equipped with an AMD Ryzen 3 2200G processor and 8GB of RAM running Windows 10 OS. Start and goal configurations were not constrained to be lattice vertices. We assume that obstacles are known to the planner ahead of time and that the environment is noiseless. As such, the results that follow can be thought of as a single iteration of a full re-planning process.

A. Memory

The motions used in this section are $G3$ curves which take the form CCC, or CSC ("C" denotes a curved segment, and "S" a straight segment). Such motions can be easily generated from 9 constants in the case of motion planning without velocity, and 19 constants otherwise [41]. Instead of storing all configurations along a motion, we store only these constants.

B. Parking Lot Navigation

We begin by validating the proposed method against HA* and S-HA* in a parking lot scenario. Though HA* is not a new algorithm, more recent state of the art approaches use HA* to plan an initial motion which is then refined (e.g. [27], [28]). We are therefore motivated to compare the run-time and path quality of the approach proposed in this paper to HA* whose run-time is a lower bound of all state of the art algorithms using it as a sub-routine. In [41], we present a method of computing motions between start and goal configurations in the configurations space $\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi) \times \mathbb{R}^3$. Here, configurations take the form $(x, y, \theta, \kappa, \sigma, \rho)$ where (x, y, θ) represent the planar coordinates and heading of a vehicle, κ the curvature, σ the curvature rate (defined as $d\kappa/ds$ for arc-length s), and ρ the second derivative of curvature with respect to arc-length. We generate motions with continuously differentiable curvature profiles assuming that κ, σ, ρ are bounded in magnitude. The motivation for this work comes from the observation that jerk, the derivative of acceleration with respect to time, is a known source of discomfort for the passengers of a car [26]. In particular, minimizing the integral of the squared jerk is often used a cost function in autonomous driving [29]. Since this value varies with σ, keeping σ low and bounded is desirable in motion planning for autonomous vehicles.

Unfortunately, due to the increased complexity of the configuration space over, say, the configuration space used in the development of Dubins' paths, $\mathcal{X}_{\text{Dubins}} = \mathbb{R}^2 \times [0, 2\pi)$, solving TPBV problems in $\mathcal{X}$ takes on average two orders of magnitude more time than computing a Dubins' path. Though motions computed using the techniques in [42], [41] are more comfortable, and result in lower tracking error than Dubins' paths, they may be computationally impractical to use in motion planners that require solving many TPBV problems online. However, they prove to be particularly useful in the development of pre-computed motion primitives.

1) Lattice Setup & Pruning: The configuration space used here is $\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi) \times \mathbb{R}^3$ with configurations $(x, y, \theta, \kappa, \sigma, \rho)$. Motion primitives were generated using the MILP in (7) for a 15×20 square lattice with 16 headings and 3 curvatures, and a value of $t = 1.1$ (10% error from optimal). To account for the off-lattice start-goal pairs, we used a higher-fidelity lattice with 64 headings and 30 curvatures. Lattice vertex values of σ, ρ were set to 0. This results in a start set $\mathcal{S}$ with 12 starts given by (4). The cost c of a motion is defined by the arc-length of that motion. These motions were computed using our work in [41] with bounds on κ, σ, ρ:

$$\kappa_{\max} = 0.1982m^{-1},\ \sigma_{\max} = 0.1868m^{-2}\ \rho_{\max} = 0.3905m^{-3}, \tag{10}$$

which were deemed comfortable for a user [42], particularly at low speeds typical of parking lots. The spacing of the lattice x, y-values was chosen to be $r_{\min}/4$ for a minimum turning radius $r_{\min} = 1/\kappa_{\max}$. Finally, if the cost of the motion from $s \in \mathcal{S}$ to a vertex j was larger than 1.2 times the Euclidean distance from s to j for all $s \in \mathcal{S}$, then j was removed from the lattice. This technique which we dub *lattice pruning* is to keep the lattice relatively small, and to remove vertices for which the optimal motion requires a large loop. The value of 1.2 comes from the observation that the optimal motion from the start vertex $s = (0, 0, 0, \kappa_{\max})$ to $j = (r_{\min}, r_{\min}, \pi/2, \kappa_{\max})$ is a quarter circular arc of radius $r_{\min}$. The ratio of the arc-length of this maneuver to the Euclidean distance from s to j is $\pi/(2\sqrt{2}) \approx 1.11$. Thus using a cutoff value of 1.2 admits a sharp left and right quarter turn but is still relatively small.

2) Adding Reverse Motion: The motion primitives returned by the MILP in (7) are motions between a starting vertex $s \in \mathcal{S}$, and a lattice vertex $j \in L - \mathcal{S}$. As such, they are for forward motion only. To add reverse motion primitives to the control set E_s for each $s \in \mathcal{S}$, we applied the forward primitives to $\mathcal{S}$ in reverse. We then rounded the final configurations of these primitives to the closest lattice vertex. For each (x, y)-value of the final configurations, we select a single configuration (x, y, θ, κ) that minimizes arc-length. This is to keep the branching factor of an online search low. Finally, to each $s = (0, 0, \theta, \kappa) \in \mathcal{S}$ we add three

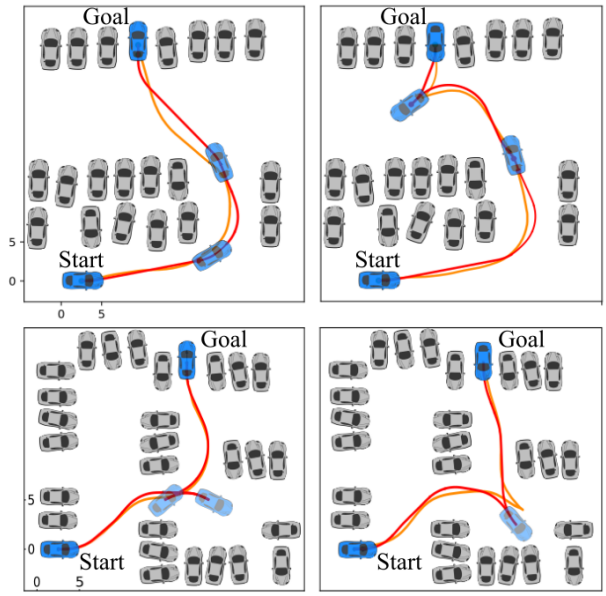

Fig. 6: Scenarios (a) - (d) (top left to bottom right). Red paths from proposed method, yellow from S-HA*.

primitives: $(0, 0, \theta, \pm\kappa_{\max}), (0, 0, \theta, 0)$ with a reverse motion penalty. These primitives reflect the cars ability to stop and instantaneously change its curvature.

3) Scenario Results: We verified our results in five parking lot scenarios (a)-(e). The first four scenarios illustrate our technique in parking lots requiring forward and reverse parking. The results are illustrated in Figure 6. Here, we have compared our approach to S-HA* using an identical collision checking algorithm, and using the same heuristic (that proposed in [25]). Initial paths for our approach were computed using $\lambda = 0.2$. These paths were then smoothed via Algorithm 2 with $n = 0$. Though the motions may appear similar, they are actually quite different. To evaluate the quality of the motions predicted, we use three metrics: the integral of the square jerk (IS Jerk), final arc-length, and runtime, and the results are summarized in Table I. These three metrics are expressed as ratios of the value obtained using S-HA* to those of the proposed. Values for the proposed motions before smoothing (using Algorithm 2) are presented in parenthesis. The runtime values for S-HA* for scenarios (a)-(e) were 280ms, 370ms, 275ms, 390ms, and 410ms, respectively. Because these techniques are determinisitc, no standard deviations are presented. The major difference between the two approaches can be seen in the IS Jerk Reduction. Because the motion primitives we employ are each $G3$ curves with curvature rates bounded by what is known to be comfortable the resulting IS Jerk values of proposed paths are up to 16 less than those computed with S-HA*. In fact, using the S-HA* approach may result in motions with infeasibly large curvature rates resulting in larger tracking errors and increased danger to pedestrians.

Despite the bounds on curvature rate, the final arc-lengths of curves computed using our approach are comparable to those of S-HA*. Further, though Reeds-Shepp paths (which are employed by HA* before smoothing) take on average two orders of magnitude less time to compute than $G3$ curves, the runtime performance of our method often exceeds that

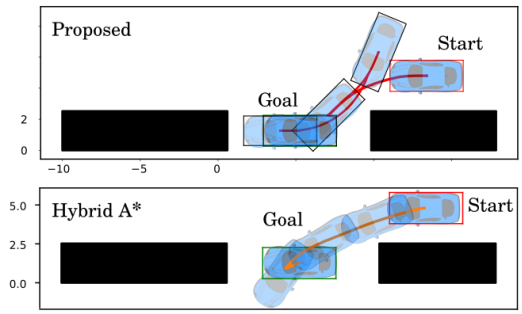

Fig. 7: Scenario (e).

of S-HA*. In fact, our proposed method takes, on average 6.9 times less time to return a path, exceeding the average runtime speedup of the method proposed in [29]. Moreover, the methods in [29] do not account for reverse motion, and assume that a set of way-points between start and goal configurations is known. It should be noted that values for HA* were not included in Table I owing to the moments of infinite jerk experienced at transitions between curvatures. However, the proposed approach took on average 4.6 times less time to compute than paths using HA* and featured an average length reduction of 1.4 over HA*.

The only scenario in which S-HA* produces a motion in less time than the proposed method is Scenario (c) in which S-HA* produced a path with no reverse motion (which accounts for the speedup). However, in order to produce this motion the path must feature moments of very large changing curvature resulting in an IS Jerk that is 16.3 times higher than the proposed method. The low run-time of the proposed method may due to the length of primitives we employ. It has been observed that HA* (which is used as a initial path for S-HA*) often takes several iterations to obtain a motion of comparable length to one of our primitives. This results in a much larger open set during each iteration of A*.

The final scenario we investigated is a parallel parking scenario (scenario (e)) illustrated in Figure 7. Though the S-HA* motion appears simpler, it requires an IS Jerk 16.7 times larger than what is considered comfortable. It should be noted that several other parallel parking scenarios in which the clearance between obstacles was decreased were considered. While the proposed method returned a path in each of these scenarios, HA* (and thus S-HA*) failed to produce a path in the allotted time. A further complex parking lot navigation scenario using the proposed approach is shown in Figure 1.

The resolution of the lattice used was $\sim 1.26m$. To verify the efficacy of our approach with higher resolution, we repeated the experiments above with a resolution of $0.5m$ as in [25] with $t = 1.2, \lambda = 0.1$. Though the size of the lattice increased by a factor of 6.4, the control set only increased by a factor of 4.1. Runtimes for these experiments was on average 1.8 times faster than S-HA* and 1.2 times faster than HA*. IS Jerk and length reduction increased by less than 1% after smoothing (via Algorithm 2) as compared to the proposed approach with lower resolution.

Scenario	IS Jerk Reduction	Length Reduction	Runtime Speedup
(a)	7.7 (7.3)	1.0 (0.8)	5.3 (5.9)
(b)	9.9 (9.5)	1.0 (0.8)	18.4 (18.7)
(c)	16.3 (15.6)	0.8 (0.7)	0.6 (0.8)
(d)	10.1 (9.9)	0.9 (0. 8)	7.3 (7.6)
(e)	16.7 (13.0)	0.6 (0.2)	3.0 (3.8)

TABLE I: Scenario Results

C. Speed Lattice

In this experiment, we generate a full trajectory (including both path and speed profile) for use in highway driving using our approach. Here, we use only forward motion as reverse motion on a highway is unlikely. This section compares the proposed method to HA*, S-HA*, and CGPrim. This latter approach was selected as a basis of comparison because it is a state-of-the-art approach that combines the use of motion primitives and numerical optimization.

In addition to developing $G3$ paths, our work in [41] details a method with which a trajectory with configurations $(x, y, \theta, \kappa, \sigma, \rho, v, a, \beta)$ may be computed. Here, v, a, β denote velocity and longitudinal acceleration, and longitudinal jerk respectively. The approach is to compute profiles of ρ and β that result in a trajectory that minimizes a weighted sum of undesirable trajectory features including the integral of the square (IS) acceleration, IS jerk, IS curvature, and final arc-length. The key feature of this approach is that both path (tuned by ρ) and velocity profile (tuned by β) are optimized simultaneously, keeping path planning in-loop during the optimization.

Computing trajectories via the methods outlined in [41] requires orders of magnitude more time than simple Reeds-Shepp paths. However, pre-computing a set of motion primitives where each motion is computed using the methods of [41] ensures that every motion used in PrAC is optimal for the user. Moreover, because we include velocity in our configurations (and primitives) we do not need to compute a velocity profile.

1) Lattice Setup & Pruning: Motion primitives were generated (7) for a 24×32 grid. Dynamic bounds for comfort were kept at (10). The x component of the lattice vertices were sampled every $r_{\text{min}}/6$ meters while the y components were sampled every $r_{\text{min}}/12$. Headings were sampled every $\pi/16$ radians (32 samples). We also assumed values of $\kappa = \sigma = \alpha = 0$ on lattice vertices. To account for off-lattice start-goal pairs, we use a higher-fidelity lattice with 128 headings, and 10 curvatures between $-\kappa_{\text{max}}, \kappa_{\text{max}}$. Finally, five evenly spaced velocities were sampled between 15 and 20 km/hr. It should be noted that this range could be changed to include 0 if that is desired without altering the methodology proposed here. Further, the performance of the proposed approach is similar for differing ranges if the fineness of the discretization is unchanged. A value of $n = 0$ was used for Algorithm 2.

2) Scenario Results: The highway scenario was chosen to closely resemble the roadway driving scenario in [29]. Initial paths for our approach were computed using $\lambda = 0.9$ and were smoothed via Algorithm 2 with $n = 0$ (though minimal smoothing was required). Results of this scenario are in Figure 8 while performance analysis is summarized in Table II. Performance is measured with three metrics: arc-length of the proposed motion, smoothness cost of the motion, maximum curvature obtained over the motion, the runtime speedup relative to HA*. The final column of the Table indicates whether a velocity profile was included during the motion computation. In Table II, two values of smoothness cost are given in the form of a tuple $(\text{Smoothness}_1, \text{Smoothness}_2)$ where

$$\text{Smoothness}_1 = \sum_{i=1}^{N-1} |\Delta \mathbf{x}_{i+1} - \Delta \mathbf{x}_i|^2,$$

$$\text{Smoothness}_2 = \int_0^{s_f} \kappa(s)^2 ds.$$

The first measure is used in [25], where N configurations are sampled along a motion, with $\mathbf{x}_i$ the vector of x, y-components of the i^{th} configuration, and $\Delta \mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$. The second measure is used in [29]. The first three methods appearing in the Table II are computed directly from the motions in Figure 8, while the last comes from [29] for an identical experiment.

The authors of [29] report an average runtime speedup of 4.5 times as compared to HA* for the path planning phase (without speed profile). On average, the proposed computed a full motion, including a speed profile 4.7 times faster than the time required for HA* to compute a path. Further, the proposed approach features significantly reduced the smoothness costs.

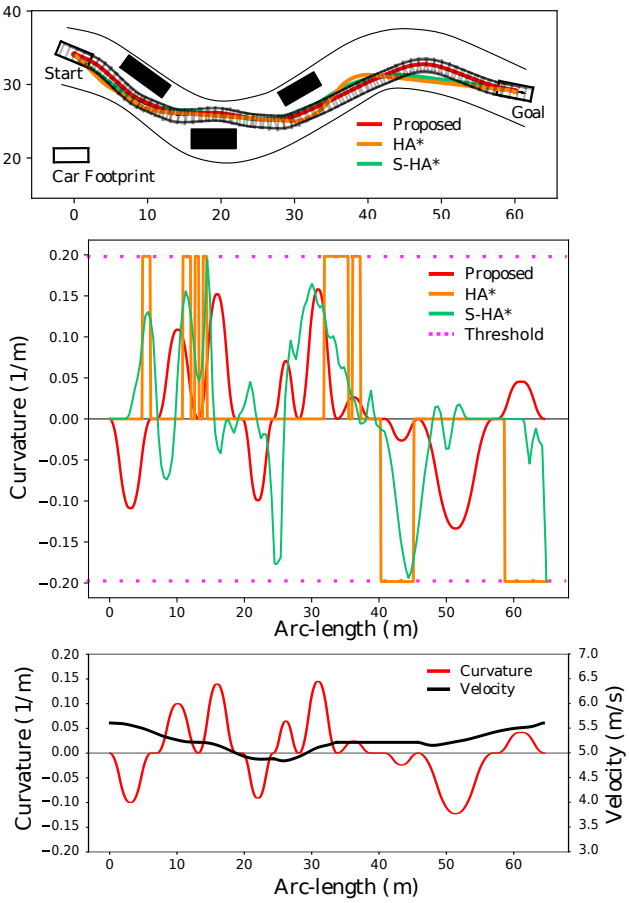

Fig. 8: Result of highway maneuver with several obstacles. Top: resulting motions, mid: curvature over motions, bottom: proposed curvature and velocity profile.

Method	Length (m)	Smoothness Cost	Max Curvature (m^{-1})	Runtime Speedup	Velocity
HA*	65.0	(1.35, 0.77)	0.198	1.0	No
S-HA*	64.9	(0.36, 0.46)	0.194	0.6	No
Proposed	64.6	(0.17, 0.28)	0.158	4.7	Yes
CGPrim	65.8	(- , 0.44)	0.189	4.5	No

TABLE II: Road navigation results. CGPrim from [29] Table 3 for identical planning problem.

VII. Discussion & Conclusion

We present a novel technique to compute an optimal set of motion primitives for use in lattice-based motion planning by way of a mixed integer linear program. Further, we propose an A*-based algorithm using these primitives to compute motions between configurations and a post-processing smoothing algorithm to remove excessive oscillations from the motions.

The results of the previous section illustrate the effectiveness of the proposed technique. Indeed, feasible, smooth motions were computed in both parking lot and highway settings. The proposed approach results in motions with an improved level of comfort in two common metrics: integral squared jerk, and smoothness as compared to state of the art approaches. While these latter techniques may be coupled with additional smoothing techniques, this process will only increase the runtime. While short turns and straight lines as primitives (as employed in [25]) lend versatility in movement – a useful feature in parking lot scenarios, they also result in larger graphs which must be traversed by a planner. A better approach resulting in smoother motions with increased runtime performance is to create longer compound actions using these basic motions, study which sequence of motions can be generated using others to within an acceptable tolerance, pre-smooth these motions (e.g. by using $G3$ curves), and store them as an action set. This is precisely the methodology employed by this work. If the motion primitives already include velocity as a state, then a velocity profile may be easily computed. This paper has not proposed a controller to track the reference paths we compute nor does it propose a framework for re-planning. Further, the resolution for the lattices used in Section VI was chosen based on trial and error given the experiments. Though the choice of resolution is crucial for lattice-based motion planning, this work focused primarily on lattice traversal rather than lattice generation. These are left for future work.

References

[1] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001 vol.2.

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[3] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[4] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, "Receding-horizon lattice-based motion planning with dynamic obstacle avoidance," in *2018 IEEE CDC*, pp. 4467–4474.

[5] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2015.

[6] M. Tiger, D. Bergström, A. Norrstig, and F. Heintz, "Enhancing lattice-based motion planning with introspective learning and reasoning," *IEEE RA-L*, vol. 6, no. 3, pp. 4385–4392, 2021.

[7] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *2005 IEEE/RSJ IROS*, pp. 3231–3237.

[8] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[9] R. Oliveira, M. Cirillo, B. Wahlberg *et al.*, "Combining lattice-based planning and path optimization in autonomous heavy duty vehicle applications," in *2018 IEEE Intelligent Vehicles Symposium*, pp. 2090–2097.

[10] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 46–61, 2018.

[11] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *2011 IEEE/RSJ IROS*, pp. 2172–2179.

[12] A. Botros and S. L. Smith, "Computing a minimal set of t-spanning motion primitives for lattice planners," in *2019 IEEE/RSJ IROS*, pp. 2328–2335.

[13] A. Rusu, S. Moreno, Y. Watanabe, M. Rognant, and M. Devy, "State lattice generation and nonholonomic path planning for a planetary exploration rover," in *65th International Astronautical Congress 2014 (IAC)*, vol. 2, p. 953.

[14] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[15] 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.

[16] L. E. Kavraki, P. Svestka, J. . 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, pp. 566–580, 1996.

[17] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, and M. Pavone, "Revisiting the asymptotic optimality of RRT," in *2020 IEEE ICRA*, pp. 2189–2195.

[18] S. D. Pendleton, W. Liu, H. Andersen, Y. H. Eng, E. Frazzoli, D. Rus, and M. H. Ang, "Numerical approach to reachability-guided sampling-based motion planning under differential constraints," *IEEE RA-L*, vol. 2, no. 3, pp. 1232–1239, 2017.

[19] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 1826–1848, 2019.

[20] H. Fuji, J. Xiang, Y. Tazaki, B. Levedahl, and T. Suzuki, "Trajectory planning for automated parking using multi-resolution state roadmap considering non-holonomic constraints," in *2014 IEEE Intelligent Vehicles Symposium (IV)*, pp. 407–413.

[21] C. G. L. Bianco and A. Piazzi, "Optimal trajectory planning with quintic g2-splines," in *2000 IEEE Intelligent Vehicles Symposium (IV)*, 2000, pp. 620–625.

[22] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges," *Sensors*, vol. 18, no. 9, p. 3170, 2018.

[23] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the driftless case," in *2015 IEEE ICRA*, pp. 2368–2375.

[24] Z. Zhu, E. Schmerling, and M. Pavone, "A convex optimization approach to smooth trajectories for motion planning with car-like robots," in *2015 IEEE CDC*, pp. 835–842.

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

[26] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 163–168.

[27] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2021.

[28] X. Zhang, A. Liniger, A. Sakai, and F. Borrelli, "Autonomous parking using optimization-based collision avoidance," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 4327–4332.

[29] Y. Zhang, H. Chen, S. L. Waslander, J. Gong, G. Xiong, T. Yang, and K. Liu, "Hybrid trajectory planning for autonomous driving in highly

constrained environments," *IEEE Access*, vol. 6, pp. 32 800–32 819, 2018.
[30] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," *Advances in neural information processing systems*, vol. 26, pp. 2616–2624, 2013.
[31] L. Jarin-Lipschitz, J. Paulos, R. Bjorkman, and V. Kumar, "Dispersion-minimizing motion primitives for search-based motion planning," *arXiv preprint arXiv:2103.14603*, 2021.
[32] L. Palmieri, L. Bruns, M. Meurer, and K. O. Arras, "Dispertio: Optimal sampling for safe deterministic motion planning," *IEEE RA-L*, vol. 5, no. 2, pp. 362–368, 2019.
[33] D. Peleg and A. A. Schäffer, "Graph spanners," *Journal of graph theory*, vol. 13, no. 1, pp. 99–116, 1989.
[34] A. Botros, N. Wilde, and S. L. Smith, "Learning control sets for lattice planners from user preferences," in *WAFR*. Springer, 2020, pp. 381–397.
[35] R. De Iaco, S. L. Smith, and K. Czarnecki, "Learning a lattice planner control set for autonomous vehicles," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 549–556.
[36] A. Botros, "Lattice-based motion planning with optimal motion primitives," Ph.D. dissertation, University of Waterloo, 2021.
[37] B. Korte and J. Vygen, *Combinatorial optimization*, 6th ed. Springer, 2018.
[38] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis, "Time constrained routing and scheduling," *Handbooks in operations research and management science*, vol. 8, pp. 35–139, 1995.
[39] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.
[40] J. Chen, R. C. Holte, S. Zilles, and N. R. Sturtevant, "Front-to-end bidirectional heuristic search with near-optimal node expansions," *arXiv preprint arXiv:1703.03868*, 2017.
[41] A. Botros and S. L. Smith, "Tunable trajectory planner using g3 curves," *IEEE Transactions on Intelligent Vehicles*, 2022.
[42] H. Banzhaf, N. Berinpanathan, D. Nienhüser, and J. M. Zöllner, "From $G2$ to $G3$ continuity: Continuous curvature rate steering functions for sampling-based nonholonomic motion planning," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 326–333.

Alexander Botros is a postdoctoral fellow in the Autonomous Systems Lab at the University of Waterloo. His research focuses primarily on local planners and trajectory generation for autonomous vehicles. In particular, Alex is researching the problem of computing minimal t-spanning sets of edges for state lattices with the goal of using these sets as motion primitives for autonomous vehicles. Alex Completed his undergraduate and M.Sc. engineering work at Concordia University in Montreal, and his Ph.D. at the University of Waterloo.

Stephen L. Smith (S'05–M'09–SM'15) received the B.Sc. degree from Queen's University, Canada in 2003, the M.A.Sc. degree from the University of Toronto, Canada in 2005, and the Ph.D. degree from UC Santa Barbara, USA in 2009. From 2009 to 2011 he was a Postdoctoral Associate in the Computer Science and Artificial Intelligence Laboratory at MIT, USA. He is currently a Professor in Electrical and Computer Engineering at the University of Waterloo, Canada and a Canada Research Chair in Autonomous Systems. His main research interests lie in control and optimization for autonomous systems, with an emphasis on robotic motion planning and coordination.