Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Temporal Reasoning with Kinodynamic Networks

Han Zhang[1], **Neelesh Tiruviluamala**[2], **Sven Koenig**[1], **T. K. Satish Kumar**[1]
[1] Department of Computer Science, University of Southern California
[2] Department of Mathematics, University of Southern California
{zhan645, tiruvilu, skoenig}@usc.edu, tkskwork@gmail.com

Abstract

Temporal reasoning is central to Artificial Intelligence (AI) and many of its applications. However, the existing algorithmic frameworks for temporal reasoning are not expressive enough to be applicable to robots with complex kinodynamic constraints typically described using differential equations. For example, while minimum and maximum velocity constraints can be encoded in Simple Temporal Networks (STNs), higher-order kinodynamic constraints cannot be represented in existing frameworks. In this paper, we present a novel framework for temporal reasoning called Kinodynamic Networks (KDNs). KDNs combine elements of existing temporal reasoning frameworks with the idea of Bernstein polynomials. The velocity profiles of robots are represented using Bernstein polynomials; and dynamic constraints on these velocity profiles can be converted to linear constraints on the to-be-determined coefficients of their Bernstein polynomials. We study KDNs for their attractive theoretical properties and apply them to the Multi-Agent Path Finding (MAPF) problem with higher-order kinodynamic constraints. We show that our approach is not only scalable but also yields smooth velocity profiles for all robots that can be executed by their controllers.

Introduction

Many problems of interest in Artificial Intelligence (AI) and robotics require rich representations of time and efficient algorithms for reasoning with them. For example, in AI, temporal reasoning has applications in autonomous space exploration (Knight et al. 2001), domestic activity management (Pecora and Cirillo 2009), job scheduling on servers (Ji, He, and Cheng 2007), human-robot interaction (Wilcox, Nikolaidis, and Shah 2013) and airport surface operations (Morris et al. 2016).

Many formalisms, with varying degrees of complexity and expressiveness, are used for reasoning with metric time. Simple Temporal Problems (STPs) are on the lower end of the scale with respect to complexity. Although their expressiveness is limited compared to other formalisms, STPs are widely used, as they can be solved in polynomial time using shortest path computations (Dechter, Meiri, and Pearl 1991). Disjunctive Temporal Problems (DTPs) (Stergiou and Koubarakis 2000) are significantly more expressive than STPs, as they can encode disjunctive constraints. They can be used to model a large variety of real-world problems, such as scheduling problems with resource contentions and positive and negative time lags (Brucker, Hilbig, and Hurink 1999). However, DTPs are NP-hard to solve in general. Simple Temporal Problems with Preferences (STPPs) (Khatib et al. 2001) are STPs with additional soft binary constraints, interpreted as local preference functions. An optimal solution for an STPP is one that satisfies all simple temporal constraints and optimizes the global preference function, interpreted as a combination of local preference functions. STPPs are also NP-hard to solve in general, although certain subclasses of them can be solved in polynomial time. Simple Temporal Problems with Uncertainties (STPUs) (Vidal 1999) are extensions of STPs that are defined by a set of timepoints $\mathcal{V} = \mathcal{V}_C \cup \mathcal{V}_U$. $\mathcal{V}_C$ and $\mathcal{V}_U$ are the sets of controllable and uncontrollable timepoints, respectively. STPUs also have a set of contingent edges that have uncertain duration. Controllability, instead of consistency, is the main characteristic feature of STPUs.

Despite the many successful applications of the abovementioned temporal reasoning frameworks in various domains, they are not expressive enough to be directly applicable to complex robots operating in the real world. This is so because most real-world robots operate under complex kinodynamic constraints typically described using differential equations. For example, suppose a robot travels from point A to point B along a straight line. Let $v_{\min}$ and $v_{\max}$ be the minimum and maximum velocities constraining the motion of the robot. This straightforward dynamic constraint can be encoded as a *simple temporal constraint* $\tau(X_B) - \tau(X_A) \in [L/v_{\max}, L/v_{\min}]$ in the STP framework. Here, $\tau(X_A)$ and $\tau(X_B)$ represent the times at which the robot is at A and B, respectively, and L is the distance between A and B. However, the STP framework is incapable of representing any higher-order dynamic constraints such as minimum and maximum accelerations or minimum and maximum jerks. Generalizations of STPs, such as DTPs, STPPs and STPUs, are also ineffective because they focus on generalizability in ways unrelated to the consideration of infinitesimal timescales needed for properly describing the kinodynamic constraints on robots. Such kinodynamic constraints are most naturally specified using differential equations in continuous time. The existing temporal reason-

ing frameworks mentioned above are incapable of reasoning with differential equations.

In this paper, we present a novel framework for temporal reasoning called Kinodynamic Networks (KDNs). KDNs combine elements of existing temporal reasoning frameworks with the idea of Bernstein polynomials. Bernstein polynomials have a number of useful mathematical properties. They can uniformly approximate any continuous function; and their derivatives are also Bernstein polynomials (Lorentz 1986). Furthermore, they are closely related to Bézier curves, which are defined by a finite number of control points. A Bézier curve lies entirely within the convex hull of its control points. Because of their mathematical properties, Bernstein polynomials and Bézier curves have been widely used in many applications, including Computer Graphics (Mortenson 1999), Computer-Aided Design (Farin 1992), path planning (Choi, Curry, and Elkaim 2008) and trajectory planning (Tang and Kumar 2016).

In our work, each robot's *distance function*, that is, the distance traveled by the robot on a given path, is represented using Bernstein polynomials with to-be-determined coefficients. Using the mathematical properties of Bernstein polynomials, any dynamic constraints on the capabilities of a single robot, such as minimum and maximum velocities, minimum and maximum accelerations, minimum and maximum jerks, or any other higher-order constraints, can be converted to linear constraints on the coefficients of the Bernstein polynomials representing the robot's distance function. In the case of multiple robots coordinating with each other, any kinematic constraints between them, such as having to maintain a safety distance between each other, can be converted to simple temporal constraints between "landmark" events (Hönig et al. 2016). Overall, KDNs allow us to represent both kinematic and dynamic constraints suitable for problems in robotics and multi-agent systems.

We study KDNs for their attractive theoretical properties and apply them to the Multi-Agent Path Finding (MAPF) problem (Hönig et al. 2016; Ma and Koenig 2017) with higher-order kinodynamic constraints. The MAPF problem involves a team of robots that are required to plan collision-free paths from their start locations to their goal locations in a common environment. Examples include autonomous aircraft towing vehicles, automated warehouse systems, office robots, and game characters in video games. We show that our approach is not only scalable but also yields smooth velocity profiles for all robots that can be executed by their controllers. Overall, our paper is about the general theory of KDNs as a powerful temporal reasoning framework with a chosen application in the MAPF domain. Of course, KDNs have many applications outside of the MAPF domain as well.

Bernstein Basis Polynomials and Bézier Curves

In mathematics, Bernstein basis polynomials of degree n are defined to be

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i \in \{0, 1 \dots n\},$$

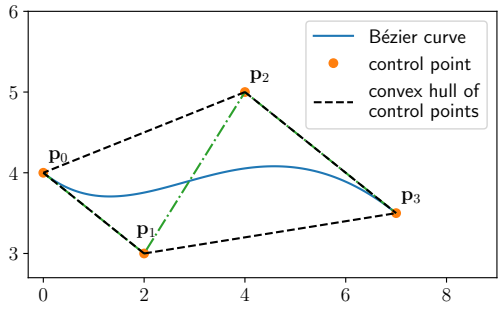


Figure 1: Illustrates an important property of Bézier curves: A Bézier curve is enclosed entirely within the convex hull of its control points. Here, the Bézier curve has 4 control points in a 2-dimensional space.

where $\binom{n}{i}$ is the binomial coefficient equal to $\frac{n!}{i!(n-i)!}$.

A k-dimensional Bézier curve of degree n is of the form

$$\mathbf{B}(t) = \sum_{i=0}^{n} \mathbf{p}_i B_{i,n}(t), \quad t \in [0, 1],$$

where $P = \{\mathbf{p}_0, \mathbf{p}_1 \dots \mathbf{p}_n\}$ are $n+1$ k-dimensional *control points*. Therefore, it is a curve parameterized by t and interpretable as a linear combination of the Bernstein basis polynomials of degree n. The coefficients of the linear combination are the $n+1$ k-dimensional control points.

A Bernstein polynomial $B(t)$ of degree n is a 1-dimensional Bézier curve of degree n. Therefore, it is a linear combination of the Bernstein basis polynomials of degree n. The coefficients of the linear combination are $n+1$ real numbers acting as 1-dimensional control points. These control points are also referred to as the Bernstein coefficients.

Bernstein polynomials and Bézier curves have a number of useful mathematical properties. For example, the Weierstrass Approximation Theorem (Lorentz 1986) establishes that any continuous real-valued function defined on the real interval $[0, 1]$ can be uniformly approximated by Bernstein polynomials.

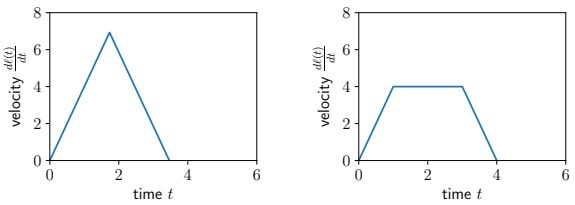
Two other useful properties of Bernstein polynomials and Bézier curves are with respect to their derivatives and their control points.

The derivative of a Bézier curve $\mathbf{B}(t)$ of degree n is a Bézier curve of degree $n-1$. In particular,

$$\frac{d\mathbf{B}(t)}{dt} = \sum_{i=0}^{n-1} \mathbf{p}'_i B_{i,n-1}(t),$$

where the control point $\mathbf{p}'_i = n(\mathbf{p}_{i+1} - \mathbf{p}_i)$, $i \in \{0, 1 \dots n-1\}$.

A Bézier curve $\mathbf{B}(t)$ is bounded by the convex hull of its control points P for $t \in [0, 1]$, as shown in Figure 1. Intuitively, this is so because, for any given value of $t \in [0, 1]$, (a) $B_{i,n}(t) \geq 0$ for $i \in \{0, 1 \dots n\}$ and (b) $\sum_{i=0}^{n} B_{i,n}(t) = 1$.



(a) (b)

Figure 2: Illustrates two possible scenarios for the optimal velocity profile in our motivating example. In both (a) and (b), $L = 12\,\mathrm{m}$, $a_{\max} = 4\,\mathrm{m/s^2}$ and $a_{\min} = -4\,\mathrm{m/s^2}$. In (a), $v_{\max} = 8\,\mathrm{m/s}$, and in (b), $v_{\max} = 4\,\mathrm{m/s}$.

Therefore, $\mathbf{B}(t)$ for $t \in [0,1]$ is interpretable as a non-negative linear combination of its control points, necessitating its presence in the convex hull. In particular, $\mathbf{B}(0) = \mathbf{p}_0$ and $\mathbf{B}(1) = \mathbf{p}_n$. In the case of Bernstein polynomials, the control points are 1-dimensional real numbers, and $B(t)$ lies entirely within $[\inf(P), \sup(P)]$.

A Formalization of KDNs

In this section, we formalize KDNs and the task of solving them. We define a KDN as a generalization of the Simple Temporal Network (STN), and the task of solving a KDN as a generalization of the task of solving an STN, that is, the STP. Of course, further generalizations of KDNs are also possible by incorporating preferences and uncertainties analogous to STPPs and STPUs, respectively.

A KDN is characterized by a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \{X_0, X_1 \ldots X_N\}$ is the set of vertices and $\mathcal{E}$ is the set of edges. Each $X_i \in \mathcal{V}$ represents an event, $\tau(X_i)$ represents the to-be-determined execution time of X_i, X_0 represents the "beginning of time", and $\tau(X_0)$ is conventionally set to 0. Each edge $e = \langle X_i, X_j \rangle \in \mathcal{E}$, annotated with the interval $[LB(e), UB(e)]$, is a simple temporal constraint between X_i and X_j, indicating that X_j must be scheduled between $LB(e)$ and $UB(e)$ time units after X_i, that is, $\tau(X_j) - \tau(X_i) \in [LB(e), UB(e)]$. A subset of the edges $\mathcal{E}_M \subseteq \mathcal{E}$ contains special edges called *motion edges*. A motion edge $e = \langle X_i, X_j \rangle \in \mathcal{E}_M$ corresponds to the traversal of a physical distance from location ℓ_s^e to location ℓ_f^e by robot R^e. It is annotated with a set of dynamic constraints $D(e)$ imposed on R^e during its traversal of the distance from ℓ_s^e to ℓ_f^e. The task of solving a KDN is to simultaneously find τ and a velocity profile for robot R^e, for each $e \in \mathcal{E}_M$, that satisfies the dynamic constraints $D(e)$. We are also required to correctly report so whenever no such τ and velocity profiles can jointly exist.

Dynamic Constraints on a Single Motion Edge

Although the formalization of KDNs is akin to that of STNs, KDNs are significantly more complex than STNs. Even a single motion edge can introduce dynamic constraints that are hard to solve. Consider the following simple motivating example. Suppose a robot is required to travel a distance L in a straight line between two points A and B. Suppose it is required to start at A and end at B with 0 velocities; and suppose it has maximum velocity $v_{\max} \geq 0$, maximum acceleration $a_{\max} \geq 0$ and minimum acceleration $a_{\min} \leq 0$.[1] The goal is to minimize the traversal time T. Intuitively, the optimal solution is to start with maximum acceleration $a_{\max}$ and stop with maximum deceleration $|a_{\min}|$. In between, the robot should cap off at the maximum velocity $v_{\max}$. The two possible scenarios are illustrated in Figure 2.

Despite the fact that the above example problem can be solved by humans intuitively, it is not easy for a computer to do so. Without the physical interpretation of the mathematical problem, solving it is not straightforward even from the perspective of techniques available in calculus. This is primarily so because of inequalities imposed on the derivatives of continuous functions. For example, if $\ell(t)$ is the distance function that represents the distance covered at time t starting from A, the above example problem stated completely mathematically is as follows (with $K = 2$, $c^0_{init} = 0$, $c^0_{final} = L$, $c^1_{init} = 0$, $c^1_{final} = 0$, $c^1_{high} = v_{\max}$, $c^2_{low} = a_{\min}$ and $c^2_{high} = a_{\max}$).

$$\begin{aligned} &\text{Find} \quad \ell(t) : [0,T] \to \mathbb{R} \text{ and minimum } T \\ &\text{s.t.} \quad \left.\tfrac{d^k \ell(t)}{dt^k}\right|_{t=0} = c^k_{init}; \left.\tfrac{d^k \ell(t)}{dt^k}\right|_{t=T} = c^k_{final} \\ &\qquad \forall k \in \{0, 1 \ldots K\} \\ &\qquad c^k_{low} \leq \tfrac{d^k \ell(t)}{dt^k} \leq c^k_{high} \\ &\qquad \forall t \in [0,T], \forall k \in \{0, 1 \ldots K\}. \end{aligned} \tag{1}$$

We note that the above formalization uses c^k_{init}, c^k_{final}, c^k_{low} and c^k_{high} for all $k \in \{0, 1 \ldots K\}$ for simplicity of abstract analysis. In general, not all of them need to be specified.

We refer to such mathematical formulations as *kinodynamic differential programs* (KDDPs), analogous to linear programs (LPs), mixed integer linear programs (MILPs) and convex programs. KDDPs are hard to solve both analytically and computationally without a physical interpretation. Even with a physical interpretation, they are not amenable to human intuition when higher-order dynamic constraints are imposed on the robot. In fact, even for a given value of T, the KDDP feasibility problem is not easy.

We now present a novel approach for solving KDDPs computationally. The advantages of solving KDDPs computationally are that (a) we can solve complex dynamic constraints associated with a single motion edge and (b) we can, combined with other procedures, solve an entire KDN with many motion edges and temporal constraints. Neither (a) nor (b) is amenable to physical interpretations or human intuition with increasing complexity.

Our approach uses four main ideas for efficiently solving KDDPs. First, we represent $\ell(t)$ using a Bernstein polynomial with to-be-determined control points. With a large enough number of control points, the Weierstrass Approximation Theorem (Lorentz 1986) guarantees that a Bernstein

[1] Therefore, the maximum deceleration is $|a_{\min}| \geq 0$.

polynomial can approximate the required $\ell(t)$ to any degree of accuracy. Second, any k^{th}-order derivative $\ell^{(k)}(t) = \frac{d^k \ell(t)}{dt^k}$ is also a Bernstein polynomial since Bernstein polynomials are closed under the operation of differentiation. Third, inequality bounds on any function represented as a Bernstein polynomial can be enforced by imposing linear constraints on each of its control points. While these three ideas are sufficient to address the feasibility problem of finding $\ell(t)$ for a given T, a fourth idea is required for minimizing T. This idea invokes binary search enabled by the Single Interval Theorem, discussed and formally proved later.

The Feasibility Problem

For the feasibility problem, we are given the value of T and are required to find the distance function $\ell(t)$. Suppose we represent $\ell(t)$ as a *scaled* Bernstein polynomial $B^T(t)$ of degree n with to-be-determined control points $P = \{p_0, p_1 \dots p_n\}$. For mathematical convenience, we would like to interpret the parameter t in $B^T(t)$ as representing time. In order to do this successfully, we need to scale the interval $[0, 1]$ to the interval $[0, T]$ since any Bernstein polynomial is only defined in the interval $[0, 1]$. The scaled Bernstein basis polynomials are now of the form

$$B^T_{i,n}(t) = \binom{n}{i} \left(\frac{t}{T}\right)^i \left(\frac{T-t}{T}\right)^{n-i}, \quad i \in \{0, 1 \dots n\}.$$

Therefore, the distance function $\ell(t)$ approximated using a scaled Bernstein polynomial is now of the form

$$B^T(t) = \sum_{i=0}^{n} p_i B^T_{i,n}(t), \quad t \in [0, T].$$

The scaled Bernstein polynomials defined in the interval $[0, T]$ have properties that are very similar to those of the Bernstein polynomials defined in the interval $[0, 1]$. In particular, $B^T(t)$ with a sufficiently large number of control points can be used to approximate any continuous function to any degree of accuracy in the interval $[0, T]$. Similarly, $\frac{dB^T(t)}{dt}$ is another scaled Bernstein polynomial of degree $n-1$ with control points $\{\frac{n}{T}(p_1 - p_0), \frac{n}{T}(p_2 - p_1) \dots \frac{n}{T}(p_n - p_{n-1})\}$. Moreover, $B^T(t)$ is also bounded by the convex hull of its control points P for $t \in [0, T]$, and $B^T(0) = p_0$ and $B^T(T) = p_n$. Following the same arguments, $\frac{d^k B^T(t)}{dt^k}$ is a scaled Bernstein polynomial of degree $n - k$ with control points $\frac{n!}{(n-k)!T^k}\left(\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{k+i-j}\right)$ for $i \in \{0, 1 \dots n-k\}$.

We now consider the KDDP in Equation 1. The first two constraints translate to $\frac{n!}{(n-k)!T^k}\left(\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{k-j}\right) = c^k_{init}$ and $\frac{n!}{(n-k)!T^k}\left(\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{n-j}\right) = c^k_{final}$, respectively, for $k \in \{0, 1 \dots K\}$. The next two constraints translate to the linear inequalities $c^k_{low} \le \frac{n!}{(n-k)!T^k}\left(\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{k+i-j}\right) \le c^k_{high}$ for $i \in \{0, 1 \dots n-k\}$ and $k \in \{0, 1 \dots K\}$. This is so because of the convexity property of Bézier curves, that is, all control points of $\frac{d^k B^T(t)}{dt^k}$ constrained to be in the interval $[c^k_{low}, c^k_{high}]$ ensures that the entire curve is also within that interval. Overall, the KDDP in Equation 1 is now translated to an LP on the to-be-determined control points of $\ell(t)$ represented as a Bernstein polynomial $B^T(t)$:

$$\begin{aligned}
&\text{Find} \quad \{p_0, p_1 \dots p_n\} \\
&\text{s.t.} \quad \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{k-j}\right) = c^k_{init} \\
&\qquad \forall k \in \{0, 1 \dots K\} \\
&\qquad \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{n-j}\right) = c^k_{final} \\
&\qquad \forall k \in \{0, 1 \dots K\} \\
&\qquad \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{k+i-j}\right) \ge c^k_{low} \\
&\qquad \forall i \in \{0, 1 \dots n-k\}, \forall k \in \{0, 1 \dots K\} \\
&\qquad \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j \binom{k}{j} p_{k+i-j}\right) \le c^k_{high} \\
&\qquad \forall i \in \{0, 1 \dots n-k\}, \forall k \in \{0, 1 \dots K\}.
\end{aligned} \tag{2}$$

An LP can be solved very efficiently in polynomial time, therefore allowing us to decode $B^T(t)$ and use it as an approximation to the required $\ell(t)$. At first glance, the above reformulation seems to have the drawback that it is possible for a Bézier curve to satisfy the dynamic constraints without the control points necessarily satisfying the corresponding linear inequalities. However, this drawback is obviated as n gets larger. This is so because the Weierstrass Approximation Theorem (Lorentz 1986) guarantees that any feasible solution can be approximated by a Bézier curve to any degree of accuracy as n gets larger.

The Single Interval Theorem

In the previous subsection, we showed how to solve the feasibility version of a KDDP. As mentioned before, solving the optimization version, that is, minimizing T, is more complicated and requires a binary search procedure. In general, a binary search procedure relies on the exploitation of a recognized monotonic property of the domain, for example, a sorted order of its elements, and an indicator function that indicates how to halve the interval of search in each iteration. In our case, the corresponding important property of KDDPs that we will recognize and exploit is encapsulated in the Single Interval Theorem, stated and proved below.

In simple words, the Single Interval Theorem states that, if it is possible for a robot to traverse the distance L from A to B in time T by meeting all constraints of the KDDP and if T can have two possible values T_1 and $T_2 > T_1$, then it is also feasible for the robot to do the traversal in time T_3 for any $T_3 \in [T_1, T_2]$. While this is a credible statement from the perspective of human experience about moving objects, proving it formally is not straightforward. Therefore, we present a rigorous formal proof in the appendix of this paper.

Theorem 1. [Single Interval Theorem] *Suppose that* $\langle \tilde{f}(s), T_1 \rangle$ *and* $\langle \tilde{g}(s), T_2 \rangle$, $T_1 < T_2$, *are feasible for a KDDP*

Algorithm 1: SOLVE-KDDP: A binary search algorithm for solving KDDPs of the form in Equation 1 using our Bézier curve approach. M is a sufficiently large number or any other upper bound on T^*. thr is a user-specified threshold parameter.

Input : A KDDP $\mathcal{L}$ of the form in Equation 1.
Output: Minimum feasible T^* and function $\ell^*(t) : [0, T^*] \rightarrow \mathbb{R}$, if they exist.

Function `ApproximateGradient(f, x)`
  Let Δx be a sufficiently small positive number;
  return $\frac{f(x+\Delta x)-f(x)}{\Delta x}$;
end
Let $\mathcal{P}(T)$ denote the LP in Equation 2 derived from the Bézier approximation of $\mathcal{L}$ for T;
Let $\mathcal{P}'(T)$ denote the modified version of $\mathcal{P}(T)$ in Equation 3 with δ as the slack variable;
Let $slack(T)$ denote the value of δ^* in $\mathcal{P}'(T)$;
Initialize $lb \leftarrow 0$, $ub \leftarrow M$;
Initialize $T^* \leftarrow Null$;
while $ub - lb > thr$ **do**
  $mid \leftarrow (lb + ub)/2$;
  if $slack(mid) = 0$ **then**
    $ub \leftarrow mid$;
    $T^* \leftarrow mid$;
  else
    if `ApproximateGradient`$(slack, mid) > 0$ **then**
      $ub \leftarrow mid$;
    else
      $lb \leftarrow mid$;
    end
  end
end
if $T^* = Null$ **then**
  return *No Solution*;
else
  return T^* *and* $\ell^*(t)$ *constructed from* $\mathcal{P}(T^*)$;
end

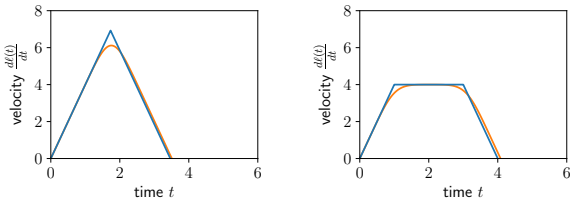
$\mathcal{L}$ of the form in Equation 1. Assume further that $\tilde{f}(s)$ and $\tilde{g}(s)$ have Lipschitz K^{th} derivatives with Lipschitz constant C. Then, for any $T_3 \in [T_1, T_2]$, there exists a function $\tilde{h}(s) : [0, T_3] \rightarrow \mathbb{R}$ such that $\langle \tilde{h}(s), T_3 \rangle$ is also feasible for $\mathcal{L}$ and the K^{th} derivative of $\tilde{h}(s)$ has Lipschitz constant C.

Proof. See the appendix. □

The Optimization Problem

Algorithm 1 presents the pseudocode of a binary search procedure for solving the optimization problem of finding T^*, that is, the minimum feasible value of T in Equation 1. It reduces the optimization problem to a series of feasibility problems for fixed values of T. This series is guided by a binary search on the current value of T in the outer loop that uses an intelligently chosen indicator for deciding whether $T^* \leq T$ or $T^* > T$. The failure of the binary search to find a feasible value of T is indicative of an infeasible KDDP.

From the Single Interval Theorem, we know that the feasible values of T are in some continuous interval $[T_1, T_2]$.



(a) (b)

Figure 3: Shows the result of applying Algorithm 1 to the running examples from Figure 2. The blue curves are the optimal velocity profiles derived from the physical interpretation. The orange curves are close approximations produced by our Bézier curve approach. In both cases, 40 control points are used for the approximation.

Therefore, $T^* = T_1$. In order to find T^* by binary search, for any given value of T, we need to be able to efficiently determine whether $T^* \leq T$ or $T^* > T$. We derive such an indicator from an auxiliary LP that introduces a slack variable $\delta \geq 0$ to the LP in Equation 2. We show that its optimal value δ^* has a distinctive behavior in each of the intervals $(0, T_1)$, $[T_1, T_2]$ and $(T_2, +\infty)$, hence establishing the correctness of the binary search procedure in Algorithm 1. The auxiliary LP is as follows.

$$
\begin{aligned}
\text{Min} \quad & \delta \\
\text{s.t.} \quad & \delta \geq 0 \\
& \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j\binom{k}{j}p_{k-j}\right) = c^k_{init} \\
& \forall k \in \{0, 1 \ldots K\} \\
& \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j\binom{k}{j}p_{n-j}\right) = c^k_{final} \\
& \forall k \in \{0, 1 \ldots K\} \\
& \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j\binom{k}{j}p_{k+i-j}\right) \geq c^k_{low} - \delta \\
& \forall i \in \{0, 1 \ldots n-k\}, \forall k \in \{0, 1 \ldots K\} \\
& \tfrac{n!}{(n-k)!T^k}\left(\textstyle\sum_{j=0}^{k}(-1)^j\binom{k}{j}p_{k+i-j}\right) \leq c^k_{high} + \delta \\
& \forall i \in \{0, 1 \ldots n-k\}, \forall k \in \{0, 1 \ldots K\}.
\end{aligned}
\tag{3}
$$

Theorem 2. *For a KDDP of the form in Equation 1, if the feasible interval of T is $[T_1, T_2]$, then δ^* in the corresponding LP of the form in Equation 3, for a sufficiently large n, is monotonically decreasing in the interval $(0, T_1)$, 0 in the interval $[T_1, T_2]$, and monotonically increasing in the interval $(T_2, +\infty)$.*

Proof. By the Weierstrass Approximation Theorem, a sufficiently large n allows us to use the formulations in Equations 1 and 2 interchangeably. Therefore, Equation 3 can also be used interchangeably with Equation 1 with the same slack. Furthermore, the LP in Equation 2 is equivalent to that

in Equation 3 when $\delta = 0$. Therefore, any feasible value of T in Equation 2 is also feasible in Equation 3 with $\delta = 0$. In fact, this is also the value of δ^* since $\delta \geq 0$. This proves that $\delta^* = 0$ in the interval $[T_1, T_2]$. We now prove that δ^* is monotonically increasing in the interval $(T_2, +\infty)$. The proof for it monotonically decreasing in the interval $(0, T_1)$ is analogous.

We first prove by contradiction that the minimum slack is monotonically non-decreasing in the interval $(T_2, +\infty)$. Suppose there exist τ and τ' such that $\tau' > \tau > T_2$ and $\delta^* > \delta'^*$, where δ^* and δ'^* are the minimum slacks required for τ and τ', respectively. Now consider Equation 1 with slack δ'^*. By definition, τ' is feasible for δ'^*. T_2 is also feasible for δ'^* since T_2 requires no slack at all. By the Single Interval Theorem, this implies that τ should also be feasible for δ'^* since $\tau \in [T_2, \tau']$. This contradicts the assumption that the minimum slack required for τ is $\delta^* > \delta'^*$.

We finally prove by contradiction that the minimum slack cannot be constant in an interval $[\tau, \tau']$ with $\tau' > \tau > T_2$. Suppose the minimum slack is $\delta^* > 0$ within the interval $[\tau, \tau']$. Without loss of generality, let τ be the minimum value of T for which the minimum slack is δ^*. Consider Equation 3 and a constraint in it that utilizes slack. From the first principles of calculus, an infinitesimal increase in slack for this constraint can only lead to an infinitesimal increase in the set of feasible values for any of the participating variables. Therefore, the set of feasible values of T for all constraints considered together increases only infinitesimally with an infinitesimal increase in the slack. Now consider the slack $\delta^* - \Delta\delta$ for an infinitesimal $\Delta\delta > 0$. The measurable non-infinitesimal interval $[\tau, \tau']$ is not feasible for $\delta^* - \Delta\delta$ but is feasible for $(\delta^* - \Delta\delta) + \Delta\delta$, leading to a contradiction. □

Figure 3 shows the result of applying Algorithm 1 on the running examples from Figure 2. The algorithm produces the minimum feasible time T^* as well as the required distance function $\ell^*(t)$, from which the velocity profile, acceleration profile and any other higher-order profile can be obtained. In fact, even for this simple example, the solution produced by Algorithm 1 is technically more accurate than the human-constructed solution since bounds on the maximum and minimum accelerations prohibit the velocity profiles from having sharp corners.

Our approach has many advantages compared to methods that try to approximate a function numerically by discretizing the timeline and finding a value for the function at each of the discrete timepoints. First, such methods do not produce analytical forms of the functions. They produce values only at discrete timepoints, requiring further interpolation, which often makes them unusable. Second, they need to determine a large number of unknowns depending on the granularity of the discretization. On the other hand, Bézier curves use significantly fewer control points, making them popular in such diverse areas as Computer Graphics (Mortenson 1999), Computer-Aided Design (Farin 1992), path planning (Choi, Curry, and Elkaim 2008) and trajectory planning (Tang and Kumar 2016). Third, they are not well suited for reasoning with high-order derivatives of functions since they need to represent the values of the function and all its higher-order derivatives at each of the discrete timepoints separately. Finally, they are also not well suited for facilitating indicators that can be used in an outer-loop binary search procedure often required for optimization problems.

Solving KDNs

In the previous section, we showed how to solve KDDPs efficiently using our Bézier curve approach. These methods reformulate KDDPs as regular LPs, which can be solved in polynomial time. However, KDDPs are associated with the dynamic constraints of a single motion edge in a KDN. In this section, we show how to solve entire KDNs efficiently.

At a high level, our approach for solving KDNs invokes the Single Interval Theorem and converts a KDDP with fixed boundary conditions to an induced simple temporal constraint. For a feasible KDDP associated with the dynamic constraints of any motion edge $e = \langle X_i, X_j \rangle$, we know that there exists a single feasible time interval $[T_1^e, T_2^e]$. In fact, Algorithm 1 not only can be used to find T_1^e but can also be adapted to find T_2^e with the same indicator function guiding the binary search in the outer loop. Therefore, for fixed boundary conditions at ℓ_s^e and ℓ_f^e, we have the induced simple temporal constraint $\tau(X_j) - \tau(X_i) \in [T_1^e, T_2^e]$. A KDN can now be solved by searching in the space of all combinations of boundary conditions at the landmarks defined to be $\cup_{e \in \mathcal{E}_M} \{\ell_s^e, \ell_f^e\}$. This can be done using a MILP solver.

To illustrate the working of our KDN solver, we choose examples from the MAPF domain. The MAPF problem involves a team of robots that are required to plan collision-free paths from their start locations to their goal locations in a common environment modeled as an undirected graph. The vertices of the graph represent locations, and the edges of the graph represent connections between locations that can be traversed in unit time. In any timestep, a robot can choose to wait at its current vertex or move to a neighboring vertex. No two robots can be in the same vertex at the same timestep; and no two robots can traverse an edge in opposite directions at the same timestep. The MAPF problem arises in many real-world domains, including autonomous aircraft towing vehicles, automated warehouse systems, office robots, and game characters in video games. A survey of MAPF research and its applications can be found in (Ma and Koenig 2017).

Despite the success of MAPF research, scalable MAPF solvers produce discrete solutions that prescribe robots to move in synchronized discrete timesteps and are not directly executable on their controllers. They do not incorporate complex dynamic constraints of the robots in the form of KDDPs.[2] MAPF-POST (Hönig et al. 2016) proposes the use of a Temporal Plan Graph (TPG) to reinstate dynamic constraints of the robots after a discrete plan has been found. However, MAPF-POST incorporates only maximum velocity constraints and makes the unrealistic assumption that robots are capable of instantaneous velocity changes.

[2]Some solvers use motion primitives, but these are not as expressive as KDDPs, and the solvers are not very scalable.

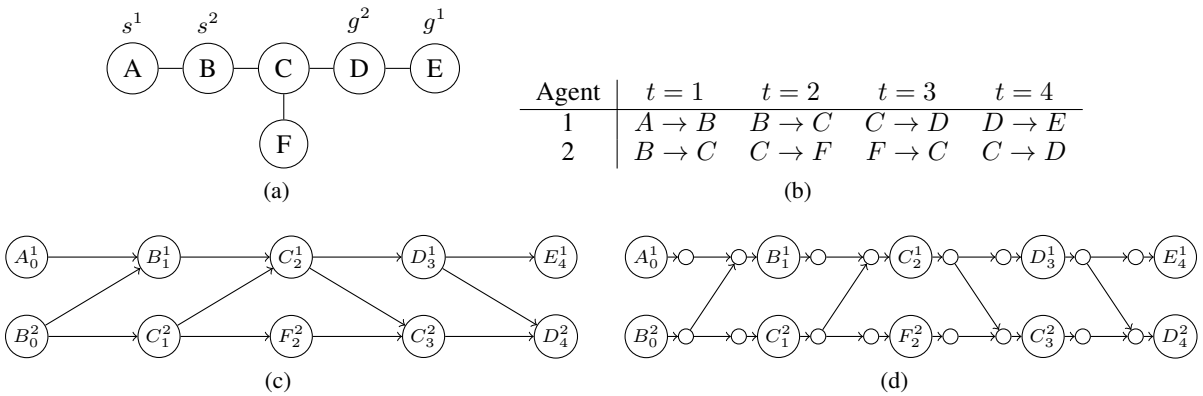


Figure 4: Illustrates the application of KDNs to MAPF. (a) shows a MAPF instance on an undirected graph with s^1 and s^2 indicating the start vertices of two robots, and g^1 and g^2 indicating their corresponding goal vertices. (b) shows a discrete plan of minimum makespan generated by a MAPF solver. (c) shows the TPG. Superscripts refer to robots, and subscripts refer to timesteps. Horizontal edges indicate motion edges for individual robots, and cross edges indicate coordination edges between robots. (d) shows the augmented TPG, in which safety distances between the robots are ensured by placing safety markers (smaller circles) around the location vertices (bigger circles). More details of safety markers and augmented TPGs can be found in (Hönig et al. 2016).

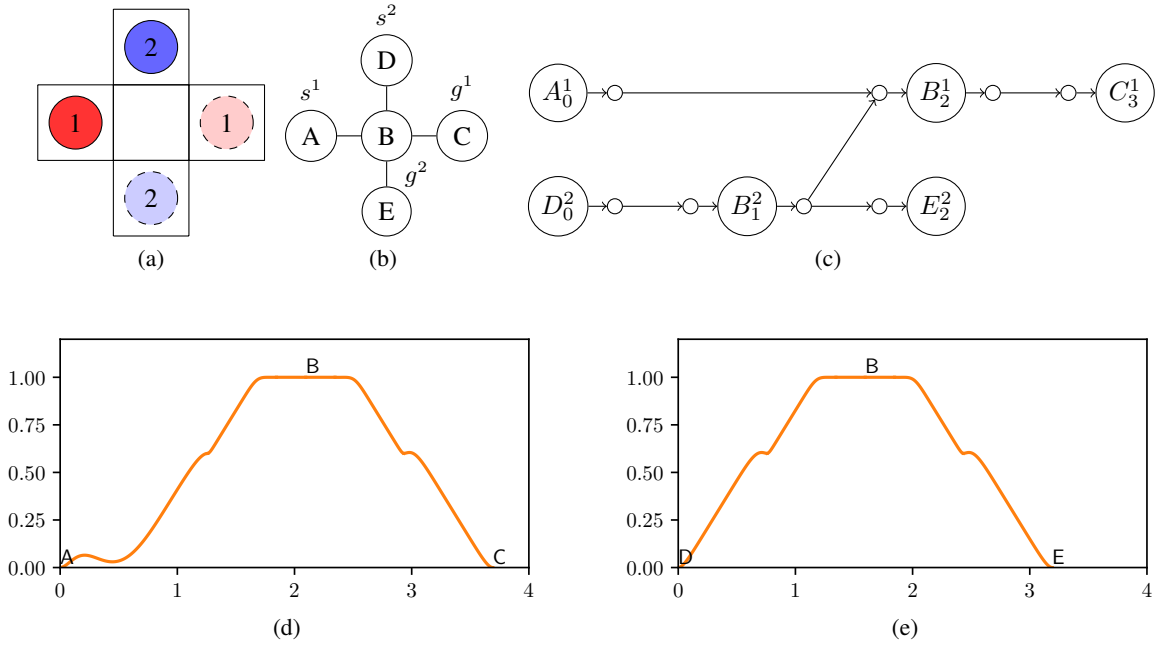


Figure 5: (a) shows a small MAPF instance with 2 robots on a four-neighbor grid map. The darker-colored circles with solid boundaries are the start locations of the robots, and the lighter-colored circles with dashed boundaries are their corresponding goal locations. (b) shows the same MAPF instance in a graph representation with vertices representing free cells and edges representing transitions between neighboring free cells. (c) shows the augmented TPG of a discrete plan generated by a MAPF solver. (d) and (e) show the velocity profiles of Robots 1 and 2, respectively, generated by our Bézier curve approach.

Figure 4 shows an example TPG generated by MAPF-POST. It also shows an augmented TPG, in which kinematic constraints representing safety distances between the robots are enforced. The augmented TPG is essentially a KDN in which arrival times of robots at various locations or safety markers represent events. The horizontal edges in it represent motion edges annotated with complex dynamic constraints of the corresponding robots. The cross edges represent non-motion coordination edges annotated with precedence constraints between robots. While MAPF-POST uses

#Robots	MAPF time	KDN time
04	00.053 (30/30)	10.064 (30/30)
06	00.214 (30/30)	18.551 (30/30)
08	11.409 (30/30)	31.882 (30/30)
10	25.819 (28/30)	43.191 (28/28)
12	68.112 (19/30)	72.172 (19/19)
14	36.938 (02/30)	92.952 (02/02)

(a)

%Obstacles	MAPF time	KDN time
10	00.296 (30/30)	5.170 (30/30)
15	00.507 (30/30)	5.468 (30/30)
20	01.142 (30/30)	6.694 (30/30)
25	19.076 (26/30)	6.757 (26/26)
30	53.692 (16/30)	9.707 (16/16)
35	08.621 (03/30)	6.144 (03/03)

(b)

Figure 6: (a) and (b) show experimental results on some MAPF instances with a varying numbers of robots and obstacle rates, respectively, on four-neighbor grid maps. In both cases, time is measured in seconds. MAPF time refers to the average time taken by a MAPF solver to generate the discrete plans. KDN time refers to the average runtime of our KDN solver for generating the velocity profiles of the robots given the output of the MAPF solver. In both cases, the average is computed over the A successfully solved instances (within a runtime limit of 5 minutes) out of the given B instances, indicated by (A/B).

only maximum velocity constraints, it can now be improved with richer dynamic constraints using the theory developed in this paper for KDNs. More generally, KDNs model many real-world multi-robot coordination problems. There are some temporal constraints that model the capabilities of the individual robots and are best specified as KDDPs. There are other temporal constraints that model the coordination between the robots and are typically precedence constraints or simple temporal constraints.

As mentioned before, the Single Interval Theorem allows us to solve a KDN by searching in the space of all combinations of boundary conditions at the landmarks. For this, we discretize the possible values of the velocities and accelerations of robots at each landmark. This discretization does not have to be fine-grained to produce good results. It is much better than discretizing the entire timeline to reason about complex differential equations. For each motion edge $e = \langle X_i, X_j \rangle$, let Δ_i and Δ_j be the set of possible boundary conditions associated with robot R^e at ℓ_s^e and ℓ_f^e, respectively. Let $\Delta_{ij} \subseteq \Delta_i \times \Delta_j$ be the subset of combinations that are feasible, that is, combinations $(b, b') \subseteq \Delta_i \times \Delta_j$ for which Algorithm 1 returns a feasible time interval $[LB^e_{b,b'}, UB^e_{b,b'}]$.

We can solve a KDN using a MILP formulation as follows. For each event X_i, we create a continuous variable t_{X_i} representing its execution time $\tau(X_i)$. For each motion edge $e = \langle X_i, X_j \rangle$, we create a 0/1 variable $x^e_{b,b'}$ for each $(b, b') \in \Delta_{ij}$. Here, $x^e_{b,b'} = 1$ indicates that robot R^e has boundary conditions b when it reaches ℓ_s^e and boundary conditions b' when it reaches ℓ_f^e. We add the constraint

$$\sum_{(b,b') \in \Delta_{ij}} x^e_{b,b'} = 1 \tag{4}$$

to enforce exactly one physical realization of the traversal. We also add the constraints

$$\begin{array}{l} \sum_{(b,b') \in \Delta_{ij}} x^e_{b,b'} LB^e_{b,b'} \leq t_{X_j} - t_{X_i} \\ \sum_{(b,b') \in \Delta_{ij}} x^e_{b,b'} UB^e_{b,b'} \geq t_{X_j} - t_{X_i} \end{array} \tag{5}$$

to indicate that fixing the boundary conditions induces simple temporal constraints by virtue of the Single Interval Theorem. Finally, for each pair of consecutive motion edges $e = \langle X_i, X_j \rangle$ and $e' = \langle X_j, X_k \rangle$, we add the constraint

$$\sum_{(b,b') \in \Delta_{ij}} x^e_{b,b'} = \sum_{(b',b'') \in \Delta_{jk}} x^{e'}_{b',b''}, \qquad \forall b' \in \Delta_j \tag{6}$$

to ensure that the boundary conditions are consistently chosen at each landmark.

The resulting MILP containing the above constraints and the constraints for the coordination edges can be efficiently solved using any MILP solver. An objective function, like makespan minimization, can also be included by simply adding the constraint $t_{X_i} \leq T$, for all X_i, and minimizing T. Figure 5 shows a small example of solving a KDN coming from a MAPF instance. Here, each edge has length $1\,\mathrm{m}$, the safety markers are placed $0.25\,\mathrm{m}$ away from the location vertices, $v_{\max} = 1\,\mathrm{m/s}$, $a_{\min} = -1\,\mathrm{m/s^2}$ and $a_{\max} = 1\,\mathrm{m/s^2}$. The velocity is discretized to be in $\{0\,\mathrm{m/s}, 0.6\,\mathrm{m/s}, 1\,\mathrm{m/s}\}$, and the acceleration is discretized to be in $\{-1\,\mathrm{m/s^2}, 0\,\mathrm{m/s^2}, 1\,\mathrm{m/s^2}\}$ at the landmarks. Using 20 control points for each stretch between consecutive landmarks yields the required velocity profiles of the robots. To minimize the makespan while maintaining a safety distance between robots, Robot 1 slows down just enough in the beginning to allow Robot 2 to cross the intersection first.

We conducted a few more experiments to test the scalability of our KDN solver. Although this paper is not about MAPF but about the general framework of KDNs, we used KDNs corresponding to MAPF instances. Figure 6 shows experimental results on MAPF instances derived from (a) a 9×19 warehouse map from (Felner et al. 2018) with a varying number of robots and (b) random 8×8 grid maps with 10 robots and varying obstacle rates. We used a state-of-the-art MAPF solver called ICBS-h4 (Felner et al. 2018), which solved a subset of the 30 instances for each setting. All our maps are four-neighbor grid maps. We ran all experiments on a laptop computer with an Intel i7 3.1GHz processor and 16GB of memory. We used SCIP 6.0 (Gleixner et al. 2018) as our MILP solver. 20 control points were used for each stretch between consecutive landmarks. The robots were modeled with the same specifications as in Figure 5. The KDN solver successfully solved all KDNs that are generated from the MAPF discrete plans. This shows that converting discrete plans to kinodynamically feasible plans for the robots using KDNs is viable, and, perhaps contrary to popular belief, rich dynamic constraints will not be the bottleneck in making multi-robot coordination scalable.

Conclusions and Future Work

Existing temporal reasoning frameworks do not efficiently incorporate differential equations or inequalities that arise in robotics. We presented KDNs to address these drawbacks. KDNs represent temporal constraints related to both coordination and motion. While the coordination kinematic constraints are usually simple temporal, the dynamic constraints on the motion of a robot can constitute a complex KDDP. We used the idea of Bernstein polynomials and Bézier curves to efficiently solve KDDPs. We also proved the Single Interval Theorem, by virtue of which KDNs can be solved efficiently using MILP solvers. We also showed that our approach is viable for converting MAPF discrete plans to smooth velocity profiles for all robots that can be executed by their controllers. An important avenue of future work is to generalize our techniques to include uncertainties, controllable, and uncontrollable events.

Acknowledgments

Han, Sven and Satish thank Neelesh for providing the proof of the Single Interval Theorem in this paper. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779 and 1935712. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

Appendix: Formal Proof of the Single Interval Theorem

Proof. Let $S \subset [T_1, T_2]$ be the subset of T values for which such a function $\tilde{h}(s)$ described in the statement of the theorem exists. We will show that $S = [T_1, T_2]$.

Lemma 1. *S is a closed subset of $[T_1, T_2]$.*

Lemma 2. *S is an open subset of $[T_1, T_2]$.*

The main work lies in proving these lemmas. Then, since $[T_1, T_2]$ is a connected set, and S is an open, closed, and nonempty subset of $[T_1, T_2]$, it will follow that $S = [T_1, T_2]$. For reference, one can consult a real analysis textbook (Rudin 1976). □

Proof of Lemma 1. Let $\{s_m\}$ in S be a sequence such that $s_m \to s$. We will show that $s \in S$. For each s_m, let g_m be the corresponding function satisfying the statement of the theorem. If $s = T_1$, we are done, so assume $T_1 < s$ and fix r_1 such that $T_1 < r_1 < s$.

Without loss of generality, we can assume that all of the g_m are defined on $[0, r_1]$ since we can drop the earlier terms of the sequence g_m until this is the case. Let $g_m^{(k)}$ denote the sequence of k^{th} derivatives. By the low-high constraints and the universal Lipschitz constant C, each sequence $g_m^{(k)}$ is uniformly bounded and equicontinuous for $0 \leq k \leq K$. Thus, by the Arzelà-Ascoli Theorem (Rudin 1976), $g_m^{(k)}$ admits a convergent subsequence. By repeatedly passing to subsequences, we can find a subsequence h_n of g_m and a function g defined on $[0, r_1]$ such that

$$h_n^{(k)} \xrightarrow{\text{uniformly}} g^{(k)} \quad \text{on } [0, r_1] \text{ for } 0 \leq k \leq K.$$

Now fix $r_1 < r_2 < s$. Start with the h_n sequence instead and repeat the process above to find a subsequence h_{n_l} and a function g_2 (automatically an extension of g) such that

$$h_{n_l}^{(k)} \xrightarrow{\text{uniformly}} g_2^{(k)} \quad \text{on } [0, r_2] \text{ for } 0 \leq k \leq K.$$

Now pick a sequence $r_1, r_2, \ldots$ increasing to s and repeat the above process for each r_i. We will construct a sequence p_n by using a diagonalization argument illustrated in Figure 7a. For every $t \in [0, s)$, define $g^\star(t)$ as $\lim_{n\to\infty} p_n(t)$, where these numbers will be defined past a large n value depending on t. By the above, $p_n^{(k)} \xrightarrow{\text{uniformly}} g^{\star(k)}$ on all intervals $[0, r]$, $r < s$ and for $0 \leq k \leq K$. Thus, since each p_n satisfies the low-high conditions, so does $g^\star$. Since each p_n satisfies the $t = 0$ constraints, so does $g^\star$. Uniform limits of Lipschitz functions with Lipschitz constant C retain this property, and so, $g^{\star(K)}$ has this property. By the uniform continuity of $g^\star$ and its derivatives, $g^\star$ extends to a function on $[0, s]$ which is K times differentiable (proved easily with the mean value theorem). Furthermore, each p_i is associated with an s_i in the original s_m sequence and $s_i \to s$. Fix k with $0 \leq k \leq K$. Without loss of generality, we can assume $s_i \nearrow s$ or $s_i \searrow s$.

If $s_i \nearrow s$:

$$\begin{aligned} |g^{\star(k)}(s) - c_{final}^k| &\leq |g^{\star(k)}(s) - g^{\star(k)}(s_i)| \\ &\quad + |g^{\star(k)}(s_i) - p_i^{(k)}(s_i)| \\ &\quad + |p_i^{(k)}(s_i) - c_{final}^k| \longrightarrow 0. \end{aligned}$$

If $s_i \searrow s$:

$$\begin{aligned} |g^{\star(k)}(s) - c_{final}^k| &\leq |g^{\star(k)}(s) - p_i^{(k)}(s)| \\ &\quad + |p_i^{(k)}(s) - p_i^{(k)}(s_i)| \\ &\quad + |p_i^{(k)}(s_i) - c_{final}^k| \longrightarrow 0. \end{aligned}$$

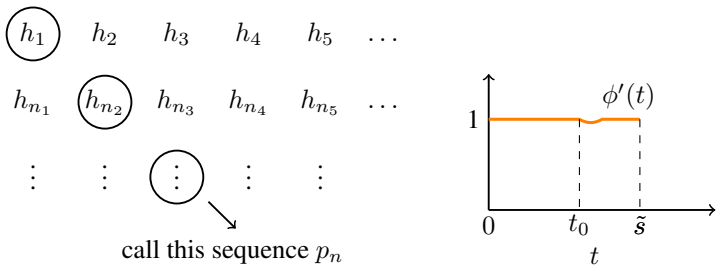
Thus, $g^\star$ satisfies the $t = s$ constraints as well and so $g^\star$ is the required function. In other words, s is in S. □

Proof of Lemma 2. Take a point $s \in S$ with $s < T_2$. We want to show that there exists some $\epsilon > 0$ such that $[s, s+\epsilon)$ is contained in the set S. (Then, by a symmetrical argument, it will hold that for all $r \in S$ with $r > T_1$, $(r - \delta, r]$ will be contained in S for some small δ. Combining these two facts will give us that all points in S are interior points, which means that S is open.) Let g be the function associated to s that meets the above theorem constraints.

Case 1: For some $s^\star \in [0, s]$, $g^{(K)}(s^\star) \notin \{0, c_{low}^K, c_{high}^K\}$. In this case, by continuity, $\exists s_\ell$ and s_r such that

$$g^{(K)}([s_\ell, s_r]) \cap \{0, c_{low}^K, c_{high}^K\} = \emptyset.$$

This implies that there is an $\tilde{s}^\star$ in $[s_\ell, s_r]$ such that $g^{(K-1)}(\tilde{s}^\star) \notin \{0, c_{low}^{K-1}, c_{high}^{K-1}\}$ and so there is a subinterval



(a) (b)

Figure 7: Shows the visual aids for the proof of the Single Interval Theorem. (a) shows the sequence p_n for the proof of Lemma 1. (b) shows the figure of $\phi'(t)$ for the proof of Lemma 2. In $\phi'(t)$, we build a divot with a smooth bump function around the point t_0 so that the total area between $t = 0$ and $t = \tilde{s}$ under the curve of $\phi'(t)$ is s. This gives us that $\phi(\tilde{s}) = s$, and $\phi(t)$ will be smooth.

I of $[s_\ell, s_r]$ such that $g^{(K-1)}(I) \cap \{0, c_{low}^{K-1}, c_{high}^{K-1}\} = \emptyset$. Continuing in this way, we can find a subinterval J of $[0, s]$ such that

$$g^{(k)}(J) \subset (c_{low}^k, c_{high}^k) \quad \text{for } k = 0, 1 \dots K.$$

Pick t_0 in the interior of J. Define $\tilde{s} = s + \varepsilon$, where $\varepsilon > 0$ will be determined later. Define ϕ: $[0, \tilde{s}] \to [0, s]$ as follows

- $\phi(0) = 0$
- $\phi'(t)$ is given by Figure 7b

Define $f : [0, \tilde{s}] \to \mathbb{R}$ as $f(t) := g(\phi(t))$. Intuitively, f is a deformation of g around the point t_0 that has the effect of stretching the graph of g slightly about this point. By continuity (since the bump function used in building the divot is small and so are its derivatives), the derivatives of f near t_0 will not differ much from those of g. Thus, since we have wiggle room near t_0, f will still obey the low-high conditions.

Whenever $\phi'(t) = 1$, which is most of the time, and in particular, at $t = 0$ and $t = \tilde{s}$:

$$\begin{aligned} f^{(k)}(t) &= g^{(k)}(\phi(t))(\phi'(t))^k = g^{(k)}(\phi(t)) \\ &\Rightarrow \text{The low-high conditions are satisfied at such } t. \\ &\Rightarrow \begin{cases} f^{(k)}(0) = g^{(k)}(\phi(0)) = g^{(k)}(0) \\ f^{(k)}(\tilde{s}) = g^{(k)}(\phi(\tilde{s})) = g^{(k)}(s). \end{cases} \end{aligned}$$

Therefore, the boundary conditions are met.

To see what happens at other t, consider $k = 3$:

$$\begin{aligned} f^{(3)}(t) &= g^{(3)}(\phi(t))[\phi'(t)]^3 + 2g''(\phi(t))\phi'(t)\phi''(t) \\ &\quad + g''(\phi(t))\phi''(t)\phi'(t) + g'(\phi(t))\phi'''(t) \\ &= g^{(3)}(\phi(t))[\phi'(t)]^3 + 3g''(\phi(t))\phi'(t)\phi''(t) \\ &\quad + g'(\phi(t))\phi'''(t) \\ &\approx g^{(3)}(\phi(t))[\phi'(t)]^3 \\ &\approx g^{(3)}(\phi(t)) \approx g^{(3)}(t_0). \end{aligned}$$

Thus, $f^{(3)}$ at such t is close to $g^{(3)}(t_0)$ and we picked t_0 to allow for a margin of error. Thus, for $\tilde{s}$ close to and to the right of s, we can construct a function f as in the above which meets the required constraints.

Case 2: $g^{(K)}(t)$ is constant and equal to $0, c_{low}^K$ or c_{high}^K.

Subcase 1: $0 < c_{low}^K < c_{high}^K$.

Suppose $g^{(K)}(t) = c_{low}^K$

$$\begin{aligned} &\Rightarrow c_{final}^{K-1} - c_{init}^{K-1} = \int_0^s g^{(K)}(t)dt = \int_0^s c_{low}^K dt \\ &< \int_0^{T_2} c_{low}^K dt \leq \int_0^{T_2} f_2^{(K)}(t)dt = c_{final}^{K-1} - c_{init}^{K-1}. \end{aligned}$$

This is clearly impossible. The only other case to worry about that is not already covered in Case 1 is $g^{(K)}(t) = c_{high}^K$. Then,

$$\begin{aligned} c_{final}^{K-1} - c_{init}^{K-1} &= \int_0^s c_{high}^K dt \geq \int_0^{T_1} c_{high}^K dt \\ &\geq \int_0^{T_1} f_1^{(K)}(t)dt = c_{final}^{K-1} - c_{init}^{K-1}. \end{aligned}$$

The only way that the inequality is an equality is if $s = T_1$. In this case, we can show that the proof in Case 1 still works. In other words, we do not need wiggle room on both sides, and the constructed $f^{(K)}$ will stay within bounds.

Subcase 2: $c_{low}^K < c_{high}^K < 0$.

A symmetrical argument to Subcase 1.

Subcase 3: $c_{low}^K < 0 < c_{high}^K$.

By a similar argument to Subcase 1, $g^{(K)}(t)$ is neither identically c_{low}^K or c_{high}^K. The only situation in which the argument in Case 1 may not work is if $g^{(K)}(t) = 0$ identically. In this case, we are only worried that $g^{(K-1)}$ is constant, and we proceed inductively.

□

References

Brucker, P.; Hilbig, T.; and Hurink, J. 1999. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics* 94(1-3): 77–99.

Choi, J.; Curry, R.; and Elkaim, G. 2008. Path planning based on Bézier curve for autonomous ground vehicles. In *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science (WCECS)*, 158–166.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3): 61–95.

Farin, G. 1992. *Curves and surfaces for computer-aided geometric design: a practical guide*. Elsevier, 3rd edition.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 83–87.

Gleixner, A.; Bastubbe, M.; Eifler, L.; Gally, T.; Gamrath, G.; Gottwald, R. L.; Hendel, G.; Hojny, C.; Koch, T.; Lübbecke, M. E.; Maher, S. J.; Miltenberger, M.; Müller, B.; Pfetsch, M. E.; Puchert, C.; Rehfeldt, D.; Schlösser, F.; Schubert, C.; Serrano, F.; Shinano, Y.; Viernickel, J. M.; Walter, M.; Wegscheider, F.; Witt, J. T.; and Witzig, J. 2018. The SCIP Optimization Suite 6.0. Technical report, Zuse Institute Berlin.

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 477–485.

Ji, M.; He, Y.; and Cheng, T. E. 2007. Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research* 34(6): 1764–1770.

Khatib, L.; Morris, P.; Morris, R.; and Rossi, F. 2001. Temporal constraint reasoning with preferences. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 322–327.

Knight, S.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: space exploration through continuous planning. *IEEE Intelligent Systems* 16(5): 70–75.

Lorentz, G. G. 1986. *Bernstein polynomials*. American Mathematical Society, 2nd edition.

Ma, H.; and Koenig, S. 2017. AI Buzzwords Explained: Multi-Agent Path Finding (MAPF). *AI Matters* 3(3): 15–19.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*, 608–614.

Mortenson, M. E. 1999. *Mathematics for computer graphics applications*. Industrial Press Inc., 2nd edition.

Pecora, F.; and Cirillo, M. 2009. A constraint-based approach for plan management in intelligent environments. In *Scheduling and Planning Applications woRKshop (SPARK) of the International Conference on Automated Planning and Scheduling (ICAPS)*, 19–23.

Rudin, W. 1976. *Principles of Mathematical Analysis*. McGraw-Hill, 3rd edition.

Stergiou, K.; and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120(1): 81–117.

Tang, S.; and Kumar, V. 2016. Safe and complete trajectory generation for robot teams with higher-order dynamics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1894–1901.

Vidal, T. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11(1): 23–45.

Wilcox, R.; Nikolaidis, S.; and Shah, J. 2013. Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. *Robotics* 441–456.