High-Dimensional Lattice Planning with 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 then combined online to generate more 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 present an \( A^* \)-based algorithm to solve the motion planning problem using these primitives. Finally, we present 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.

I. INTRODUCTION

Broadly, the motion planning problem is to plan a collision free motion between configurations of a vehicle and optimizes desirable properties like travel time or comfort \([1]\). The motion can be used as a reference for a tracking controller \([2]\) 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 \([3]\) owing in part to potentially complex vehicle constraints that impede the resolution of two point boundary value (TPBV) problems. Therefore, simplifying assumptions are made. The variety in possible simplifications has given rise to several planning techniques. These typically fall into one of four categories \([4]\): 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 and is one of the most common approaches to solving the motion planning problem \([4]\, [5]\). 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 \([5]\, [6]\). Elements of this control set can then be concatenated in real time to form complex maneuvers. Lattice-based planners simplify the motion planning problem by limiting the set of available motions instead of simplifying their kinodynamics—guaranteeing feasibility. Computationally expensive TPBV problems are solved offline between lattice vertices. Further, the regularity of the lattice structure allows for theoretical guarantees on the cost of lattice-optimal paths compared to free space optimal under certain assumptions \([9]\). These guarantees do not require the fineness of the lattice grid structure to approach infinity, which is typically required in sampling-based techniques. However, 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 \([10]\, [11]\) 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 \([10]\) 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—with whose costs are no more than a factor of \( t \) from optimal. They propose to find the smallest set that, for a given value \( t > 1 \), 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 \([11]\). Hitherto, no non brute-force solution to the MTSCS problem had been proposed.

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 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 \([10]\). When motion primitives are concatenated, invariant states may experience a discontinuous jump if all primitives are computed relative to a single starting vertex \([10]\). 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 \([12]\, [13]\). A solution approach to this critique that we employ is to compute a control set for a lattice with

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)
several starting vertices [5], [10].

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 [6] 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.

- **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.

- **Planning 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 [11], we introduced a 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

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*) [14], and Probabilistic Roadmap (PRM) [15]. 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 [4]. Sampling-based planners, while not complete, can be asymptotically complete with a convergence rate that worsens as the dimensionality of the problem is increased [16]. To improve the performance of RRT* in higher dimensions, the authors of [17] 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.

The usefulness of sampling based techniques is considered to be restricted to unconstrained motion planning problems [18], limiting their use in scenarios like highway driving.

Techniques using the interpolating curve approach include fitting Bezier curves, Clothoid curves [19], or polynomial splines [20] 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 [4], [21]. 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 [4], [21]. In [22], 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 [23]. Here, motions with non-affine non-holonomic constraints and piece-wise linear velocity profiles are developed. The authors of [23] illustrate the benefits of computing way-points using a system of similar complexity to the desired final motion. However, by the nature of the simplification, it is possible for infeasible way-point paths to be developed. A common critique of both interpolating curve and numerical optimization approaches is their reliance on global way-points [4], [21]. Moreover, it can be difficult to specify additional state constraints (e.g., curvature constraints, etc.) [4], [21].

Lattice-based planners are versatile in the problems they address. The authors of [24], [25] demonstrate lattice adaptations made to account for the structured environments of urban roads for use in autonomous driving, while in [26], a set of motion primitives is computed (based on experience) for a UAV exploring mine-shafts. In [12], [27], the authors use motion primitives to traverse a lattice with states given by position and heading. In both these works, the authors use a rounding technique to account for off-lattice primitive concatenations due to non-cardinal headings. Compound motions comprised of several primitives are discontinuous in curvature – a known source of slip and discomfort [28]. To alleviate this curvature discontinuity, the authors of [29] use the techniques of [27] to compute an initial trajectory which is then smoothed via numerical optimization. This approach requires at least as much computation time as the methods proposed in [27]. On the other hand, the authors of [30] compute motion primitives that minimize the integral of the squared jerk over the motion but limit their results to forward motion. In [3] the authors consider the control set to be the entirety of the lattice. The large control set may necessitate coarser lattices.

The choice of motion primitives is of particular interest. Typically motion primitives are chosen to achieve certain objectives for the paths they generate. For example, in [31], the authors introduce the notion of probabilistic motion primitives which achieve a blending of deterministic motion primitives to better simulate real user behavior. In [32], Dispertio – a dispersion minimizing algorithm from [33] – is used to compute a set of motion primitives that result in motions with minimum dispersion. In [34], a set of motion primitives is computed that relies on a set of user-specified maneuvers. In [6], on the other hand, the authors present the notion of using a MTSCS of motion primitives. These are similar to graph t-spanners first proposed in [35]. The process of computing such...
a set is elaborated in [36] 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 [36, 37] to provide an improved set of user-specific motion primitives.

In [10], 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 & MULTI-START LATTICES

In this section, we introduce the notion of a multi-start lattice, and illustrate its uses in motion planning. We also formulate the Minimum t-Spanning Control Set Problem.

A. Lattice-Based Motion Planning

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 sequence of configurations 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 $c : \mathcal{M} \rightarrow \mathbb{R}_{\geq 0}$. 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 $p_s, p_g \in \mathcal{X} - \mathcal{X}_{\text{obs}}$, 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 $\mathcal{X} - \mathcal{X}_{\text{obs}}$, 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 $i, j$ be vertices in $L$, and let $p$ be a motion that solves Problem II.1 for $p_s = i, p_g = j$, written $i \rightarrow p = j$. 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 $E$ 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 actions available at $s$.

B. Multi-Start Lattices

There are examples for which a single start is insufficient to capture the full variety of motions. For example, let $\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi], L = \mathbb{Z}^2 \times \{i\pi/4, i = 0, \ldots, 7\}, s_0 = (0, 0, 0)$,

$$\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi], L = \mathbb{Z}^2 \times \{i\pi/4, i = 0, \ldots, 7\},$$

$$s_0 = (0, 0, 0),$$

$$\text{denote configurations space, lattice, and starting vertex, respectively – as in Figure 2 (a). If } i = (1, 1, \pi/4) \in L \text{ and } j = (2, 2, \pi/4) \in L, \text{ then the motion } p_0 \text{ from } i \text{ to } j \text{ is such that } s_0 \cdot p_0 \not\in L - \text{ that is, the motion } p_0 \text{ applied to } s_0 \text{ is not in the lattice. Therefore, the simple diagonal motion } p_0 \text{ will not appear in any action set } E \text{ which may result in motions with excessive oscillations.}

We propose a solution to this problem by way of a multi-start lattice. Given a set of starts $S \subseteq \mathbb{R}^2$, 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 $s \in S$ such that $s \cdot p \in L$. In the example in Figure 2 (b), let $S = \{s_0 = (0, 0, 0), s_1 = (0, 0, \pi/4)\}$, then the motion $p_0$ from $i$ to $j$ can be applied to $s_1$ to reach a lattice vertex (namely $i$). Therefore, $i \oplus p_0$ is a valid concatenation.

An ideal set of starts $S$ 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

$$\mathcal{X} = \mathbb{R}^2 \times [0, 2\pi) \times U_0 \cdots \times U_N,$$
evenly spaced samples, and for \( i = 0 \ldots N \), it partitions \( U_i \)
into \( m_i \) evenly spaced samples. For \( X, L \) from \ref{4}, \( \ref{3} \), let
\[
S = \left\{ (0, 0, \theta, u_0, \ldots, u_N) : \begin{align*}
\theta &\in \left( j \pi / 2^{n^2 - 1}, j = 0, \ldots, 2^{n^2 - 2} - 1 \right), \\
u_i &\in \{ U_i^0 + j/m_i, j = 0 \ldots (U_i^1 - U_i^0)m_i \}.
\end{align*} \right\}
\]
This starting set has the property that for any vertices \( i, j \in L \), the
motion \( p \) from \( i \) to \( j \) is such that \( \exists s \in S \) with \( s \oplus p \in L \).

C. Motion Planning with a Multi-Start Control Set

In this section, we describe how multi-start lattices can be
used in motion planning and we formulate the Minimum \( t \)-
Spanning Control Set (MTSCS) problem. We assume that for
all \( i, j \in X \), there exists a motion \( p \) solving Problem \ref{1} \( \text{from } i \) to \( j \) in the absence of obstacles. We assume 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 — that is, if \( p_1 \) is a motion from \( i \) to \( j \) \( \in X \) and \( p_2 \) is a motion from \( i \) to \( j \) by way of \( r \in X \), then \( c(p_1) \leq c(p_2) \). This cost is left general, and may represent
travel time, integral of the squared jerk, etc.

Given \( (X, L, S, c) \) of configurations space, lattice, starting
set and cost of motions, respectively, the high level idea behind
motion planning using a multi-start control set is the following.
For each \( s \in S \), we pre-compute a set \( B_s \) of cost-minimizing feasible motions from \( s \) to each vertex \( i \in L - S \), and we
denote \( B = \bigcup_{s \in S} B_s \). We select a subset \( E \subseteq B \) which we
use as an action set during an online search in the presence
of obstacles. To this end, we offer the following definition:

Definition II.2 (Relative start). For vertex \( i \in L \), the relative
start of \( i \) is the vertex \( R(i) \in S \) 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 \( E \subseteq B \), we choose a set \( E_s \) representing
the action set for all lattice vertices with relative start \( s \), and let \( E = \bigcup_{s \in S} E_s \). If \( L \) and \( S \) are given by \ref{3},
\ref{4}, respectively, then the relative start of any configuration
\( i = (x, y, \theta, u_1, \ldots, u_N) \in L \) is \( R(i) = (0, 0, \theta', u_1, \ldots, u_N) \) where
\[
\theta' = \frac{\pi}{2^{n^2 - 1}} \left( \left( \frac{2^{n^2 - 2} \theta}{\pi} \right) \mod 4 \right). \]
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 \( E \subseteq B \), we denote by \( \overline{E} \) the set of all tuples
\((i, j) \in L^2 \) such that if \( p \) is the optimal feasible motion from
\( i \) to \( j \), then \( p \in E_{R(i)} \):
\[
\overline{E} = \{(i, j) \in L^2 : (i \oplus p = j) \Rightarrow p \in E_{R(i)} \}. \tag{5}
\]
Computing a motion between lattice vertices given a set \( E \)
is equivalent to computing a shortest path in the weighted
directed graph \( G = (L, E_{CF}, c) \) where \( E_{CF} \subseteq E \) is the set
of all collision-free edges in \( E \), and for all \((i, j) \in E_{CF},
c((i, j)) \) 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 \((X, L, S, c, E)\):

Definition II.3 (Path using \( E \)). A path using \( E \) from \( s \in S \)
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
\( G_{Free} = (L, E, c) \) in the absence of obstacles.

Definition II.4 (Motion cost using \( E \)). The cost of a motion
using \( E \) from \( s \in S \) to \( j \in L \), denoted \( d^E(s, j) \) is the cost of the
path \( \pi^E(s, j) \) in \( G_{Free} = (L, E, c) \).

Let \( i \in L - S, \ s \in S \) and let \( p \) a cost minimizing motion from
\( s \) to \( i \). Observe that if \( E = B \), then \( d^E(s, j) = c(p) \). This
implies that using \( B \) as a control set will result in paths with
minimal cost. However, because \( B \) can be 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 the minimal cost for all \( j \in L \). This
motivates the following definition:

Definition II.5 (\( t \)-Error). Given the tuple \((X, L, S, c, E)\), the
\( t \)-error of \( E \) is defined as
\[
\text{tEr}(E) = \max_{s \in S} \left| \frac{d^E(s, j) - d^B(s, j)}{d^B(s, j)} \right|. \tag{6}
\]
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 \( j \) taken over all starts \( s \) and
vertices \( j \) in the lattice. The \( t \)-error can be used to evaluate
the quality of a control set \( E \):

Definition II.6 (\( t \)-Spanning Set). Given the tuple
\((X, L, S, c, E)\), and a real number \( t \geq 1 \), we say that a
set \( E \) is a \( t \)-spanner of \( L \) (or \( t \)-spans \( L \)), if \( \text{tEr}(E) \leq t \).

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 II.7 (Minimum \( t \)-spanning Control Set Problem).
Input: A tuple \((X, L, S, c)\), and a real number \( t \geq 1 \).
Output: A control set \( E = \bigcup_{s \in S} E_s \) that \( t \)-spans \( L \) where
\( \max_{s \in S} |E_s| \) is minimized.

Using a solution \( E \) to Problem \ref{1.7} as a control set for lattice
planning 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 \( \max_{s \in S} |E_s| \)
is minimized. Second, for every \( i \in L - S, s \in S \), it must hold that \( d^E(s, i) \leq td^B(s, i) \). Thus, Problem \ref{1.7}
represents a trade-off between branching factor and motion quality. As
an alternative, the framework presented here is capable of
computing a \( t \)-spanning set \( E \) such that the sum \( \sum_{s \in S} |E_s| \)
is minimized. However, this last does not ensure a minimum value
of the branching factor at each vertex.

III. MAIN RESULTS

In this section we present a MILP formulation of Problem
\ref{1.7} and algorithms to compute and smooth motions using
computed control sets.

A. Multi-Start MTSCS Problem: MILP Formulation

In \cite{38} Theorem 5.2.1, we show that Problem \ref{1.7} is NP-
hard, which motivates the MILP formulation presented here.
Let \((X, L, S, c)\) denote configuration space, lattice, start set,
and cost of vertex-to-vertex motions in $L$, respectively. For any motion $q \in B = \bigcup_{s \in S} B_s$, 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 $s \in S$ 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 $E = \bigcup_{s \in S} E_s$ be a solution to Problem II.7. Let $G_{\text{Free}} = (L, E, c)$ be the weighted directed graph with edges $E$ given in $[5]$. For each $s \in S$ and each $(i, j) \in E$, 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 $(i, j) \in E$. For each $s \in S$ we construct a new graph
\[
T^s = (L, E^s_T),
\]
whose edges $E^s_T$ are defined as follows: let
\[
P^s = \bigcup_{i \in L-S} \pi^E(s, i).
\]
Thus $P^s$ is the set of all minimal cost paths from $s$ to vertices $i \in L - S$ in the graph $G_{\text{Free}}$. These paths are expressed as a sequence of edge copies $(i, j)^s$ where $(i, j) \in E$. For each $i \in L - S$, and for each $s \in S$, if $P^s$ contains at least two paths $\pi^E_1, \pi^E_2$ from $s$ to $i$, determine the last common vertex $j$ in paths $\pi^E_1, \pi^E_2$, and delete the the copy of the edge in $\pi^E_2$ whose endpoint is $j$. The remaining edges are the set $E^s_T$. The graph $T^s$ has a useful property defined here:

**Definition III.1 (Arborescence).** From Theorem 2.5 of [39], 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 $s \in S$, the graph $T^s$ an arborescence rooted at $s$. This is shown in the following Lemma:

**Lemma III.2 (Arborescence Lemma).** Let $E = \bigcup_{s \in S} E_s$ be a solution to Problem II.7 and $G_{\text{Free}} = (L, E, c)$. For each $s \in S$, $T^s$ (in \textbf{(4)}) is an arborescence rooted at $s$. Further, $\forall i \in L - S$, $d^E(s, i)$ is the length of the path in $T^s$ to $i$.

**Proof.** Let $s \in S$. 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 II.7, then $E$ t-spans $L$. In particular, there must be at least one path, $\pi^E(s, i)$ using $E$ from $s$ to each $i \in L$ implying that $\pi^E(s, i) \in P^s$. Since $E^s_T$ deletes duplicate paths from $P^s$, there must be a path in $T^s$ from $s$ to $i$. By construction of the edge set $E^s_T$, duplicate paths (with equal cost) are deleted ensuring the uniqueness of paths from $s$ to each $i \in L$ completing the proof 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 $d^E(s, i)$. \hfill $\Box$

Lemma III.2 implies that $E$ is a $t$-spanner of $L$ if and only if $\forall s \in S$ 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 some $q \in E$, and such that the cost of the path from $s$ to any vertex $i \in L$ in $T^s$ is no more than a factor of $t$ from the optimal path from $s$ to $i$. Indeed, the forward implication follows from Lemma III.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 be a $t$-spanning control set of $L$:

**Usable Edge Criteria:** For any $q \in B$, for any $s \in S$, 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 $s \in S, q \in B, (i, j) \in S_q$, 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 cost of the motion from $i$ to $j$.

**$t$-Spanning Criteria:** For any vertex $j \in L - S$, and for any start $s \in S$, 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 $s \in S$.

We now present a MILP encoding of these criteria. Let $|L| = n$ and all the vertices are enumerated as $1, 2, \ldots, n$ with $s \in S$ taking the values $1, \ldots, m$ for $m \leq n$. For any control set $E = \bigcup_{s \in S} E_s$, define $m(n - m - 1)$ decision variables:
\[
y_q^s = \begin{cases} 1, & \text{if } q \in E_s \\ 0, & \text{otherwise} \end{cases}, \quad q = m + 1, \ldots, n
\]

For each edge $(i, j) \in L^2$, and each $s \in S$ 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 $s \in S$) 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 $L^t = L - S$. The criteria above can be encoded as the following MILP:

\[
\text{min}_{K \in \mathbb{R}} K
\]
\[
s.t. \forall s \in S
\]
\[
\sum_{q \in L'} y_q^s \leq K,
\]
\[
x_{ij}^s - y_q^s \leq 0, \quad \forall (i, j) \in S_q,
\]
\[
z_i^s + c_{ij} - z_j^s \leq M_{ij}^s (1 - x_{ij}^s), \quad \forall (i, j) \in L \times L',
\]
\[
z_j^s \leq t c_{sj}, \quad \forall j \in L',
\]
\[
\sum_{i \in L} x_{ij}^s = 1, \quad \forall j \in L',
\]
\[
x_{ij}^s \in \{0, 1\}, \quad \forall (i, j) \in L \times L'
\]
\[
y_q^s \in \{0, 1\}, \quad \forall q \in B.
\]

where $M_{ij}^s = t c_{si} + c_{ij} - c_{sj}$. The objective function (7a) together with constraint (7b) minimizes $\max_{s \in S} |E_s|$ as in
Indeed, requires that if requires that \( x_{ij} = 0 \) for all \((i, j) \in S, s' \in S \). Alternatively, if \( y_{ij} = 1 \), then \( x_{ij} \) is free to take values 0 or 1 for any \((i, j) \in S, s' \in S \). Thus constraint (7c) encodes the Usable Edge Criteria.

### Constraint (7d):
Constraint (7d) takes a similar form to Equation (3.7a). Note that \( M_{ij}^y \geq 0 \) for all \((i, j) \in B, s \in S \).

If \( x_{ij} = 1 \), then \((i, j)^*\) is on the path in \( T^* \) to vertex \( j \) and (8) reduces to \( z^*_{ij} \geq c_{ij} - c_{x_{ij}} \) which encodes the Cost Continuity Criteria. If, however, \( x_{ij} = 0 \), then (9) reduces to \( z^*_{ij} - z^*_{ij} \leq \) which holds trivially by constraint (7c) and by noting that \( z^*_{ij} \geq c_{ij}, \forall j \in L \) by the triangle inequality.

### Constraint (7f):
Constraint (7f) together with constraint (7d) yield the Arborescence Criteria. Indeed for all \( s \in S \), by Theorem 2.5 of [39], \( T^* \) is an arborescence rooted at \( s \) if every vertex in \( T^* \) other than \( s \) has exactly one incoming edge, and \( T^* \) contains no cycles. The constraint (7f) ensures that every vertex in \( L \), which is the set of all vertices in \( T^* \) other than those in \( S \), has exactly one incoming edge, while constraint (7d) ensures that \( T^* \) has no cycles. Indeed, suppose that a cycle existed in \( T^* \), and that this cycle contained vertex \( i \in L \). Suppose that this cycle is represented as

\[
i \rightarrow j \rightarrow \cdots \rightarrow k \rightarrow i.
\]

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 \( z^*_{ij} < z^*_{ij} \) for any \((i, j) \in L^2 \). Therefore,

\[
z_1 < z_2 < \cdots < z_k < z_i,
\]

which is a contradiction.

### B. Motion Planning With a MTSCS

The previous section illustrates how to compute a control set \( E = \bigcup_{s \in S} E_s \) given the tuple \((\mathcal{X}, L, S, c)\). We have also presented a description of how such a control set may be used during an online \( A^* \)-style search of the graph \( G = (L, E_{\text{CF}}, c) \) from a start vertex \( p_s \) to a goal vertex \( p_g \) in \( L \). Here, the edge set \( E_{\text{CF}} \) is the set of pairs \((i, j) \in L^2 \) such that there exists a collision-free motion \( p \in E_{\text{CF}}(i) \) from \( i \) to \( j \) such that \( R(i) \in S \) 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, \textit{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 \( g \) Score, \( g(i) \) representing the current minimal cost to reach vertex \( i \) from the starting vertex \( p_s \), and the \( f \) Score, \( f(i) = g(i) + \lambda h(i) \) which is the sum of the \( g \) Score and an estimated cost to reach the goal vertex \( p_g \) by passing through \( i \) given heuristic \( h \). We implement the weighted \( f \) Score from \([41]\). 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 \( g \) Score and heuristic are weighted equally and standard \( A^* \) is recovered. However, as \( \lambda \) approaches 0, more weight is placed on the heuristic which promotes \textit{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 \( \lambda \) work well for longer maneuvers like traversing a parking lot.

### Expanding Start and Goal Vertices:
We employ the bi-directional algorithm from [42] 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 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

\[
h(i) = \min_{j \in O_s} h'(i,j) + g(p_g, j), \quad \forall i \in O_s
\]

\[
h(j) = \min_{i \in O_s} h'(j,i) + g(p_s, i), \quad \forall j \in O_g
\]

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 \([4]\), let Round\((v)\) denote a function that returns the element of \( L \) computed by rounding each state of \( v \) to the closest discretized 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 \( p_s, p_g \in \mathcal{X} \) (e., \([8]\)) 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 \([41]\). We compute a control set \( E_{\text{off}} \) of primitives from off-lattice configurations 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 \( Tol = (x_{tol}, y_{tol}, \theta_{tol}, u_{0, tol}, \ldots, u_{N, tol}). \) 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 possess 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.

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

**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 \( \pi \) be the set of vertices in \( O_s \) that is to be expanded. Let \( \pi \) be the optimal path from \( i \) to \( p_g \) in \( G \). Then \( \pi \) must pass through a vertex \( r \) in \( O_s \). This result holds by the completeness of A* with initial configuration \( p_g \) with paths traversed in reverse. Then by the triangle inequality, \( c(\pi) \geq c(\pi_{ECF}(i, r)) + g(p_g, r) \). Finally, because \( h' \) is an admissible heuristic, \( c(\pi_{ECF}(i, r)) + g(p_g, r) \geq h'(i, r) + g(p_g, r) \geq \min_{j \in O_s} h'(i, j) + g(p_g, j) = h(i) \). Therefore, \( h(i) \leq c(\pi) \) which concludes the proof. \( \square \)

**Algorithm 1 Generate Off-Lattice Control Set**

1: procedure \( \text{OFFLATTICE}(L, E, Tol) \)
2: \( E_{off} \leftarrow \emptyset \)
3: \( Q \leftarrow \{2ix_{tol}, i = 0, \ldots, \alpha/2x_{tol}\} \times \{2iy_{tol}, i = 0, \ldots, /2y_{tol}\} \times \{2\theta_{tol}, i = 0, \ldots, (\pi-\theta_{tol})/\theta_{tol}\} \times \prod_{i=0}^{N}(U_{1}^{i} + 2ju_{i, tol}, j = 0, \ldots, (U_{1}^{j} - U_{0}^{j})/2u_{i, tol}). \)
4: for \( q \in Q \) do
5: \( (x_q, y_q, \theta_q, u_{0,q}, \ldots, u_{N,q}) \leftarrow q \)
6: \( E_q \leftarrow \emptyset \)
7: \( q' \leftarrow \text{Round}(q) \)
8: for \( p \in E_{R}(q') \) do
9: \( (x_p, y_p, \theta_p, u_{0,p}, \ldots, u_{N,p}) \leftarrow p \)
10: for \( x \in \{x_p, x_{p} + \text{sign}(x_{p} - x_{q})\} \) do
11: for \( y \in \{y_{p}, y_{p} + \text{sign}(y_{p} - y_{q})\} \) do
12: \( p' \leftarrow (x, y, \theta_p, u_{0,p}, \ldots, u_{N,p}) \)
13: Compute motion from \( q \) to \( p' \)
14: for \( i \in O \) do
15: \( E_q \leftarrow E_{off} \)
16: return \( E_{off} \)

**C. Motion Smoothing**

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

We now present a smoothing algorithm based on the shortcut approach that takes as input a path in \( G \), here \( \pi_{ECF}(p_s, p_g) \), between start and goal configurations \( p_s, p_g \). This path is expressed as a sequence of edges in \( E_{CF} \). Thus, \( \pi_{ECF}(p_s, p_g) = \{ (i_r, i_{r+1}), r = 1, \ldots, m - 1 \} \) for some \( m \in \mathbb{N}_{\geq 2} \) where \( (i_r, i_{r+1}) \in E_{CF} \) for all \( r = 1, \ldots, m - 1 \) and \( i_1 = p_s, i_m = p_g \). Let \( C_{\pi} \) denote the set of all configurations along motions from \( i_r \) to \( i_{r+1} \) for all \( (i_r, i_{r+1}) \in \pi_{ECF}(p_s, p_g) \). Algorithm 2 summarizes the proposed approach.

Algorithm 3 takes as input a collision-free, dynamically feasible path \( \pi_{ECF}^{i_r} \) between start and goal configurations – such as that returned by PrAC – as well as the set of all configurations \( C_{\pi} \), obstacles \( X_{obs} \), cost function of motions \( c \), and a non-negative natural number \( n \). It also takes a function

---

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

**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 \( p \) results in a loop as \( \text{to} \). (c) Instead, \( q \) is replaced with the neighboring vertex \( p' \) and a motion from \( q \) to \( p' \) is computed.
Algorithm 2 Smoothing Lattice Motion

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

\[ \Psi : \mathbb{R} \rightarrow \mathbb{R} \] 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 $\pi^E$. For each pair $(i_u, i_v)$ of configurations with $i_v$ appearing after $i_u$ in $\pi^E$, 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 $p_s$ and $p_g$, respectively. This motivates the following observation and Theorem:

Observation III.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 III.5. Let $\pi^E$ be the input path to Algorithm 2 between configurations $p_s$, $p_g$ with cost $c(\pi^E)$. Let $\pi^E_2$ be the path returned by Algorithm 2 with cost $c(\pi^E_2)$. Then $c(\pi^E_2) \leq c(\pi^E)$. Moreover, if $n = 0$, this algorithm runs in time quadratic in the number of vertices along $\pi^E$.

Proof. By Observation III.4 Algorithm 2 constructs a DAG containing configurations $p_s, p_g$ as vertices. It also solves the minimum path cost problem on this DAG. Observe further that by construction of the DAG vertex set in Line 3, all configurations along the original path $\pi^E$ are in $V$. Thus $\pi^E_2$ is an available solution to the minimum path problem on the DAG. This proves that the minimum cost path can do no worse than $c(\pi^E_2)$. If $n = 0$, then $V$ is the set of endpoints of edges in $\pi^E$. Therefore, $V \subseteq L$. Because all motions between lattice vertices have been pre-computed in $E$, 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 $\pi^E_2$.

IV. Results

We verify our proposed technique against two techniques in two common navigational settings: Hybrid A* [27], and CG [30]. 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 all 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.

![Fig. 5: Comparison of smoothing algorithms for same input path](image)
A. Memory

The motions used in this section are \( G3 \) curves which take the form CCC, or CSC where “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 [43]. Thus 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 a common technique: Hybrid A* [27] in a parking lot scenario. Though Hybrid A* is not a new algorithm, more recent state of the art approaches use Hybrid A* to plan an initial motion which is then refined (e.g. [29]). We are therefore motivated to compare the run-time and path quality of the approach proposed in this paper to Hybrid A* whose run-time is a lower bound of all state of the art algorithms using it as a sub-routine. In [43], we present a method of computing motions between start and goal configurations in the configurations space \( \mathcal{X} = \mathbb{R}^2 \times [0,2\pi) \times \mathbb{R} \). Here, configurations take the form \( (x,y,\theta,\kappa,\sigma,\rho) \) where \( (x,y,\theta) \) represent the planar coordinates and heading of a vehicle, \( \kappa \) the curvature, \( \sigma \) the curvature rate (defined as \( d\kappa/ds \) for arc-length \( s ) \), and \( \rho \) the second derivative of curvature with respect to arc-length. We generate motions with continuously differentiable curvature profiles assuming that \( \kappa,\sigma,\rho \) 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 [28]. In particular, minimizing the integral of the squared jerk is often used a cost function in autonomous driving [30]. Since this value varies with \( \sigma \), keeping \( \sigma \) 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 [44], [45] 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 \( \mathcal{X} = \mathbb{R}^2 \times [0,2\pi) \times \mathbb{R} \) with configurations \( (x,y,\theta,\kappa,\sigma,\rho) \). Motion primitives were generated using the MILP in (7) for a \( 15 \times 20 \) square lattice with 16 headings and 3 curvatures, and a value of \( t = 1.1 \) (10% error from optimal). To account for off-lattice start-goal pairs, we used a higher-fidelity lattice with 64 headings and 30 curvatures. Lattice vertex values of \( \sigma,\rho \) were set to 0. This results in a start set \( S \) 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 [43] with bounds on \( \kappa,\sigma,\rho \):

\[
\kappa_{\text{max}} = 0.1982m^{-1}, \quad \sigma_{\text{max}} = 0.1868m^{-2} \quad \rho_{\text{max}} = 0.3905m^{-3},
\]

which were deemed comfortable for a user [44], particularly at low speeds typical of parking lots. The spacing of the lattice \( x,y \)-values was chosen to be \( r_{\text{min}}/4 \) for a minimum turning radius \( r_{\text{min}} = 1/\kappa_{\text{max}} \). Finally, if the cost of the motion from \( s \in S \) to a vertex \( j \) was larger than 1.2 times the Euclidean distance from \( s \) to \( j \) for all \( s \in S \), 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 \( s = (0,0,0,\kappa_{\text{max}}) \) to \( j = (r_{\text{min}},r_{\text{min}},\pi/2,\kappa_{\text{max}}) \) is a quarter circular arc of radius \( r_{\text{min}} \). 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 \( s \in S \), and a lattice vertex \( j \in L - S \). As such, they are for forward motion only. To add reverse motion primitives to the control set \( E \), for each \( s \in S \), we applied the forward primitives to \( S \) 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 \( (x,y,\theta,\kappa) \) that minimizes arc-length. This is to keep the branching factor of an online search low. Finally, to each \( s = (0,0,0,\kappa) \in S \) we add three primitives: \( (0,0,\theta,\pm\kappa,\kappa) \), \( (0,0,\theta,0) \) 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 Hybrid A* using an identical collision checking algorithm, and using the same heuristic (that proposed in [27]). Though the motions may appear similar, they are actually quite different. This difference is illustrated in Figure 7 which shows the heading \( \theta \) of the vehicle along the motion proposed herein vs that of Hybrid A*. Heading profiles for the other scenarios are not included for brevity, though results are similar. 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. These three metrics are expressed as ratios of the value obtained using the methods of [27] to those of the proposed. The results are summarized in Table 4. Observe that because these techniques are deterministic, no standard deviations are presented. The major difference between the two approaches can be seen in the IS Jerk Reduction. That is, the ratio of the IS Jerk using the methods of [27] to those of the proposed. This is because the motion primitives we employ are each \( G3 \) curves with curvature rates bounded by what is known to be comfortable. Observe that the value of IS Jerk obtained using our approach is up to 16 times less than that of [44]. In fact, using a Hybrid A* 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 Hybrid A*. Further, though Dubins’ paths (which are employed by Hybrid A*) take on average two orders of magnitude less time to compute than G3 curves, the runtime performance of our method often exceeds that of Hybrid A*. 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 [30]. Moreover, the methods in [30] do not account for reverse motion, and also assume that a set of way-points between start and goal configurations is known.

The only scenario in which Hybrid A* produces a motion in less time than the proposed method is Scenario (c) in which Hybrid A* produced a path with no reverse motion (which accounts for the speedup). However, in order to produce this motion, the curvature of the motion must change instantaneously multiple times 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 Hybrid A* 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)) which is illustrated in Figure 8. Though the motion computed with Hybrid A* may appear simpler, it requires a curvature rate that is 16.7 times larger than what is considered comfortable. It should also 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, Hybrid A* failed to produce a path in the allotted time.

A complex parking lot navigation scenario using the motion primitives from this section can be found in Figure 1.

### 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.

In addition to developing G3 paths, our work in [43] also details a method with which a trajectory with configurations \((x, y, \theta, \kappa, \sigma, \rho, v, a, \beta)\) may be computed. Here, \(v, a, \beta\) represent velocity and longitudinal acceleration, and longitudinal jerk respectively. The approach is to compute profiles of \(\rho\) and \(\beta\) that result in a trajectory that minimizes a user-specified cost function. This cost function is 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 \(\rho\)) and velocity profile (tuned by \(\beta\)) are optimized simultaneously, keeping path planning in-loop during the optimization. As in the previous set of examples, computing trajectories via the methods outlined in [43] require orders of magnitude more time than simple Dubins’ paths.

However, pre-computing a set of motion primitives where each motion is itself computed using the methods of [43]
ensures that every motion used in PrAC is optimal for the user. Moreover, because we include velocity in our configurations – and therefore in our 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 \(-r_{\text{max}}, r_{\text{max}}\). Finally, five evenly spaced velocities were sampled between 15 and 20 km/hr. A value of \( n = 0 \) was used in the smoothing Algorithm (2).

2) Scenario Results: The highway scenario was chosen to closely resemble the roadway driving scenario in [30]. The results of this scenario can be found in Figure 9, while performance analysis is summarized in Table II. The metrics used to measure performance are the arc-length of the proposed motion, the smoothness cost of the motion, the maximum curvature obtained over the motion, and a runtime speedup normalized to Hybrid A* (HA*). The final column of the Table indicates weather 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 (Smoothness1, Smoothness2) where

\[
\text{Smoothness}_1 = \sum_{i=1}^{N-1} |\Delta x_{i+1} - \Delta x_i|^2,
\]

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

The first measure is used in [27], where \( N \) configurations are sampled along a motion, with \( \Delta x_i \) denoting the vector of \( x,y \)-components of the \( i^{th} \) configuration, and \( \Delta x_i = x_i - x_{i-1} \). The second measure is used in [30]. The first two methods appearing in the Table II are computed directly from the motions in Figure 9 while the second two come from [30] for an identical experiment. Here, CG refers to the method proposed in [30], while HA2* refers to the implementation of Hybrid A* as it appears in [30].

The authors of [30] report an average runtime speedup of 4.5 times as compared to Hybrid A* for the path planning phase (without speed profile). On average, PrAC computed a full motion, including a speed profile 4.7 times faster than the time required for Hybrid A* to compute a path. Furthermore, the use of PrAC with G3 motion primitives significantly reduced the smoothness cost in both of its possible definitions.

| Method | Length (m) | Smooth Cost | Max Curvature (m⁻¹) | Runtime Speedup | Velocity |
|--------|------------|-------------|---------------------|------------------|----------|
| HA*    | 65.0       | (1.35, 0.77)| 0.198              | 1                | No       |
| Proposed | 64.6     | (0.17, 0.28)| 0.158              | 4.7              | Yes      |

|          |            |             |                     |                  |          |
| CG       | 65.8       | (0.44)      | 0.189              | 4.5              | No       |
| HA2*     | 65.6       | (0.88)      | 0.196              | 1                | No       |

TABLE II: Road navigation results: HA* and proposed shown above. CG and HA2* from [30] Table 3 for similar motion planning problem.

V. DISCUSSION & CONCLUSION

In this work, 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. We also present an A*-based algorithm which uses these primitives to compute motions between configurations. Finally, we develop a post-processing smoothing algorithm to remove excessive oscillations from the resulting motions.

The results of the previous section illustrate the effectiveness of the proposed technique. Indeed, feasible, smooth motions were computed between start and goal locations in both parking lot and highway settings. By adding reverse motion primitives, complex problems like navigating an obstacle-rich parking lot, or parallel parking were solved. Moreover, 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. These are left for future work.

REFERENCES

[1] B. Paden, M. Čap, 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.
[2] 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.
[3] O. Andersson, O. Ljungqvist, M. Tiger, D. Axelhll, and F. Heintz, “Receding-horizon lattice-based motion planning with dynamic obstacle avoidance,” in 2018 IEEE CDC, pp. 4467–4474.

[4] D. González, J. Pérez, V. Milánés, 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] A. Rusu, S. Moreno, Y. Watanabe, M. Rognant, and M. Devy, “Satellite lattice generation and nonholonomic path planning for a planetary exploration rover,” in 55th International Astronautical Congress 2014 (IAC), vol. 2, p. 953.

[13] 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. 30, no. 7, pp. 846–894, 2011.

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] M. Ruhl and R. Siegwart, “On the design of deformable input-state lattice graphs,” in 2010 IEEE ICRA, pp. 3071–3077.

[24] M. McNaughton, C. Urmon, J. M. Dolan, and J.-W. Lee, “Motion planning for autonomous driving with a conformal spatiotemporal lattice,” in 2011 IEEE ICRA, pp. 4889–4895.

[25] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, “Motion primitives-based path planning for fast and agile exploration using aerial robots,” in 2020 IEEE ICRA, pp. 179–185.

[26] 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.

[27] J. Levinson, J. Askell, 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.

[28] X. Zhang, A. Linger, and F. Borrelli, “Optimization-based collision avoidance,” IEEE Transactions on Control Systems Technology, vol. 29, no. 3, pp. 972–983, 2021.

[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. Paschos, 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, “Disperatio: Optimal sampling for safe deterministic motion planning,” IEEE RA-L, vol. 5, no. 2, pp. 362–368, 2019.

[33] K. Bergman, O. Ljungqvist, and D. Axelhll, “Improved optimization of motion primitives for motion planning in state lattices,” in 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 2307–2314.

[34] D. Peleg and A. A. Schaffer, “Graph spanners,” Journal of graph theory, vol. 13, no. 1, pp. 99–116, 1989.

[35] D. Peleg and A. A. Schaffer, “Graph spanners,” Journal of graph theory, vol. 13, no. 1, pp. 99–116, 1989.

[36] A. Botros, N. Wilde, and S. L. Smith, “Learning control sets for lattice planners from user preferences,” in WAFR, Springer, 2020, pp. 381–397.

[37] 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.

[38] A. Botros, “Lattice-based motion planning with optimal motion primitives,” 2021.

[39] B. Korte, J. Vygen, B. Korte, and J. Vygen, Combinatorial optimization, 6th ed. Springer, 2018.

[40] J. Desrosiers, O. 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.

[41] I. Pohl, “Heuristic search viewed as path finding in a graph,” Artificial intelligence, vol. 1, no. 3–4, pp. 193–204, 1970.

[42] 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.

[43] A. Botros and S. L. Smith, “Tunable trajectory planner using g3 curves,” IEEE Transactions on Intelligent Vehicles, 2022.

[44] H. Banzhaf, N. Berinpanathan, D. Nienhuser, and J. M. Zollner, “From robots to software: A decade of research at the University of Waterloo,” in 2021 IEEE/RSJ IROS, pp. 3280–3288.

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 constructing 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.