# Space-Time Conflict Spheres for Constrained Multi-Agent Motion Planning

Anirudh Chari[1], Rui Chen[2], and Changliu Liu[2]

***Abstract*—Multi-agent motion planning (MAMP) is a critical challenge in applications such as connected autonomous vehicles and multi-robot systems. In this paper, we propose a space-time conflict resolution approach for MAMP. We formulate the problem using a novel, flexible sphere-based discretization for trajectories. Our approach leverages a depth-first conflict search strategy to provide the scalability of decoupled approaches while maintaining the computational guarantees of coupled approaches. We compose procedures for evading discretization error and adhering to kinematic constraints in generated solutions. Theoretically, we prove the continuous-time feasibility and formulation-space completeness of our algorithm. Experimentally, we demonstrate that our algorithm matches the performance of the current state of the art with respect to both runtime and solution quality, while expanding upon the abilities of current work through accommodation for both static and dynamic obstacles. We evaluate our algorithm in various unsignalized traffic intersection scenarios using CARLA, an open-source vehicle simulator. Results show significant success rate improvement in spatially constrained settings, involving both connected and non-connected vehicles. Furthermore, we maintain a reasonable suboptimality ratio that scales well among increasingly complex scenarios.**

## I. Introduction

Connected autonomous vehicles (CAVs) are self-driving vehicles with the ability to communicate with infrastructure and other vehicles. Vehicle-to-vehicle communication enables coordination among CAVs, which will greatly improve both the safety of road participants [1], [2] and the efficiency of traffic flow [3]. We are particularly interested in the CAV coordination at traffic intersections [4], which are the site of a majority of road accidents due to human error [5]. The problem can be formulated as multi-agent motion planning (MAMP), which plans and coordinates trajectories among a group of agents such that each agent can travel from its start location to its goal without collisions with other agents or with the environment. MAMP is also useful in surveillance, search-and-rescue, warehouse, and assembly robot groups. See [6] for a thorough review.

MAMP is a generalization of the multi-agent path-finding problem (MAPF), where time is discretized into timesteps and agents move along the edges of a discrete graph. Finding an optimal solution to MAPF is NP-hard [7], hence optimal MAMP is also computationally intractable. There are generally two approaches to MAPF: coupled methods and decoupled methods. Coupled methods are often also referred to in literature as centralized, and decoupled methods are often referred to as decentralized or distributed. Coupled methods search for solutions within a configuration space containing all agents, which enables guarantees of optimality and completeness. Despite recent advances in efficiency [8]–[10], coupled methods are unable to escape the exponential time complexity that comes with the high dimensionality of the configuration space. On the other hand, decoupled methods consider agent paths individually before combining paths through conflict resolution strategies, which enables faster processing and better scalability, but with the drawback of difficulty in guaranteeing completeness and solution quality. In pursuit of computational tractability, we are motivated to further explore decoupled approaches.

Within decoupled MAMP approaches, there exist two primary conflict resolution strategies: temporal approaches and path prioritization. Temporal conflict resolution involves manipulating agent velocity profiles along respective paths, causing agents to pass through the conflict zone at different moments [11]–[13]. In path prioritization, each agent is assigned a priority, and agents plan their paths sequentially in order of priority, with lower priority agents treating higher priority agents as dynamic obstacles [14]–[16]. Temporal methods are inherently suboptimal due to the omission of spatial trajectory manipulation. Path prioritization methods face an inevitable bottleneck due to the requirement of sequential processing. Thus, we are motivated to address these drawbacks by pursuing a space-time conflict resolution strategy with the ability for parallelized path prioritization.

Due to the applicability to CAVs and other domains, we also desire continuous-time feasibility and accommodation for dynamic obstacles as critical properties of a MAMP algorithm. Recent work in decoupled MAPF has excelled in providing computational guarantees such as completeness [17], [18]. However, it is difficult to generalize MAPF solutions to MAMP problems, as discretization error may give rise to new conflict in continuous-time, and these approaches do not consider agent kinematic constraints. Thus, we reach a dilemma: discretization of MAMP allows for easier formulation of robust and efficient algorithms, but at the expense of continuous-time feasibility, and consequently applicability to real-world settings. On the other hand, attacking continuous MAMP directly risks computational intractability and complicates trajectory formulation, which may in turn restrict solution flexibility. Some work attempts to adapt discrete solutions to the continuous problem [19], [20], but no formal proof of feasibility in continuous time is present, and thus

[1] Anirudh Chari is with the Illinois Mathematics and Science Academy (achari@imsa.edu)

[2] Rui Chen and Changliu Liu are with the Robotics Institute, Carnegie Mellon University ({ruic3, cliu6}@andrew.cmu.edu)

much of this work is unsuitable for real-world implementation. While the authors of [21] provide a proof of continuous-time feasibility for their discretization strategy, dynamic obstacles are ignored, which especially hinders applicability to the CAV domain, where scenarios involving pedestrians and non-connected vehicles are commonplace. To the best of the authors' knowledge, we are still missing literature regarding **decoupled, discretized MAMP algorithms with continuous-time feasibility and accommodation for dynamic obstacles**. In this work, we aim to fill this gap.

We propose a novel decoupled approach to MAMP called the **S**pace-**T**ime **C**onflict **S**pheres (STCS) algorithm. STCS utilizes a sphere-based trajectory discretization to manipulate paths both spatially and temporally during conflict resolution. We theoretically prove STCS's continuous-time feasibility and formulation-space completeness. We experimentally demonstrate that the algorithm exhibits comparable performance to the current state of the art while offering a greater range of problem settings and more consistency in finding solutions, namely within environments that are spatially constrained and contain dynamic obstacles.

The rest of the paper is organized as follows. Section II formulates the problem of MAMP. Section III discusses the STCS algorithm. Section IV discusses the theoretical properties of STCS. Section V presents experimental results in simulation. Finally, Section VI concludes our study and presents future directions.

## II. Problem Formulation

We are given a set $\mathcal{A}$ containing $N$ agents of radius $r$ in a continuous-time, two-dimensional space, where each agent $\alpha_i \in \mathcal{A}$ is defined by its starting location $q_i^s$, goal location $q_i^g$, acceleration bound $a_i^{max}$, priority $\phi_i$, and path rigidity $\gamma_i$. We are also given a workspace $\mathcal{W}$ containing $M$ obstacles, where each static obstacle $o_i^s \in \mathcal{W}$ has a known location, and each dynamic obstacle $o_i^d \in \mathcal{W}$ has a known trajectory.

We use $\mathcal{P}$ to denote the set of agent and obstacle paths. Each path $\pi_i \in \mathcal{P}$ can be represented spatiotemporally as a capsule, which we define as a curve inflated with radius $\lambda r$ for some $\lambda > 1$. The capsule representation is analogous to a chain of infinitely many spheres with radius $\lambda r$.

We introduce a novel, flexible discretization of this representation to a finite chain of spheres, and introduce a requirement that adjacent spheres within a path must be no further than tangential to each other. By Theorem IV.1, this discrete representation maintains continuous-time feasibility (i.e. no discretization error) when $r$ is scaled by $\lambda^* = \frac{1}{\sqrt{3}-1} \approx 1.366$, meaning paths can be interpolated safely for agent motion control. For succinct representation, we can also introduce a requirement that alternating spheres within a path must be further than tangential to each other, ensuring that a path is always built from the minimum number of spheres.

Using this representation, a path $\pi_i$ can be defined as a sequence of spatiotemporal spheres $\{s_{i,1}, ..., s_{i,n}\}$ as waypoints, where $s_{i,k}$ is the $k$-th waypoint in $\pi_i$. A waypoint $s_{i,k}$ has components $(x_{i,k}, y_{i,k}, t_{i,k})$, where $(x_{i,k}, y_{i,k})$ is a physical location in the environment and $t_{i,k}$ is the time at which the location is occupied. Each $s_{i,k}$ also has a corresponding velocity vector $\vec{\omega}_{i,k}$, which is computed based on conflict resolution conditions and kinematic constraints.

All paths in $\mathcal{P}$ can exist synchronously within a central *space-time grid* (STG), which we define as a subspace of $\mathbb{R}^3$ with basis $\{\hat{x}, \hat{y}, \hat{t}\}$. We assume that each agent $\alpha_i$ uses some single-agent motion planner to sequentially upload waypoints to its path $\pi_i$ in the STG. In practice, either some centralized infrastructure can manage the STG while agents upload and query data, or each agent can maintain its own copy of the STG and send and receive broadcasts in a decentralized manner. Among all paths in $\mathcal{P}$, an intersection between a pair of spheres belonging to distinct paths in the STG implies conflict. Agents take turns uploading waypoints to the STG, and when a conflict is detected, it must be resolved through the manipulation of paths in the STG. We use $s_{i,k}^r$ and $s_{i,k}^*$ to denote the initial (reference) and final (optimal) locations of a sphere $s_{i,k}$, respectively.

Thus, we leverage our sphere-based discretization strategy to formulate conflict resolution for MAMP as an optimization problem:

$$
\begin{aligned}
s_{i,k}^* \forall (i,k) = \\
\underset{s_{i,k} \forall (i,k)}{\operatorname{argmin}} \quad & \sum_{i=1}^{n} \phi_i \left\| s_{i,k} - s_{i,k}^r \right\|_2 && \text{(1a)} \\
\text{s.t.} \quad & \left\| s_{i,k} - s_{j,l} \right\|_2 \geq 2r, \forall (s_{i,k}, s_{j,l}) \in \mathcal{P}, i \neq j && \text{(1b)} \\
& \left\| s_{i,k+1} - s_{i,k} \right\|_2 \leq 2r, \forall (s_{i,k}, s_{i,k+1}) \in \mathcal{P} && \text{(1c)} \\
& t_{i,k+1} - t_{i,k} \leq (\delta t)_{i,k}, \forall (s_{i,k}, s_{i,k+1}) \in \mathcal{P} && \text{(1d)}
\end{aligned}
$$

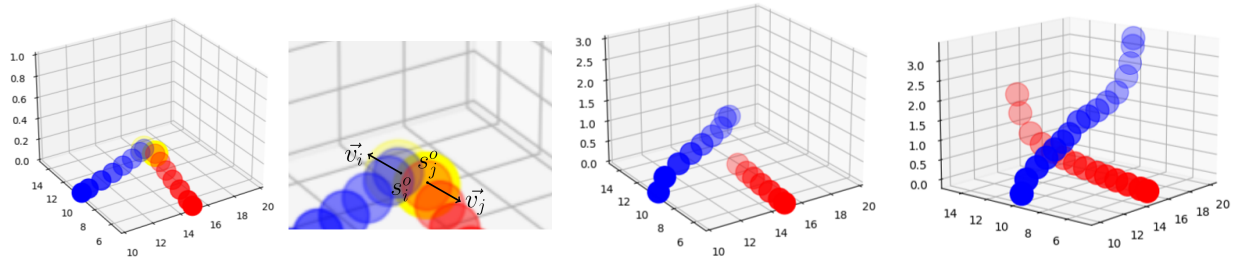
Equation (1a) minimizes the prioritized total displacement of spheres from their "optimal" original state in their respective path. Equation (1b) enforces that no pair of spheres from different paths can intersect. Equation (1c) enforces that consecutive spheres within a path must intersect. Equation (1d) enforces that time intervals between consecutive spheres within a path must be compatible with kinematic constraints; the computation of $(\delta t)_{i,k}$ is given in eq. (5).

Solving this optimization problem to resolve conflicts following the convergence of all agent paths to their respective goals would yield an optimal solution to MAMP. However, this leaves the task of solving a non-convex and potentially large optimization, which risks computational intractability. Instead, conflict resolution can be applied following each agent waypoint upload. Employing this method as a heuristic, as we will observe, enables fast convergence to feasible solutions that are suboptimal within a reasonable bound.

## III. Space-Time Conflict Spheres

### A. Overview

As agents sequentially upload waypoints to the STG, a conflict may arise between agents as an intersection between




(a) Conflict detection (b) DV Computation (c) Path Shift (d) Final solution

Fig. 1: A simple scenario involving two agents initially positioned perpendicular to each other, each with a goal directly across the field. The conflict is highlighted in yellow. Because the red path is given higher priority, the conflict search finds that shifting the blue path yields the best solution.

**Algorithm 1** Calling STCS

```
1:  function SolveMAMP(𝒜)
2:      while not all α_i reached goal do
3:          for all α_i ∈ 𝒜 do
4:              w ← α_i.GetWaypoint( )
5:              STG.UploadWaypoint(π_i, w)
6:              if STG.HasConflict( ) then
7:                  STG.RunSTCS( )
8:                  for all α_j ∈ 𝒜 do
9:                      p ← STG.GetPath(α_j)
10:                     α_j.SetPath(p)
```

spheres of distinct paths in the STG. The objective of STCS is to resolve this intersection upon formation, while attempting to minimize the total displacement of spheres in the STG during this process. Simultaneously, the algorithm must ensure that connectivity (1c) and compliance with kinematic constraints (1d) is maintained within each path. Resolving one intersection may lead to the formation of many others, making this a difficult problem.

We begin by approaching the sub-problem of computing the minimum displacement required to resolve an intersection between a single pair of spheres, and we solve this by introducing the idea of *displacement vectors* (DVs). Then, we move to considering the effect of a sphere's translation on its individual path, namely through a *path shift*, which first deforms a path around its displaced sphere then applies a smoothing operation to maintain connectivity and adherence to temporal and kinematic constraints. Finally, we employ the two above concepts and formulate an efficient search procedure for collecting complete solutions to the current conflict using a depth-first paradigm; we refer to this process as *conflict search*.

The high-level conflict resolution procedure is outlined in Algorithm 1, and a visual overview of the process is given in Figure 1. Each of the following three subsections details one of the aforementioned aspects of STCS.

### B. Displacement Vectors

We define an *outstanding* sphere as a sphere in the STG with the potential to be involved in an intersection with another sphere of a different path. We denote the set of all outstanding spheres in a solution as $\mathcal{S}$. Each sphere $s_i^o \in \mathcal{S}$ has a single corresponding DV $\vec{v}_i \in \mathcal{V}$, where $\mathcal{V}$ is the set of all DVs. Our goal is to compute vectors for each of the outstanding spheres such that applying $s_i^o \mathrel{+}= \vec{v}_i$ for all $s_i^o \in \mathcal{S}$ is guaranteed to yield an intersection-free STG.

The translation caused by DVs for a pair of intersecting spheres can be visualized as a repulsive force acting between two charged particles. In general, the DV $\vec{v}_i$ of outstanding sphere $s_i^o$ to resolve intersection with sphere $s_j^o$ is given by

$$\vec{v}_i = \left( \frac{2r}{\left\| s_i^o - s_j^o \right\|_2} - 1 \right) (s_i^o - s_j^o) \tag{2}$$

The magnitude of $\vec{v}_i$ is the minimum displacement of $s_i^o$ necessary to resolve the intersection, assuming $s_j^o$ is stationary. The direction of $\vec{v}_i$ is orthogonal to the plane of intersection between $s_i^o$ and $s_j^o$ (see Figure 2). In the case where two paths advance straight towards one another, we can introduce a small bias in the angle of each DV in the conflicting sphere pair to ensure the paths can navigate around each other. Note that the first sphere in any path is immutable both spatially and temporally, and the final sphere in any path that has converged to its goal is immutable spatially.

### C. Path Shifts

Path shifts are first simulated during the conflict search stage, then finally applied post-optimization. The displacement of any outstanding sphere along its DV will cause a shift within that sphere's path, centered around it. This can be intuitively visualized as a rubber rod deforming after being hit by a ball (see Figure 3).

For a path $\pi_i$ that contains an outstanding sphere $s_j^o$ with DV $\vec{v}_j$, we define the following coefficient for each sphere $s_{i,k} \in \pi_i$:

$$\mu_{i,k} = \exp\left( -\gamma_i \left( \frac{d_{i,k}}{d_i^{max}} \right)^2 \right), \tag{3}$$

where $d_{i,k}$ is the distance between $s_{i,k}$ and the outstanding sphere $s_j^o$, $d_i^{max}$ is the maximum distance between any point along $\pi_i$ and $s_j^o$ (see Figure 3a). Furthermore, $\gamma_i$ is a positive constant assigning the path rigidity of $\pi_i$: larger $\gamma_i$ localizes

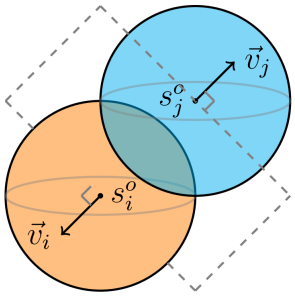
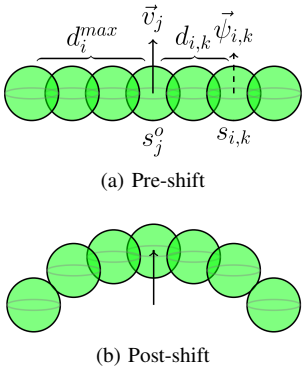



Fig. 2: Computed DVs for a pair of intersecting spheres.




(a) Pre-shift

(b) Post-shift

Fig. 3: A path shift centered around $s_j^o$ translating across $\vec{v}_j$.

the effects of the collision around the outstanding sphere, while smaller values resonate the effects throughout the path.

Then, each $s_{i,k}$ is accordingly translated along a path shift vector $\vec{\psi}_{i,k}$ generally given by

$$\vec{\psi}_{i,k} = \mu_{i,k}\vec{v}_j \tag{4}$$

Similar to the conditions for DV computation, the first sphere in any path is immutable both spatially and temporally, and the final sphere in any path that has converged to its goal is immutable spatially. In the former case, we can simply set the DV or path shift vector to $\vec{0}$, and in the latter case, we can project the vector onto $\hat{t}$.

By adapting basic kinematics equations and solving for time, the minimum timestep $(\delta t)_{i,k}$ required for an agent $\alpha_i$ to traverse between points $s_{i,k}$ and $s_{i,k+1}$ on its path is given by

$$(\delta t)_{i,k} = \frac{-\left\|\vec{\omega}^{\sigma}_{i,k}\right\|_2 + \sqrt{\left\|\vec{\omega}^{\sigma}_{i,k}\right\|_2^2 + 2a_i^{max}\|\vec{\sigma}_{i,k}\|_2}}{a_i^{max}} \tag{5}$$

where $\vec{\sigma}_{i,k}$ denotes the projection of $(s_{i,k+1} - s_{i,k})$ onto $\{\hat{x}, \hat{y}\}$, i.e. the spatial displacement between $s_{i,k}$ and $s_{i,k+1}$, $\vec{\omega^{\sigma}}_{i,k}$ is the projection of agent $\alpha_i$'s velocity vector $\vec{\omega}_{i,k}$ at $s_{i,k}$ onto $\vec{\sigma}_{i,k}$, i.e. the velocity of $\alpha_i$ along $\pi_i$ at point $s_{i,k}$, and $a_i^{max}$ is the agent's acceleration bound. After applying eq. (4), the following path-smoothing operation is executed by iterating forward through the current agent path $\pi_i$, which computes a velocity profile and removes all kinematic constraint violations:

$$\vec{\omega}_{i,k} := \vec{\omega}_{i,k-1} + a_i^{max}(\delta t)_{i,k}\hat{\sigma}_{i,k} \tag{6a}$$

$$t_{i,k} := \max(t_{i,k}, t_{i,k-1} + (\delta t)_{i,k}) \tag{6b}$$

This smoothing strategy pushes the trajectory to its kinematic limits by maximizing velocity while ensuring agreement with kinematic constraints.

For scenarios in which agent paths are spatially constrained along a general direction, e.g. lane markings at a traffic intersection, it is simple to introduce a requirement that makes certain spheres along the trajectory immutable spatially.

### D. Conflict Search

We can utilize a three-dimensional range querying data structure to efficiently query pairs of intersecting spheres within the STG during each iteration of conflict resolution. We choose to employ the $k$-d tree data structure, which provides average-case logarithmic time complexity for range queries using space partitioning.

In order to determine the set of outstanding spheres $\mathcal{S}$ and their respective DVs $\mathcal{V}$, we employ a recursive, depth-first conflict search. To construct the recursion, we define *calling* a sphere as translating the sphere across some specified DV within some current state of the STG, applying a path shift, querying further intersections, computing the DV of each sphere involved in an intersection, and finally calling each of these involved spheres with their respective DVs and the new STG state. There are two base cases for this recursion when a sphere is called: if the sphere has already been visited in the current recursion sequence, it returns false (infeasible), and if no more intersections arise following the sphere's translation and path shift, it returns true (feasible).

Through recursion, a sequence of sphere translations is accumulated, which generates a solution. Multiple solutions are obtainable since each intersection can be decomposed into two cases (e.g. $s_1$ moves vs. $s_2$ moves). In the case of an agent-obstacle sphere intersection, only the agent sphere can be called. In the case of a chain reaction of intersections (e.g. $s_1$ intersects $s_3$ after resolving intersection with $s_2$), the requirement that visited spheres cannot be called implies that only one new sphere will be called. Once all solutions have been collected, the best can be selected by minimizing the following objective function:

$$\sum_{i=1}^{n} \phi_i \|\vec{v}_i\|_2 \tag{7}$$

where $\phi_i$ is the priority value of the path that $s_i^o$ belongs to. Note that eq. (7) is a refinement of eq. (1a) that only allows the manipulation of outstanding spheres, and restricts the movement of these spheres to the magnitude and direction of their respective DVs. This formulation enables parallelized path prioritization, since trajectories can be planned simultaneously while still implicitly favoring high-priority agents during conflict.

The above procedure is summarized in Algorithm 2. We use $\mathcal{L}$ to denote the set of all solutions found by the conflict search, where each solution $\mathcal{L}_i$ is an object containing a set of outstanding spheres and respective DVs $\{\mathcal{S}_i, \mathcal{V}_i\}$ in a sequence that resolves all conflict. We also utilize a variable $\mathcal{T}$ to store the state of the STG following a particular sequence of translations during conflict search, and an array $vis$ to query whether a given sphere has been visited. We will show in Theorem IV.2 that this algorithm is complete with respect to the formulation space $\mathcal{F}$ of the conflict (see Definition IV.2).

## IV. Theoretical Properties

**Definition IV.1** (feasibility)**.** *We refer to a MAMP solution as* ***feasible*** *if the computed path configuration is conflict-free in*

**Algorithm 2** Conflict search

**function** RESOLVE($\mathcal{T}_{cur}, s_{cur}, \vec{v}_{cur}, vis$)
  **if** $vis[s_{cur}]$ **then**
    **return** [ ] ▷ Infeasible, already visited
  $vis[s_{cur}] \leftarrow$ True
  $\mathcal{L}_{cur} \leftarrow$ [ ]
  $\mathcal{T}_{new} \leftarrow$ PATHSHIFT($\mathcal{T}_{cur}, s_{cur}, \vec{v}_{cur}$)
  $query \leftarrow$ QUERYPAIRS($\mathcal{T}_{new}$)
  $feasible \leftarrow$ False
  **for all** $(s_{i,k}, s_{j,l})$ in $query$ **do**
    $\vec{v}_{i,k} \leftarrow$ COMPUTEDV($\mathcal{T}_{new}[s_{i,k}], \mathcal{T}_{new}[s_{j,l}]$)
    $\vec{v}_{j,l} \leftarrow$ COMPUTEDV($\mathcal{T}_{new}[s_{j,l}], \mathcal{T}_{new}[s_{i,k}]$)
    $\mathcal{L}_{i,k} \leftarrow$ RESOLVE($T_{new}, s_{i,k}, \vec{v}_{i,k}, vis$)
    $\mathcal{L}_{j,l} \leftarrow$ RESOLVE($T_{new}, s_{j,l}, \vec{v}_{j,l}, vis$)
    $\mathcal{L}_{cur} \mathrel{+}= \mathcal{L}_{i,k} + \mathcal{L}_{j,l}$
    **if** $(\mathcal{L}_{i,k} + \mathcal{L}_{j,l})$ not empty **then**
      $feasible \leftarrow$ True
  **if** not $feasible$ and $query$ not empty **then**
    **return** [ ] ▷ Infeasible, unresolved conflict
  **if** $\mathcal{L}_{cur}$ empty **then**
    PUSH($\mathcal{L}_{cur}$, [ ]) ▷ Feasible, end of solution
  **for all** $sol$ in $\mathcal{L}_{cur}$ **do**
    PUSH($sol, (s_{cur}, \vec{v}_{cur})$) ▷ Feasible, build solutions
  **return** $\mathcal{L}_{cur}$ ▷ All solutions

*the continuous time domain.*

**Theorem IV.1** (continuous-time feasibility)**.** *If there exists a solution to the discrete-time problem* (1) *when $r$ is scaled by $\lambda^* = \frac{1}{\sqrt{3}-1}$, then the solution is feasible.*

*Proof.* The chain-of-spheres path representation is discrete, and thus discretization error is inherent. The error occurs if an intersection exists in the capsule representation, but not in the chain-of-spheres representation. Suppose we have a sphere $s_{i,k}$ from path $\pi_i$, and two adjacent spheres $s_{j,l}$ and $s_{j,l+1}$ from a second path $p_j$. Here, discretization error would occur if $s_{i,k}$ intersects neither $s_{j,l}$ nor $s_{j,l+1}$, but it is still within the bounding capsule $c$ between $s_{j,l}$ and $s_{j,l+1}$. A conflict exists between paths, but it will not be detected.

In the above case, the maximum violation of $c$ occurs when $s_{j,l}$ and $s_{j,l+1}$ are a distance $2r$ apart (the maximum), and $s_{i,k}$ is equidistant to each of the spheres and as close as possible to the pair without intersecting either. This forms three mutually tangential spheres. To resolve this violation, we can scale the radius of all spheres in the space by some constant $\lambda > 1$. If large enough, the three spheres can remain mutually tangential while $s_{i,k}$ and $c$ no longer intersect. Note that $c$ retains its original radius, as $c$ represents the actual area of space-time through which the agent will travel. By geometry, $s_{i,k}$ is exactly tangential to $c$ when the radius of each sphere is scaled by $\lambda^* = \frac{1}{\sqrt{3}-1} \approx 1.366$. Thus, scaling $r$ by at least $\lambda^*$ guarantees no discretization error, i.e. all intersections relevant to continuous-time agent conflict will be detected and resolved by STCS. □

**Definition IV.2** (formulation space)**.** *We define the **formulation space** $\mathcal{F}$ of a MAMP conflict as the set of all possible path configurations that can be reached from some initial configuration by executing some sequence $\mathcal{V}$ of DV translations, where $\mathcal{V}$ can be feasibly constructed by the conflict search procedure given the formulation of the DV computation and path shift operations.*

**Theorem IV.2** (formulation-space completeness)**.** *If there exists a solution to* (1) *that also exists in $\mathcal{F}$, then STCS will find and return a feasible solution.*

*Proof.* We can prove the above using induction. First suppose that there exists some single DV $\vec{v}_n$ such that applying $s_n^o \mathrel{+}= \vec{v}_n$ resolves some previous conflict involving $s_n^o$ without generating any subsequent conflict; this implies a feasible solution. Next, suppose that there exists a second DV $\vec{v}_{n-1}$ such that applying $s_{n-1}^o \mathrel{+}= \vec{v}_{n-1}$ resolves some previous conflict involving $s_{n-1}^o$ *and* generates a subsequent conflict between $s_{n-1}^o$ and $s_n^o$, which can in turn be resolved through the previous statement; this also implies a feasible solution.

Generally, we can state that if $\vec{v}_i$ will yield a feasible solution following some additional sequence of translations $\{\vec{v}_{i+1}, \vec{v}_{i+2}, ..., \vec{v}_{n-1}, \vec{v}_n\}$, then $\vec{v}_{i-1}$ will also yield that same feasible solution, given $\vec{v}_i$ and the same additional sequence. Because DV translations and subsequent path shifts are applied in the same order in which they are computed and path shifts are simulated during the conflict search stage, a solution in the form of a DV sequence must yield an intersection-free STG. By nature, the depth-first conflict search procedure performs a complete search of the solution space.

By Theorem IV.1, an intersection-free STG implies a solution to MAMP, due to the absence of discretization error. Conflict resolution is applied at each iteration of agent planning, making the overall algorithm complete with respect to the formulation space. □

## V. EXPERIMENTAL RESULTS

We first evaluate STCS under motion planning tasks, and then verify the planned trajectories in realistic traffic simulation. We compare our algorithm to a baseline in term of various performance metrics.

### A. Simulation Setup

We simulate motion planning tasks where $N$ agents are each given starting and goal locations and must cooperatively plan trajectories within an environment containing $M$ static and dynamic obstacles, for $N \in [2, 4]$ and $M \in [0, 5]$. We implement STCS in Python to configure and solve each motion planning instance. The planned trajectories from each scenario are then executed in a traffic intersection setting using CARLA, an open-source autonomous driving simulator [23]. During CARLA evaluation, we verify the feasibility of solutions through adherence to car dynamics for control and spatial constraint of agent motion to the dimensions of the

intersection. In all experiments, the field size is $20\text{m} \times 20\text{m}$, and for all agents, we assign the radius $r = 3.5\text{m}$, the acceleration bound $a_i^{max} = \pm 3\text{m/s}^2$, and the path rigidity $\gamma_i = 10$. We assign the priority $\phi_i = 100$ for Agent 1 and $\phi_i = 1$ for all other agents.

For evaluation, we configure various scenarios that have either no obstacles, static obstacles only, dynamic obstacles only, or a mix of static and dynamic obstacles. We also consider scenarios involving non-connected vehicles, since this is a special case of dynamic obstacles that is particularly applicable to the CAV domain.

*1) Obstacle-Free:* We compare the performance of STCS to that of S2M2 in three obstacle-free scenarios.

- **Obstacle-Free 1 (F1):** Agent 1 travels from bottom to top. Agent 2 travels from left to right.
- **Obstacle-Free 2 (F2):** Agent 1 travels from bottom to top. Agent 2 travels from top to bottom. Agent 3 travels from left to right.
- **Obstacle-Free 3 (F3):** Agent 1 travels from bottom to top. Agent 2 travels from top to bottom. Agent 3 travels from left to right. Agent 4 travels from right to left.

*2) Static Obstacles:* We compare the performance of STCS to that of S2M2 in three static obstacle scenarios.

- **Static Obstacles 1 (S1):** Agent 1 travels from bottom to top. Agent 2 travels from top to bottom. A static obstacle is in the center of the field.
- **Static Obstacles 2 (S2):** Agent 1 travels from bottom to top. Agent 2 travels from left to right. One static obstacle is in the center of the field, and one static obstacle is in the center of each of the four quadrants of the field.
- **Static Obstacles 3 (S3):** Agent 1 travels from bottom to top. Agent 2 travels from left to right. Agent 3 travels from right to left. A static obstacle is in the center of the field.

*3) Dynamic Obstacles:* We evaluate the performance of STCS in three dynamic obstacle scenarios.

- **Dynamic Obstacles 1 (D1):** Agent 1 travels from bottom to top. Agent 2 travels from top to bottom. A dynamic obstacle travels from top left to bottom right.
- **Dynamic Obstacles 2 (D2):** Agent 1 travels from bottom to top. Agent 2 travels from left to right. Agent 3 travels from right to left. A dynamic obstacle travels from top left to bottom right.
- **Dynamic Obstacles 3 (D3):** Agent 1 travels from bottom to top. Agent 2 travels from left to right. Agent 3 travels from right to left. One static obstacle is in the center of the first quadrant of the field, and a second static obstacle is in the center of the third quadrant. A dynamic obstacle travels from top left to bottom right.

*4) Non-Connected Vehicles:* We evaluate the performance of STCS in three non-connected vehicle scenarios.

- **Non-Connected Vehicles 1 (N1):** Agent 1 travels from bottom to top. Agent 2 travels from left to right. One non-connected vehicle travels from right to left.
- **Non-Connected Vehicles 2 (N2):** Agent 1 travels from bottom to top. Agent 2 travels from right to left. One non-connected vehicle travels from top to bottom. A second non-connected vehicle travels from left to right.
- **Non-Connected Vehicles 3 (N3):** Agent 1 travels from bottom to top. Agent 2 travels from left to right. Agent 3 travels from right to left. One non-connected vehicle travels from top to bottom.

### *B. Baseline and Evaluation Metric*

To assess solution quality, we compute *suboptimality ratios* for various metrics, which we define as the ratio between the observed value of the metric and a lower bound on its optimal value. In particular, we measure total distance, which is simply the sum of the distances traveled by all agents in the solution, and makespan, which is the time that the last agent reaches its goal. We obtain a lower bound on the optimal total distance of a solution using the sum of L2 norms between each agent's starting point and its goal. We obtain a lower bound on the optimal makespan of a solution by solving for time using basic kinematics formulas, assuming that traveled distance is the L2 norm between an agent's starting point and its goal and that the agent has a constant acceleration $a_i^{max}$. After computing suboptimality ratios for each of these metrics, we compute the *overall* suboptimality ratio as the average of these two ratios. A lower overall suboptimality ratio implies higher solution quality, and a value of 1 is the minimum. Because we use lower bounds to compute these metrics, a value of 1 is often unattainable while preserving solution feasibility.

We compare the performance of STCS to that of S2M2, a MAMP algorithm proposed in [22]. While S2M2 has proved to excel among nonlinear dynamics and bounded disturbances and in settings involving static obstacles, the algorithm has not shown compatibility with dynamic obstacles. Thus, we evaluate the performance of S2M2 in scenarios that are obstacle-free or that contain only static obstacles. We configure all parameters unique to S2M2 to the default values provided by the authors. We measure each algorithm's runtime in each scenario as the average runtime over 20 trials. All experiments were run on a desktop computer with an AMD Ryzen 5 2600X CPU and 16GB RAM.

### *C. Motion Planning Evaluation*

We evaluate the performance of STCS in motion planning with respect to runtime and suboptimality ratio in the twelve outlined scenarios, comparing it to S2M2 in obstacle-free and static-obstacle scenarios. Figure 4 indicates that STCS and S2M2 provide similar runtime values in obstacle-free scenarios. Figure 5 indicates that the algorithms also exhibit comparable solution quality in these scenarios, as implied by the overall suboptimality ratio metric. Likewise, Figure 6 and Figure 7 extend these observations to scenarios involving static obstacles. Table I and Table II demonstrate that the performance of STCS with respect to both runtime and solution quality scales well to scenarios involving dynamic obstacles and non-connected vehicles, respectively.

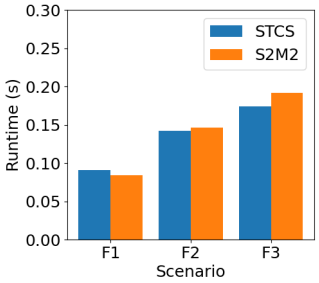



Fig. 4: Average runtime, in seconds, of STCS and S2M2, as measured over 20 trials in each of the three obstacle-free scenarios.

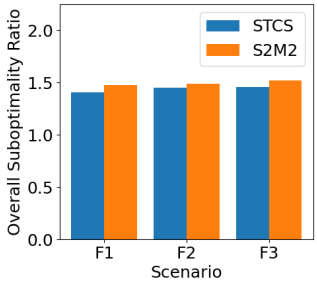



Fig. 5: Overall suboptimality ratio of STCS and S2M2, as measured in each of the three obstacle-free scenarios.

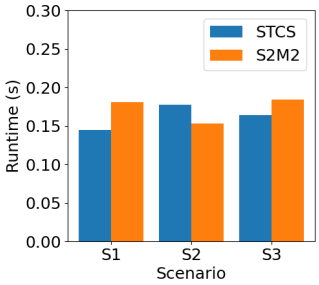



Fig. 6: Average runtime, in seconds, of STCS and S2M2, as measured over 20 trials in each of the three static obstacle scenarios.

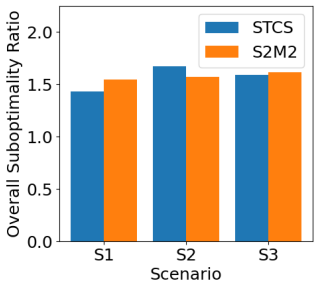



Fig. 7: Overall suboptimality ratio of STCS and S2M2, as measured in each of the three static obstacle scenarios.

Thus, STCS matches state-of-the-art performance with respect to both runtime and solution quality in obstacle-free and static-obstacle environments. Furthermore, STCS extends this performance to settings involving dynamic obstacles.

### *D. CARLA Evaluation*

We further evaluate STCS in the setting of traffic intersections using the CARLA autonomous vehicle simulator. We use a four-way, two-lane uncontrolled intersection setting. We assign cars for each agent and non-connected vehicle, and bicycles and pedestrians for each static or dynamic obstacle. Furthermore, we introduce spatial constraints on each planner to ensure that vehicles stay within the bounds of the intersection during execution. In each scenario, we evaluate the ability of STCS and S2M2 to produce a solution that can be feasibly executed in CARLA. A full solution generated by STCS for Scenario **N2** is shown in Figure 8.

As indicated by Table III, the MAMP solutions produced by STCS were feasible in the traffic intersection setting for all obstacle-free, static-obstacle, dynamic-obstacle, and non-connected-vehicle scenarios. On the other hand, S2M2 occasionally failed to produce feasible solutions when given the spatial constraints of this setting, specifically in the more challenging scenarios **F3** and **S3**. Thus, we experimentally demonstrate that STCS expands upon the current state of the art by solving complex traffic intersection scenarios involving pedestrians and non-connected vehicles, and by maintaining continuous-time feasibility and formulation-space completeness in all cases. In particular, we show that the latter property enables solutions to be found more consistently in challenging scenarios, given the spatial constraints of the traffic intersection setting.

| | D1 | D2 | D3 |
|---|---|---|---|
| **Runtime** (s) | 0.231 | 0.246 | 0.212 |
| **Distance Suboptimality** | 1.059 | 1.094 | 1.128 |
| **Makespan Suboptimality** | 1.825 | 2.424 | 2.324 |
| **Overall Suboptimality** | 1.442 | 1.759 | 1.726 |

TABLE I: Performance evaluation of STCS in the three dynamic obstacle scenarios.

| | N1 | N2 | N3 |
|---|---|---|---|
| **Runtime** (s) | 0.155 | 0.224 | 0.237 |
| **Distance Suboptimality** | 1.561 | 1.146 | 1.785 |
| **Makespan Suboptimality** | 1.657 | 2.281 | 1.848 |
| **Overall Suboptimality** | 1.609 | 1.713 | 1.817 |

TABLE II: Performance evaluation of STCS in the three non-connected vehicle scenarios.

## VI. Conclusion

In this work we presented STCS, a novel discrete-time formulation and decoupled algorithm for multi-agent motion planning. We theoretically proved the continuous-time feasibility and formulation-space completeness of STCS. We experimentally validated the algorithm's performance in various scenarios with application to unsignalized traffic intersections, demonstrating that we expand upon the current state of the art with regard to dynamic obstacle compatibility and consistency in constrained settings, while maintaining runtime and solution quality. As STCS is a novel approach to MAMP, there still remains much work to be done. In future work, we intend to further explore optimization techniques and formally prove suboptimality bounds and adherence to motion constraints.

## References

[1] L. Ye and T. Yamamoto, "Evaluating the impact of connected and autonomous vehicles on traffic safety," Physica A: Statistical Mechanics and its Applications, vol. 526, p. 121009, 2019.

[2] A. Reyes-Muñoz and J. Guerrero-Ibáñez, "Vulnerable road users and connected Autonomous Vehicles Interaction: A Survey," Sensors, vol. 22, no. 12, p. 4614, 2022.

[3] A. Talebpour and H. S. Mahmassani, "Influence of connected and autonomous vehicles on traffic flow stability and throughput," Transportation Research Part C: Emerging Technologies, vol. 71, pp. 143–163, 2016.

[4] M. Khayatian, M. Mehrabian, E. Andert, R. Dedinsky, S. Choudhary, Y. Lou, and A. Shirvastava, "A survey on intersection management of Connected Autonomous Vehicles," ACM Transactions on Cyber-Physical Systems, vol. 4, no. 4, pp. 1–27, 2020.

[5] F. Fan, "Study on the cause of car accidents at intersections," OALib, vol. 05, no. 05, pp. 1–11, 2018.

[6] Á. Madridano, A. Al-Kaff, D. Martín, and A. de la Escalera, "Trajectory planning for multi-robot systems: Methods and applications," Expert Systems with Applications, vol. 173, p. 114660, 2021.

[7] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," Proc. Conf. AAAI Artif. Intell., vol. 27, no. 1, pp. 1443–1449, 2013.

[8] G. Wagner and H. Choset, "Subdimensional expansion for Multirobot Path Planning," Artificial Intelligence, vol. 219, pp. 1–24, 2015.

| | STCS | S2M2 | | STCS |
|---|---|---|---|---|
| **F1** | ✓ | ✓ | **D1** | ✓ |
| **F2** | ✓ | ✓ | **D2** | ✓ |
| **F3** | ✓ | ✗ | **D3** | ✓ |
| **S1** | ✓ | ✓ | **N1** | ✓ |
| **S2** | ✓ | ✓ | **N2** | ✓ |
| **S3** | ✓ | ✗ | **N3** | ✓ |

TABLE III: Feasibility of solutions generated by STCS and S2M2 during CARLA evaluation.

Fig. 8: Trajectories generated by STCS for Scenario **N2** in CARLA, where the red and yellow belong to Agents 1 and 2, respectively, and the blue and green paths belong to non-connected vehicles.


[9] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," Artificial Intelligence, vol. 219, pp. 40–66, 2015.

[10] J. Yu and S. M. LaValle, "Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics," in IEEE Transactions on Robotics, vol. 32, no. 5, pp. 1163-1177, Oct. 2016, doi: 10.1109/TRO.2016.2593448.

[11] C. Liu, C. -W. Lin, S. Shiraishi and M. Tomizuka, "Distributed Conflict Resolution for Connected Autonomous Vehicles," in IEEE Transactions on Intelligent Vehicles, vol. 3, no. 1, pp. 18-29, March 2018, doi: 10.1109/TIV.2017.2788209.

[12] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," Int. J. Rob. Res., vol. 24, no. 4, pp. 295–310, 2005.

[13] J. van den Berg, Ming Lin and D. Manocha, "Reciprocal Velocity Obstacles for real-time multi-agent navigation," 2008 IEEE International Conference on Robotics and Automation, 2008, pp. 1928-1935, doi: 10.1109/ROBOT.2008.4543489.

[14] W. Wu, S. Bhattacharya and A. Prorok, "Multi-Robot Path Deconfliction through Prioritization by Path Prospects," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 9809-9815, doi: 10.1109/ICRA40945.2020.9196813.

[15] P. Velagapudi, K. Sycara and P. Scerri, "Decentralized prioritized planning in large multirobot teams," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 4603-4609, doi: 10.1109/IROS.2010.5649438.

[16] S.-K. Huang, W.-J. Wang, and C.-H. Sun, "A path planning strategy for multi-robot moving with path-priority order based on a generalized Voronoi diagram," Appl. Sci. (Basel), vol. 11, no. 20, p. 9650, 2021.

[17] K. C. Wang and A. Botea, "MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees," J. Artif. Intell. Res., vol. 42, pp. 55–90, 2011.

[18] S. Tang and V. Kumar, "A complete algorithm for generating safe trajectories for multi-robot teams," in Springer Proceedings in Advanced Robotics, Cham: Springer International Publishing, 2018, pp. 599–616.

[19] J. Kottinger, S. Almagor, and M. Lahijanian, "Conflict-based search for multi-robot motion planning with kinodynamic constraints," arXiv [cs.RO], 2022.

[20] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, 'Multi-Agent Pathfinding with Continuous Time'. arXiv, 2019.

[21] D. Dayan, K. Solovey, M. Pavone, and D. Halperin, "Near-optimal multi-robot motion planning with finite sampling," in 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 9190–9196.

[22] J. Chen, J. Li, C. Fan, and B. Williams, "Scalable and safe multi-agent motion planning with nonlinear dynamics and bounded disturbances," arXiv [cs.RO], 2020.

[23] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," arXiv [cs.LG], 2017.