Volume xx (*200y*), *Number z, pp. 1–11*

Multi-Agent Path Planning with Asymmetric Interactions In Tight Spaces

V. Modi[1] Y. Chen [1], A. Madan [1], S. Sueda [2], and D. I. W. Levin [1]

[1]University of Toronto, Toronto, Canada
[2]Texas A&M University, College Station, TX

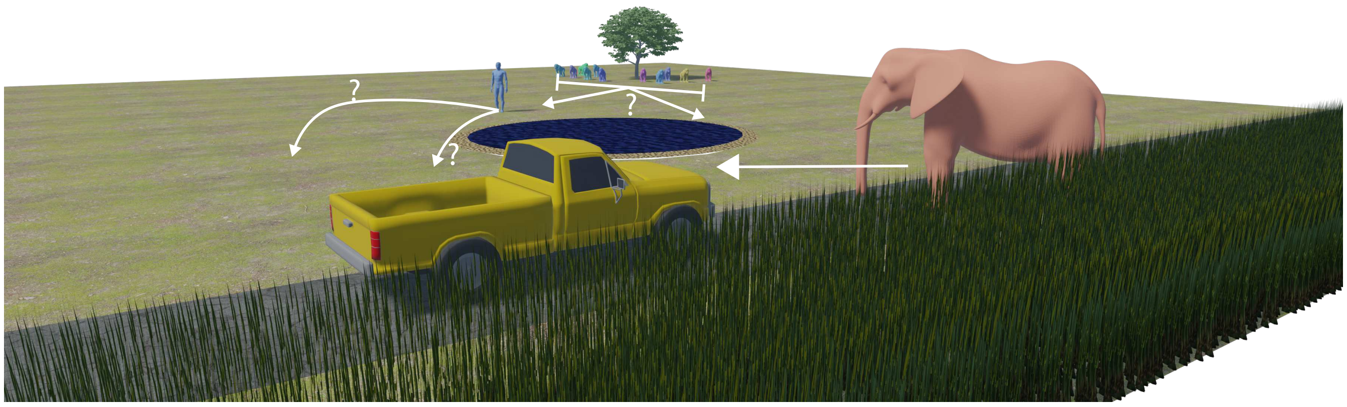


Figure 1: Safari Escape

Abstract

By starting with the assumption that motion is fundamentally a decision making problem, we use the world-line concept from Special Relativity as the inspiration for a novel multi-agent path planning method. We have identified a particular set of problems that have so far been overlooked by previous works. We present our solution for the global path planning problem for each agent and ensure smooth local collision avoidance for each pair of agents in the scene. We accomplish this by modeling the trajectories of the agents through 2D space and time as curves in 3D. Global path planning is solved using a modified Djikstra's algorithm to ensure that initial trajectories for agents do not intersect. We then solve for smooth local trajectories using a quasi-Newton interior point solver, providing the trajectory curves with a radius to turn them into rods. Subsequently, resolving collision of the rods ensures that no two agents are in the same spatial position at the same time. This space-time formulation allows us to simulate previously ignored phenomena such as highly asymmetric interactions in very constrained environments. It also provides a solution for scenes with unnaturally symmetric agent alignments without the need for jittering agent positions or velocities.

1. Introduction

On a hot dry day in Kruger National Park, an empty truck idles on the side of a road. Sam, the driver of the truck, has wandered a hundred meters off the road in an attempt to take a picture of a tree with ten baboons. The baboons suddenly jump out of the tree and charge towards Sam. Sam panics and starts running back to the truck; however, coincidentally an elephant wandering in the area is on a path perpendicular to Sam's, between him and the truck. How will Sam get back to the truck safely while outrunning the baboons and avoiding collision with the elephant? What are the paths of the twelve agents in the scene: Sam, the elephant and the ten baboons?

The problem above poses many challenges to a multi-agent path planning algorithm. First Sam must anticipate collisions ahead of time, in order to move quickly and efficiently to their truck. Second, the three groups of agents, Sam, the baboons and the elephant have dramatically different masses and behaviors. For instance while Sam seeks to avoid all animals, the massive elephant is untroubled, and will stubbornly continue on its path. Finally, geographic features such as additional trees and ponds can lead to a highly constrained environment. Existing state-of-the-art crowd simulation methods struggle to compute anticipatory agent paths in constrained environments when asymmetric interactions are involved (Table 1) making them ill-suited for application in planning problems such as the example given above.

We propose a new multi-agent path planning algorithm well suited for these problems. Our model directly optimizes the space-time trajectories of all agents which allows for per-agent physical and psychological characteristics and smooth anticipatory trajectories. A novel, differentiable space-time repulsive energy ensures collision free trajectories. Using our approach, Sam arrives safely at the truck, escaping the baboons and avoiding the elephant.

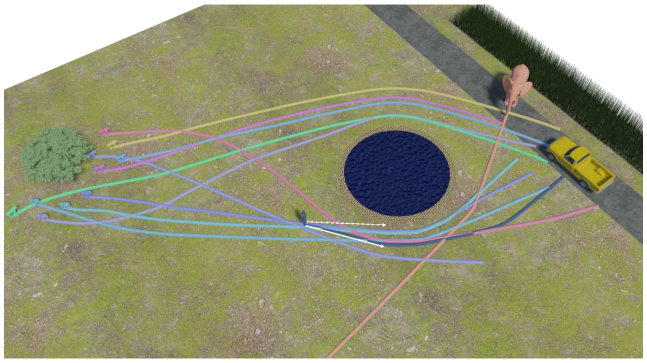


Figure 2: Sam needs to escape ,from the baboons while avoiding the elephant, whereas the elephant is unconcerned with the other agents in the scene.

2. Related Work

Successful multi-agent path planning requires an algorithm to both correctly model the behavior of independent agents as well as their interactions. A common approach is to apply a dynamics model based on Newton's second law of motion which is integrated over time to produce plausible agent trajectories. Psychological and social characteristics of the group can be incorporated into the dynamics equations as social forces [POSB05; HM98; WLJT17]. Intra-agent forces, such as those that handle collision avoidance are added through a variety of means, and we can partition the space of successful approaches based on locality of their models in both space and time. Approaches such as [GKLM11] incorporate psychological factors into an underlying dynamics model, but it is impossible to tell apart the psychological traits of the agents by simply observing the simulation. Our method makes the impact of agent characteristics on the trajectory very obvious, while providing a standalone local and global dynamics model for the scene.

In local methods, agent decision making requires information only local in space and in time. Local collision avoidance methods [VGLM11; GNCL14; WLP16] compute collision response using local information in space and time (the planning horizon can be as small as a single time step). These methods struggle to generate smooth anticipatory collision responses. Implicit Crowds [KSNG17] attempts to overcome this difficulty by introducing a time-to-collision potential to the crowd dynamics. This gives agents richer space-time information on which to act; however this energy is effectively local, computed from the current state of the system (position and velocity). NH-TTC [DKG20] improves upon prior work by utilizing longer planning horizon, geometrically represented by curves in space. Intersection checks between these curves allow agents to react to collisions likely to occur in the near future. Similarly, vision based methods, such as [OPOD10; DMC*17] provide an anticipatory collision avoidance model, but with extremely limited path planning capabilities and no guarantee of smooth agent motion. Data-driven aproaches such as [CC14] use an underlying state-action graph created via external data to generate trajectories. However, these trajectories are highly data-dependent, do not factor in environmental constraints, seem to operate in sparse crowds and must be used in conjunction with some other higher level global path planner.

An alternative approach to more local methods is to extend the collision response globally in space using a fluid like pressure solve [Hug02; TCP06; NGCL09]. However because these methods only consider the configuration of the system (position and velocity of each agent) at a single time, their ability to produce smooth anticipatory collision response, especially in a sparser setting, is reduced. Continuum Crowds stands out as one of the only methods that incorporates both a global planner through Djikstra's search and local collision avoidance through a fluid-like pressure solve. Another option is to handle local collision avoidance using RVO or Social Forces, as done by the data-driven method [TYK*09]. Optionally, one might use [HKHL13], another data-driven method which uses a purely stochastic collision avoidance method. None of these approaches solve the problems of handling tight environmental constrains or asymmetric agent interactions. In fact, an adjacent field of research, that involves measuring the "correctness" of various crowd models ([GVL*12; WJO*14]) also fails to test for asymmetric agents and behavior in complex environments.

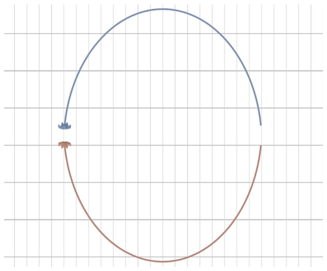


Figure 3: The repulsive curves energy forces agents to be maximally far away from one another which is not an intuitive behavior.

Finally, while not strictly designed for multi-agent path planning, Repulsive Curves [YSC21] can potentially be used for agent planning. While it yields impressive results in 3D curve untangling, when applied to multi-agent path planning it has several drawbacks. First, in unconstrained environments, Repulsive Curves will maximally repell agents away as shown in Figure 3. Secondly, like other previous methods, there is no straightforward way to model asymmetric agents. Lastly, Repulsive Curves uses random "jitter" to ensure that no trajectories initially overlap. From our own experiments, we have observed that this is not sufficient to guarantee non-overlapping initial trajectories. Meanwhile, our space-time Djikstra's approach is guaranteed to create initially non-intersecting trajectories, but is not sufficient for overcoming the other limitations of Repulsive Curves.

In this paper we present a multi-agent path planner that computes its response *globally in space and time*. Rather than formulating our multi-agent planner as an initial value problem, we instead take a space-time optimization approach.

Space-time trajectory optimization [WK88] was initially applied to keyframe interpolation, with subsequent methods such as [PSE*00] allowing manual editing of object trajectories and [SKF08] allowing interactive motion correction and synthesis using graph search methods. The concept of using space-time graph search methods for collision avoidance is furthered in [LLKP11] for a single agent and [SKH*11] for small groups. For larger situations, standard space-time approaches would be computationally prohibitive. Our method builds on these prior methods by expanding the look-ahead globally and ensuring smooth paths by collision resolution on the entire trajectories rather than using a limited space-time graph search approach.

Our method treats each agent as an individual space time curve with only three requirements: a starting position and time and an ending position (at indeterminate time). We use a globally supported, differentiable LogSumExp smooth distance in space-time to guarantee collision free trajectories and solve the resulting problem using an interior-point technique. For small to medium scale crowds, our method outperforms current state-of-the-art methods in simple scenarios (Figure 4a and Figure 4b) and more complicated, constrained environments. Our agents exhibit anticipatory behaviors such as slowing and waiting to let others pass and makes no assumptions about agents having identical mass or other physical properties. This enables planning of intricate scenes with environmental constraints such as the one described in our introduction.

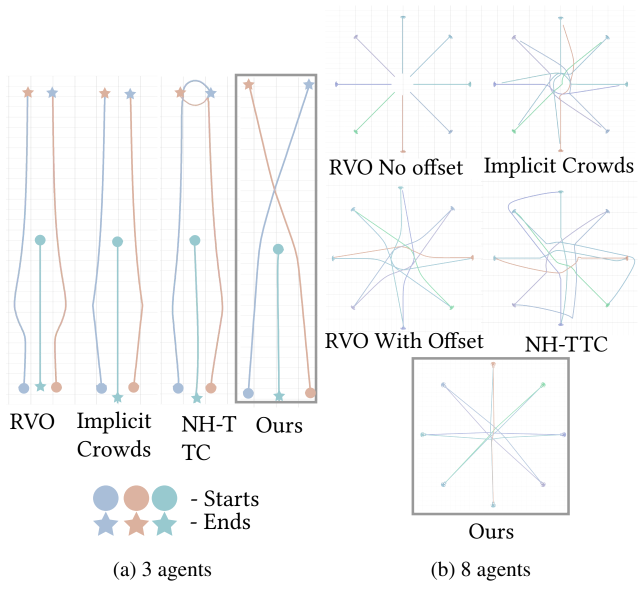


Figure 4: On the left, circles denote the start position of the agents and starts denote the desired end position. Only our method lets agents reach their desired ends without any locking. On the right, each agent wants to traverse to the directly opposite end of the circle. Our method leads to the most efficient trajectories.

3. Method

At a high level we are influenced by the notion of correlated equilibrium. In a correlated equilibrium solution to a non-cooperative game, an "oracle" chooses a strategy for each player, and no player has any reason to deviate from the chosen strategy assuming others do not deviate either. The result is an equilibrium solution which maximizes collective utility.

Our method acts as the oracle of the scene and plans agent trajectories in a way that collectively maximizes the utility of the entire scene. Additionally, unlike previous approaches which ‘pre-set’ trajectories for agents (often done manually), we allow our agents to find their own utility-maximizing trajectories. Lastly, real life agents have different sizes, masses, and personalities, which affect the agents’ utilities, and therefore its path as well. Our local-global path planning approach allows for these nuances.

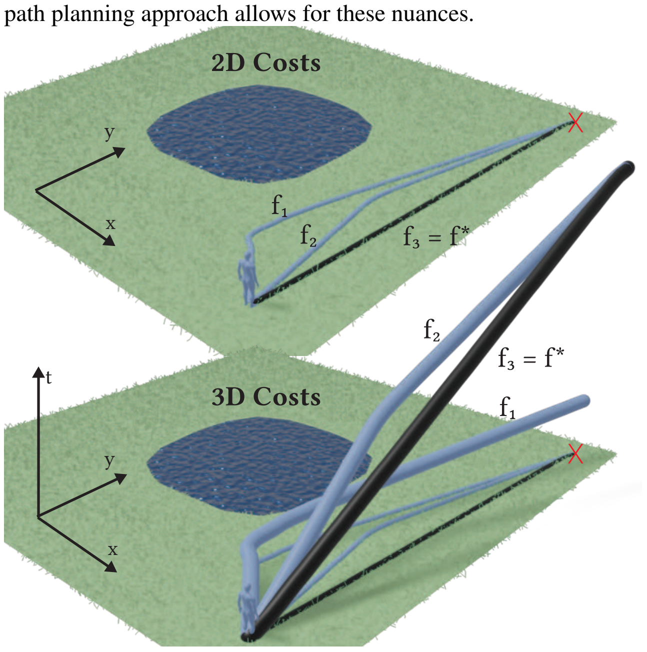


Figure 5: Top: A path through space can be represented as a curve embedded in R^2. Every path has a scalar cost (utility) value. For a cost function that minimizes distance travelled, c_1 and c_2 are in-optimal paths from the start to the goal (red X), but c_3 is optimal. Bottom: A path through space and time can be represented as a curve in R^3. A space-time rod (as shown here) is simply a 3D curve with a collision radius.

3.1. Paths Map To Utility

As shown in Figure 5, for an agent, each path from the start location to the end location through space maps to some scalar utility value. For example, if the agent’s utility minimized distance traveled, a straight line from start to end would prove to be the optimal path. In Figure 5, the 2D x-y plane describes the domain of spatial motion for agent paths. We make the additional observation that agents do not simply move through space, but also move through time which is denoted by the vertical z-axis. So, in 3D space-time, the agent’s motion through space (x, y) and time (t) is described by a 3D curve which corresponds to a utility. For any given moment in

Features	Ours	(1) RVO	(2) IC	(3) NH-TTC	(4) RC	(5) CC	(6) STPL	(7) AABS
Asymmetric Interactions	Y	N	P	P	N	N	N	N
Extremely Constrained Environments	Y	N	N	N	Y	N	P	Y
Smooth Local Interactions	Y	N	Y	N	Y	N	Y	N
Intuitive Control Parameters	Y	Y	N	P	N	N	N	Y
Multiple Agents	Y	Y	Y	Y	Y	Y	N	Y
Code Available Online	Y	Y	Y	Y	Y	N	N	N

Table 1: Compare (1) RVO ([VGLM11]), (2) Implicit Crowds ([KSNG17]), the brand new time-to-collision method (3), NH-TTC ([DKG20]), (4) Repulsive Curves ([YSC21]), (5) Continuum Crowds ([TCP06]), (6) Space Time Planning With Parameterized Locomotion Control ([LLKP11]), and (7) Modular Framework for Adaptive Agent Base Steering ([SKH*11]). Y - feature is available, N - feature is not possible, P - feature might be possible, but not demonstrated.

time, the projection of the curve onto the x-y plane gives us the agent's location.

Let us consider a simple case where there are no other agents in the scene and the terrain is flat and free of obstacles. The agent, indexed henceforth by subscript a, wishes to maximize its utility (by minimizing cost, $\tilde{\Psi}$); the tilde symbol indicates the variable or function is continuous, not discrete. Lower case variables indicate a single agent. Upper case indicates the variable aggregates all agents. The motion of the agent through space and time denoted by the agent's 3D space-time curve $\tilde{\mathbf{s}}_a$ embedded in $R^{x,y,t}$ requires constraints. Fortunately all our constraints are linear, so we lump them together into $\tilde{B}$ for now. Put together, the full optimization for a single agent a is

$$f_a^* = \min \int_{\tilde{\mathbf{s}}_a} \tilde{\psi}(\tilde{\mathbf{s}}_a) d\tilde{\mathbf{s}}_a \tag{1a}$$

$$s.t. \qquad \tilde{B}\tilde{\mathbf{s}}_a \leq \mathbf{0}. \tag{1b}$$

The cost, Equation 1a, for agent a is integrated over the trajectory of the agent ($\tilde{\mathbf{s}}_a$). The specific nature of the cost function determines the agents' behavior based on its characteristics. An agent might have a preferred walking speed, or a stubbornness factor, or a radius of comfort, all of which (and more) can be encoded into components of $\tilde{\psi}$. In order to solve this optimization, we must first discretize the agent's trajectory into the discrete curve $\mathbf{s}_a$ shown in Figure 6. We also discretize the cost $\tilde{\psi}$ into separate intra-agent costs, which solely affect the path of one individual, and interaction costs, which can affect multiple agents.

3.2. Discretizing

Our agent's cost function takes in the agent's **discrete** 3D space-time curve $\mathbf{s}_a$ as input and outputs a scalar cost for that path, f_a. We descretize the agent's continuous $\tilde{\mathbf{s}}$ into a piecewise linear curve described by $n+1$ nodes $\mathbf{s}_a = [(x_a^0, y_a^0, t_a^0), \ldots, (x_a^i, y_a^i, t_a^i), \ldots., (x_a^n, y_a^n, t_a^n)]$ for nodes $i = 0..n$ connected sequentially by edges. We must also discretize our constraints. First, even though time is a variable in our formulation, Equation 2b ensures the agent cannot move backwards in time. Second, the agent has a start location (x_a^0, y_a^0) and start time t_a^0 denoted in Equation 2c. Third, Equation 2d sets a goal or an end location (x_a^n, y_a^n). Lastly, the agent cannot take an infinite amount of time, so Equation 2e bounds the agent by a max time T_a^{max}. Putting all of this together, we can re-write the optimization for agent a with the discrete generalized cost function ψ as

$$f_a^* = \min \sum_{i=0}^{n} \psi(\mathbf{s}_a) \tag{2a}$$

$$s.t. \qquad t_a^i \leq t_a^{i+1} \tag{2b}$$

$$(x_a^0, y_a^0, t_a^0) = (\mathbf{x_a}, \mathbf{y_a}, \mathbf{t_a})^{\mathbf{start}} \tag{2c}$$

$$(x_a^n, y_a^n) = (\mathbf{x_a}, \mathbf{y_a})^{\mathbf{end}} \tag{2d}$$

$$t_a^n \leq T_a^{max}. \tag{2e}$$

Now with a template for our optimization problem, we can replace the generalized cost function with specific discrete costs. Our specific cost functions are derived from observation of real behavior. These behavioral observations are divided into two categories, intra-agent costs, and interactions costs. Intra-agent cost functions only look at one agent's path at a time. Interaction cost functions include avoiding collisions with other agents, collisions with static obstacles such as walls or furniture, grouping behavior within friends, asymmetric behavior based on the mass, size of the agents, or other characteristics.

3.3. Intra-Agent Costs

Since these costs apply to a single agent, we calculate intra-agent costs for arbitrary agent a and later we show how to sum the costs over the entire scene. The path of our agent, $\mathbf{s}_a$, is comprised of of $n+1$ nodes indexed by $i = 1..n$. Each node $(x^i(\mathbf{s}_a), y^i(\mathbf{s}_a), t^i(\mathbf{s}_a))$ is comprised of the agent's spatial (x, y) coordinates and time (t) coordinates. For all cost functions, agent a also has constant weighting terms, K_a, as well as constant characteristics such as mass, m_a, and preferred end time, T_a^p.

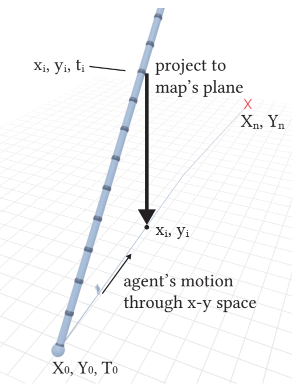


Figure 6: This space-time rod describes a discrete space-time trajectory curve made up of many nodes. Our agent moves along the projection of the rod from its start to end, taking into account the boundary constraints of the problem.

3.3.1. Intra-Agent Kinetic Cost

Our agent a will try to minimize energy expenditure. Given

an agent's space-time curve $\mathbf{s}_a$,
the kinetic energy cost of the agent over the curve can be written as

$$C_a^K(\mathbf{s}_a, K_a^K) = K_a^K \sum_{i=0}^{n} \frac{1}{2} m_a \frac{(\Delta \mathbf{x}_a^i)^T (\Delta \mathbf{x}_a^i)}{(\Delta t_a^i)^2} (\Delta t_a^i) \quad (3)$$

$$= K_a^K \sum_{i=0}^{n} \frac{1}{2} m_a \frac{(x^{i+1}(\mathbf{s}_a) - x^i(\mathbf{s}_a))^2 + (y^{i+1}(\mathbf{s}_a) - y^i(\mathbf{s}_a))^2}{t^{i+1}(\mathbf{s}_a) - t^i(\mathbf{s}_a)} \quad (4)$$

where Equation 3 is the kinetic energy $\frac{1}{2}mv^2$ for curve segment $e_a^i = [x_a^i, y_a^i, t_a^i, x_a^{i+1}, y_a^{i+1}, t_a^{i+1}]$ integrated over total time travelled $\Delta t_a^i = t_a^{i+1} - t^i$. This simplifies into Equation 4 where K_a^K is the agent-wise weighting coefficient, m_a is the mass of the agent and $\mathbf{s}_a$ is the discretized path curve (our input variable) made up of $n+1$ nodes.

3.3.2. Intra-Agent Acceleration Cost

In order to penalize acceleration in agent a's trajectory, the acceleration cost

$$C_a^A(\mathbf{s}_a, K_a^A) = K_a^A \sum_{i=1}^{n-1} \frac{1}{2} (\theta_a^i)^2 \quad (5)$$

measures the curvature of $\mathbf{s}_a$ through the discretized acceleration cost where angle θ is the angle between two piece-wise linear segments of the curve computed using the stable arctan function $arctan2$. The angle is $\theta_a^i = arctan2(\frac{\|(\mathbf{s}_a^{i+1} - \mathbf{s}_a^i) \times (\mathbf{s}_a^i - \mathbf{s}_a^{i-1})\|}{(\mathbf{s}_a^{i+1} - \mathbf{s}_a^i)^T (\mathbf{s}_a^i - \mathbf{s}_a^{i-1})})$ where $\mathbf{s}_a^i = [x^i, y^i, t^i]$ for each node in the curve.

3.3.3. Intra-Agent Preferred End Time Cost

Any agent a has a preferred end time, T_a^p at which they expect to reach the end position. Sometimes this is the same as the max end time T_a^{max} by when the agent is **required** to be at the end positions, but sometimes the preferred end time T_a^p can be sooner. Deviation from the preferred end time is modeled as a quadratic cost

$$C_a^T(\mathbf{s}_a, T_a^p, K_a^T) = K_a^T \frac{1}{2} (t_a^n - T_a^p)^2 \quad (6)$$

incentivizing the agents to arrive at their preferred end time T_a^p by keeping the actual end time for each agent, t_a^n, close to T_a^p.

3.3.4. Regularizing Cost

We find that adding a regularizing term to penalize extremely short time segments improves the overall quality of the paths by reducing near instantaneous motions in time. To penalize very fast agent motion, we use the regularizing term

$$C_a^R(\mathbf{s}_a, K_a^R) = K_a^R \sum_{i=0}^{n} \left(\frac{\frac{t_a^n}{n}}{t_a^{i+1} - t_a^i} \right). \quad (7)$$

This is important because it allows us to feed the solver an initial space-time curve such as the ones in Figure 11 with many near instantaneous agent motions, and the solver is able to optimize the final space-time curve to a much more reasonable trajectory.

3.4. Interaction Cost

Interactions include agent-environment interactions and agent-agent interactions for which we must introduce a new arbitrary agent indexed by b where $a \neq b$. All our interactions depend on each agent's 'radius of comfort' (or collision radius), denoted by r_a, r_b for agents a and b. The collision radius might be based on size, or a combination of size and personality: for example, people stay away from angry people. We encode these agent characteristics into our cost functions by extruding circle with radius r_a along the agent's path $\mathbf{s}_a$ thus forming a rod. In a scene with multiple agents, as long as no two rods are intersecting, the scene is collision free as shown in Figure 7. The collision radius ensures that no two agents are in the same place at the same time. So even though the forces between the two space-time rods might be symmetric, the agents' response to these forces (change in path) leads to asymmetric interactions. For static object interactions, the environment boundary is encoded into boundary vertices b_v and boundary elements b_e. Let us examine how we detect which rods are intersecting and how we deal with our three interaction types: agent-agent collisions, agent-agent groupings (friendships), and agent-environment collisions.

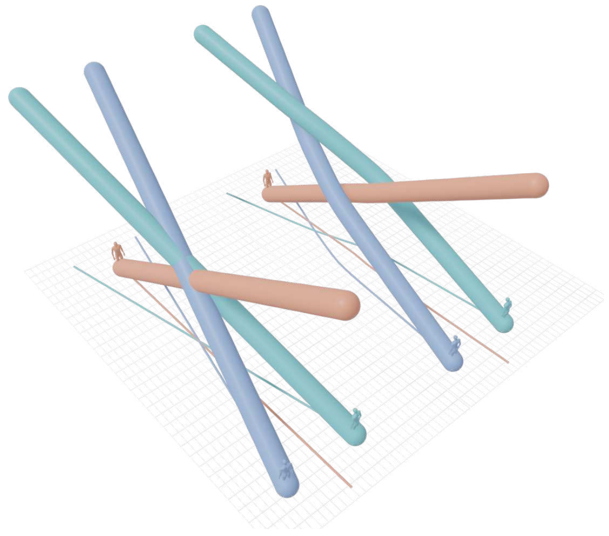


Figure 7: Resolving intersections between agents space-time rods resolves collisions in the scene since no agent shares the same x, y, t location at any point in the scene.

3.4.1. Collision Interaction Cost

Given a pair of agents a, b with 3D space-time curves $\mathbf{s}_a, \mathbf{s}_b$ and collision radii r_a, r_b, we densely sample each space-time curve uniformly. Next, we calculate the minimum smooth distances between the upsampled centerlines using the method described in subsection 3.5. As long as the smooth minimum distance d between the upsampled rods $u_a = \text{upsample}(\mathbf{s}_a), u_b = \text{upsample}(\mathbf{s}_b)$ is further apart than $r_a + r_b$, collisions will not occur. We implement this non-linear constraint on the agent paths as a log

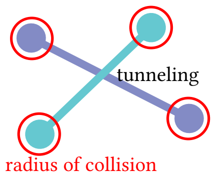


Figure 8: Tunneling artifact.

barrier energy:

$$C^C_{a,b}(\mathrm{d}(\alpha, \mathrm{upsample}(\mathbf{s}_a), \mathrm{upsample}(\mathbf{s}_b)), r_a, r_b, K^C_{a,b}) = -K^C_{a,b} \log(-(r_a + r_b) + \mathrm{d})). \tag{8}$$

where $K^C_{a,b}$ is the pair-wise weighting coefficient on the energy. The intuition behind a log barrier energy is explained in Figure 9. Upsampling the trajectories prevents tunneling artifacts (Figure 8) between edges during collision detection. The collision cost increases exponentially as the minimum distance between the two agent rods approaches the sum of the collision radii, so collisions are exponentially penalized. By summing this cost with the kinetic energy term, which incentivizes agents to move at a constant velocity, we get smooth local collision resolution as shown in Figure 7.

We employ two methods for sparsifying interactions. First, we use a 3D tree structure to store agent trajectory nodes (analogous to a Bounded Volume Hierarchy BVH) for broad-phase collision detection between trajectories. The tree-based broad phase reduces collision detection costs from $O(n^2)$ to $O(nlogn)$. Second, if two agents are known to be far apart, we entirely avoid the collision detection step between those two agents thus reducing costs to $O(n)$ in the best case. Using a combination of tree based broad phase along with manual denotation of interacting agents, we find that our method scales nearly linearly $O(n)$as the number of agents in the scene increases.

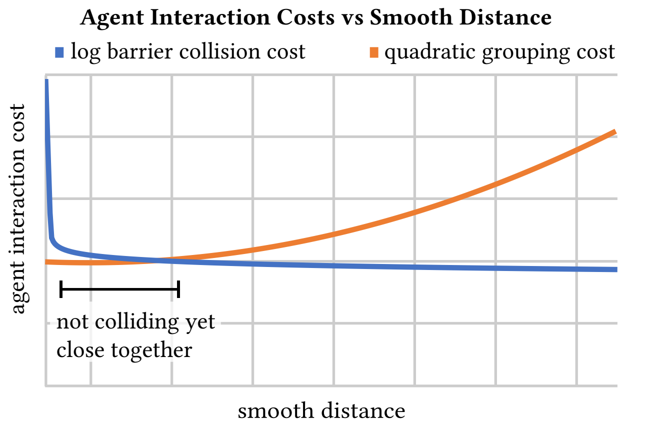


Figure 9: The log barrier energy quickly tends towards infinity as the minimum distance approaches the collision radius.

3.4.2. Grouping Interaction Cost

While the collision resolution term pushes agent paths apart, we introduce a grouping term which pulls paths together

$$C^G_{a,b}(\mathrm{d}(\alpha, \mathrm{upsample}(\mathbf{s}_a), \mathrm{upsample}(\mathbf{s}_b)), r_a, r_b, K^G_{a,b}) = K^G_{a,b}(\mathrm{d} - (r_a + r_b))^2. \tag{9}$$

This quadratic grouping term allows us to model scenarios in which friends going in the same direction tend to stick together as shown in Figure 17, or scenarios in which an agent needs to deliver a message to another agent who is out of the way from his final destination such as Figure 19. The weighting term $K^G_{a,b}$ controls the desire of the agents a, b to group together.

3.4.3. Agent-Environment Interaction

We surpass previous methods in three ways in terms of agent-environment interactions. Firstly, our method works on agents traversing highly constrained environments such as Figure 14a, Figure 14b, or Figure 18a. Secondly, our method allows agents to explore routes when multiple paths are available and choose the most optimal path using a novel modification to Djikstra's algorithm. Thirdly, our path planning step is topology aware, so this method will work on more fascinating terrain.

Environment maps are stored as 2D mesh files with vertices and faces. We pre-compute the vertices and edges around the boundaries of the map and any static obstacles into b_v and b_e as a one-time preprocessing step. Next, we compute the minimum smooth distance between the map boundary edges and the agent rod. Luckily since static obstacles are fixed in space for all time, we can ignore the third dimension and only compute the 2D minimum distance, $\mathrm{d}(\alpha, \mathbf{s}_a, b_v)$ from the rod's projection onto the map and the map's boundary edges. Next we pass the smooth distance d into the log barrier function

$$C^M_a(\mathrm{d}(\alpha, \mathrm{upsample}(\mathbf{s}_a), b_v), r_a) = -K^M_a log(-r_a + d) \tag{10}$$

where K^M_a is the energy coefficient. Like in the agent-agent collision function (Equation 8), we can use the collision radius r_a for each agent since the initial agent paths are sufficiently far enough away from static obstacles such that $d > r_a$. The coarseness of the map mesh does not impact the smoothness of agent path; however, Djikstra's algorithm requires two vertices on the map which correspond to the agent's start and end positions to generate the initial path. Therefore, we set the nearest vertices to the given start and end points as the boundary conditions for the solve.

3.5. Smooth Min Distance Implementation

The absolute minimum distance between agent paths is not a differentiable function. Therefore, we use a differentiable LogSumExp smooth minimum distance function to smoothly approximate the distance between two upsampled space-time curve vertices u_a and u_b. See the supplementary material for derivatives of the function. The smooth minimum distance function is

$$d(\alpha, u_a, u_b) = \frac{-1}{\alpha} \log\left(\sum_{i=0}^{n}\sum_{j=0}^{n} e^{-\alpha\|\mathbf{s}^i_a - \mathbf{s}^j_b\|}\right) \tag{11}$$

where the α term controls the numerical sensitivity of the distance which changes depending on the size of the environment mesh and the distances between the agents' path curves. A higher α leads to a more accurate minimum distance, but reduces numerical stability. Since this smooth minimum distance function is an underestimation of the true minimum distance, it is possible to get negative distances for very close objects.

We use 1 to update α during the solve to guarantee a usable (real, non-negative) smooth minimum distance. Our initial α_0 for the scene is as low as possible, but large enough to guarantee that both our log barrier interaction costs are real and defined. If the initial α_0 is too small, the distance underestimation is too great, and the log barrier costs are either complex or undefined, then the interior point solver will throw an error. We must choose a large enough

α_0 that the heuristic will provide an α that guarantees a real smooth distance values where $D > r_a + r_b$ to ensure real log barrier costs. For subsequent iterations, any real, non-negative smooth minimum distance is fine.

Algorithm 1 A heuristic to find a usable α to be used within the agent-collision and map interaction cost, gradient and hessian functions.

```
function ALPHAHEURISTIC($\alpha_0, \mathbf{s}_a, \mathbf{s}_b$)
    $\alpha \leftarrow \alpha_0$
    $D \leftarrow d(\alpha, \mathbf{s}_a, \mathbf{s}_b)$
    while $D \leq 0$ do
        $\alpha \leftarrow \alpha + 0.1 * \alpha_0$
        $D \leftarrow d(\alpha, \mathbf{s}_a, \mathbf{s}_b)$
    end while
    return $\alpha$
end function
```

3.6. Initial Paths

Using an interior point solver requires that we find feasible initial trajectories in which the agents do not collide with other agents or any obstacles. The supplementary material shows our failed experiments in generating initial paths based on shortest distances. Some global path planner is necessary to generate feasible initial paths. Initially intersecting space-time rods causes NaNs in the log barrier interaction costs which makes the interior point method intractable. In order to generate initial plausible paths to feed into the optimization, we sequentially run a modified Djikstra's algorithm for all agents traversing the environment mesh inspired by [TCP06].

Djikstra finds the distance-weighted least costly path along the edges of the map mesh from the start location to the end location for a given agent. We avoid obstacles (environment boundary) by giving these edges, b_e, a prohibitively high traversal cost. We only consider map edges that have a distance from the map boundary at least greater than the agent's collision radius.

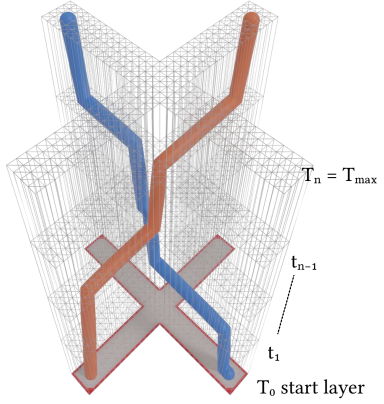


Figure 10: Running Djikstra's algorithm to set initial agent paths on the map. Red map edges indicate a higher traversal cost. Since we want to set a feasible initial path for each agent, we keep them away from the map edges as well as away from other agent's initial paths.

Furthermore, we ensure that every agent has a collision-free initial path by making it impossible for an agent's initial path to pass too close to the initial path of another agent. For each agent, upon finding an initial path, we wipe out all edges connected to the path within the agent's collision radius. For very constrained scenes, it becomes impossible to ensure collision free initial trajectories since agents often run out of traversible edges from $[x^0, y^0]$ to $[x^n, y^n]$.

Since agents move through both space and time, we extrude our environment map into the t dimension and create a pre-set number of layers from t^0 to T^{max} as shown in Figure 10. Solving this 3D space-time Djikstra's algorithm makes it a lot more feasible to find collision-free inital paths for all agents. Sometimes, the coarseness of the environment mesh, or the low number of layers in the 3D space-time map makes it impossible for Djikstra's algorithm to find a feasible initial path. In this case, either the environment map would need to be subdivided, more layers would need to be added, or the scene itself might be infeasible given the T^{max} time constraints and the number and size of agents involved. Additionally, although the space-time Djikstra's algorithm will provide feasible initial paths, they will have many kinks and sharp turns (Figure 11) indicating unnatural motion and thus cannot be used in the simulation directly. These get smoothed out during the optimization process resulting in much more realistic motion. It is possible that this proposed method to generate initial paths will affect the state of the final trajectories. For example, if an initial path passes through the left of an obstacle, it might be stuck in a local minimum even though an optimal trajectory might be through the right side. This drawback is somewhat mitigated by using Djikstra's search to pre-compute the least costly paths. Either way, whether or not the solver produces a globally minimum outcome, the trajectories are guaranteed to be smooth, viable and collision free.

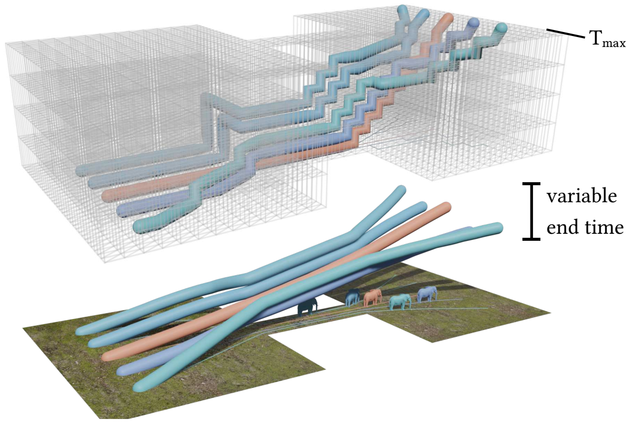


Figure 11: Top is the initial space-time curve output from Djikstra's path finding algorithm. Bottom is the final result of the solver with optimal paths. Agents are allowed to find their own optimal end-times while avoiding collisions in tightly constrained bottlenecks such as this.

3.6.1. Optimization

Now aggregating over all the agents in the scene, $||A||$, we put intra-agent cost functions Equations 4-7 together with interaction costs Equation 8-10 and construct an optimization problem for our scene. We aggregate boundary conditions for each agent and denote them with capital letters to show that they apply to the entire scene. The optimization problem

$$f^* = \min \sum_{a=1}^{||A||} C_a^K(\cdot) + C_a^A(\cdot) + C_a^T(\cdot) + C_a^R(\cdot) + C_a^M(\cdot) + \quad (12a)$$

$$\sum_{\substack{a,b\varepsilon \\ \{1..\|A\|\} \\ |a\neq b}} C^G_{a,b}(\cdot) + C^C_{a,b}(\cdot) \tag{12b}$$

$$s.t. \quad T^i \leq T^{i+1} \tag{12c}$$

$$(X^0, Y^0, T^0) = (X,Y,T)^{start} \tag{12d}$$

$$(X^n, Y^n) = (X,Y)^{end} \tag{12e}$$

$$T^n \leq T^{max} \tag{12f}$$

is solved using a quasi-newton interior point method. An outer loop over this optimization is required for a proper log-barrier method as explained by [NW06]. The full pseudocode overview of our method's outer loop, 2, illustrates this. The supplementary material for our method include the subroutines, the calculations for the gradients and hessian approximations of our costs, intuition for controlling agent behavior through function weights.

Algorithm 2 A full pseudocode overview of our method with subroutines provided in the supplementary material.

```
global input variables
    $OuterIts \geq 1$, outer loop iterations
    $\mu, c = 0.75$, log-barrier coeff and its decrement factor
    $\alpha_0, cutoff = 0.2$, initial alpha, Hess sparsifying cutoff
    $T^{max}$, max time constraint for agents
    $b_e, b_v$, environment edges and vertices.
    $K$, cost function weights for agents
    $R$, collision radius for agents
    $T$, preferred end times
    $M$, agent masses
end global input variables

interior point solver parameters
    $MaxIts$, max iterations
    $B$, trajectory boundary conditions
    Stop if $StepSize > 10^{-2}(meters)$, norm of solver step
    Stop if $FirstOrderOpt > 10^{-2}(meters)$, first order optimality criteria
end interior point solver parameters

Require: @COSTS, @GRADS, @HESS (supplementary material)

function FINDOPTIMALPATHS
    $[\mathbf{S}, B] \leftarrow DJIKSTRASPREPROCESS(R, T, b_v)$
    $\mathbf{S} \leftarrow$ increment $t$ by $\varepsilon$ to ensure $t_{i+1} > t_i$
    while OuterIts>0 do
        $\mathbf{S} \leftarrow IPSOLVER(\mathbf{S}, B, @COSTS, @GRADS, @HESS)$
        $\mu \leftarrow c\mu$
        $OuterIts \leftarrow OuterIts - 1$
    end while
    return S
end function
```

4. Results

THe performance of our method relies on (1) the number of agents in the scenes and (2) the complexity of the agent's paths through space and time. We push along each of these axes separately in this section. We show extremely complex maze-like environments with bottlenecks, as well as large scenes with tens (to hundreds) of agents. In addition to the comparisons shown in the related works, we include the obligatory circles of agents Figure 13. Notice that our agents take smooth, natural trajectories rather than the strange spinning motions demonstrated by other methods. Next, in Figure 17 we highlight the several different types of asymmetric interactions supported by our method. We show the naive case where agents collide. We show standard symmetric collision between agents. We show a stubborn snake which forces the scared humans to move out of its path. We show a large elk who is still scared of the humans so it changes its trajectory just as the human agents change theirs. We show a large and stubborn elephant that goes straight to its destination while the humans have to significantly alter trajectories to avoid it. Finally, we show a scenario where two friends stick closer to each other on their way to their final destination.

In Figure 14 and Figure 15, we show our method works in extremely tight environments. RVO and Implicit Crowds show that they can navigate environmental constraints with a lot of manual pre-processing, but nowhere near as tight as our examples. Meanwhile, NHTTC does not implement environmental constraints altogether. We show that our method works in extremely constrained scenarios such as a subway tunnel or a tight warehouse of robots where each lane is big enough to accommodate only a single agent and agents have to take turns to pass through to avoid locking.

Although our method is built to handle asymmetric interactions and agents in highly constrained spaces, we also show several other features of our method. First, our agents have flexible arrival times. In Figure 11 we show a bottleneck where all the agents simply cannot arrive at their destinations by their preferred end times. This feature serves as a counterpoint to the idea of using fixed start and end boundary conditions to model trajectory curves. In Figure 15 we show a highly constrained bottleneck of agents trying to get to their seats on an airplane.

Additionally, we show the navigability of agents in larger constrained environments in Figure 18a and Figure 18b. Agents are able to navigate the mazes while avoiding collisions with each other at the bottlenecks.

In Figure 19 we show the flexibility and controllability of our method. We create a scenario where one agent must meet up with another agent to deliver a message and then arrive at his end goal faster than the other agents. In another scenario, we direct the agent to meet up with the second agent, yet arrive at his end goal at the same time as the other agents. Lastly, we also provide the simple scene where agents are ignorant of each other. The flash mob example in Figure 20 shows how our method can be used to position agents intricately. The corresponding submission video shows how we are able to control the order of the placement of the agents in the flash mob as well through the preferred end time parameter. Our method allows careful control with no manual effort on the part of the user, thus making it useful for games or other industrial applications.

And finally, for the sake of completeness we show Figure 16, a

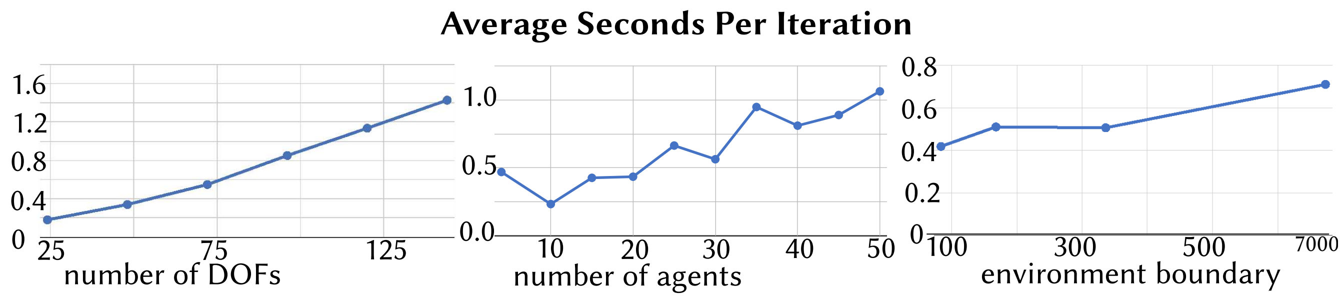


Figure 12: (left) we varying number of agents with 300 DOFs each. (middle) we vary the number of DOFs/agent from 90 to 540 on a fixed scene with 8 agents. (right) we vary DOFs in the environment boundary from 2520 to 20160 on the corn maze mesh with 3 agents of 600 DOFs each in the scene. Scaling tests show near linear performance in the worst-case and linear performance in the best case.

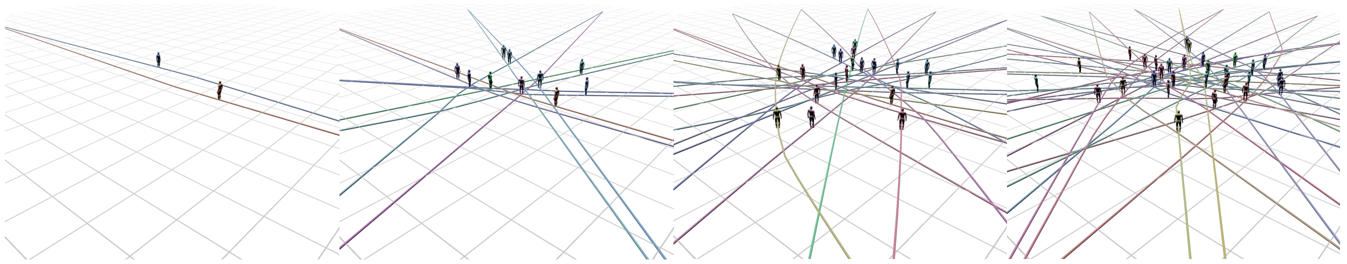
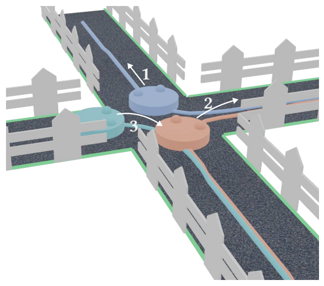
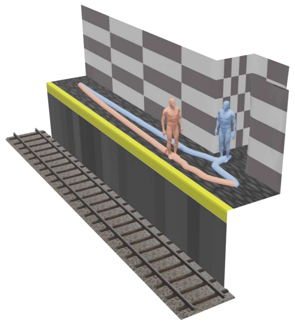


Figure 13: Circles of agents.



(a) Maze of roombas.



(b) Subway platform

Figure 14: In both (a) and (b) only one agent can move through the intersection at a time.

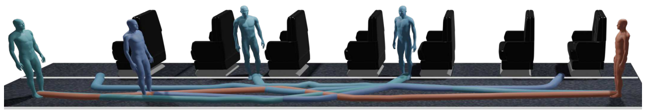


Figure 15: Chaos in first class. All the passengers are in incorrect seats.

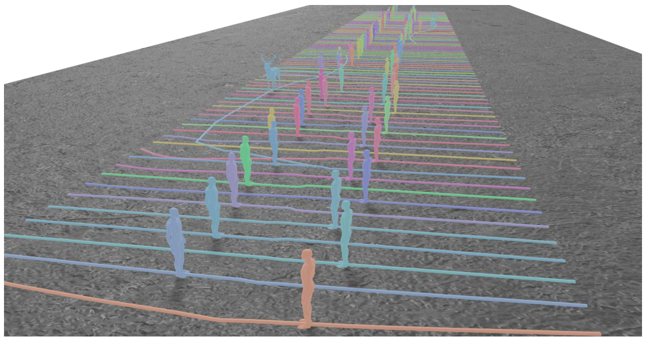


Figure 16: A large scene with 102 agents moving to opposite sides of a meadow while a couple of deer avoid them.

large scene with 104 agents. We include this example to demonstrate that even though our method is designed for small to medium sized scenes, it can work on simple scenes with larger numbers of agents with asymmetric interactions and still scales linearly in time.

4.1. Timings

Figure 12 provides three plots, each with different scaling information for different usage scenarios. The first plot measure the average time per solver function iteration for an increasing number of agents. Each agent trajectory has 100 nodes (300 DOFs) and the performance is linear with broad-phase collision detection enabled.

Our second plot measure the average time per iteration for an increasing number of DOFS for an 8 agent circle. The number of nodes starts off at 30 and goes all the way to 180 nodes per agent.

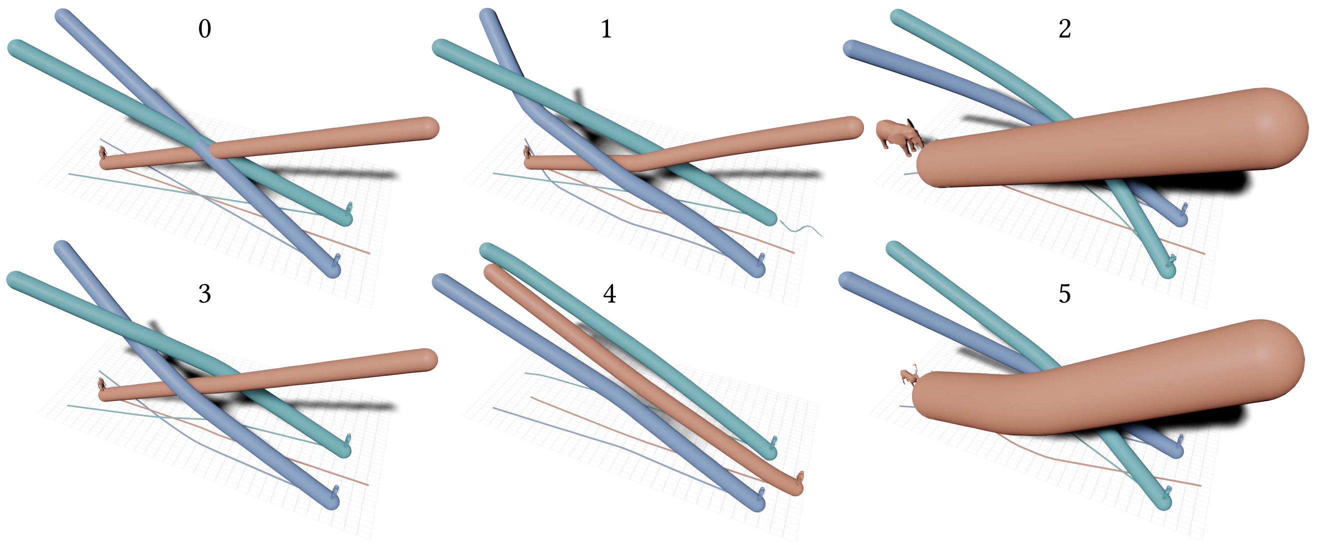
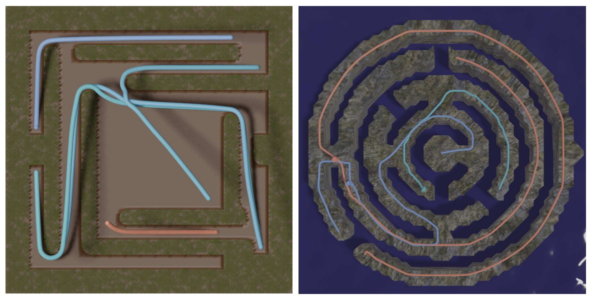


Figure 17: (0) No interactions. (1) Mass (analogous to stubbornness) weighted asymmetric interactions. (2) Size and mass weighted interactions. (3) Symmetric interactions.(4) Grouping friends together. (5) Size based interactions.



(a) Corn maze (b) Circle maze

Figure 18: (a) large cornmaze with eight agents all trying to get to different locations through the maze. (b) a circular shaped maze where agents need to navigate while avoiding collisions.

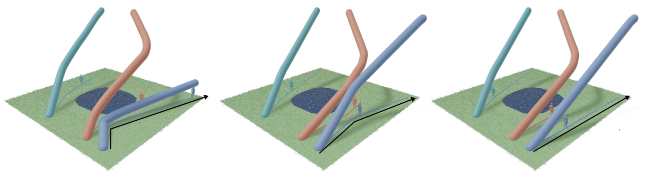
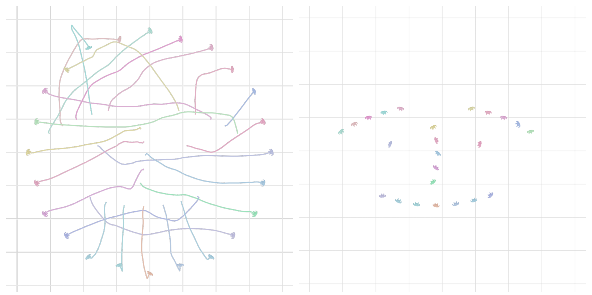


Figure 19: Left: An agent delivers a message to another agent and rushes to his goal position. Middle: Two agents meet up briefly before walking to their separate destinations. Right: No contact between any agents.

Performance is very nearly linear with broad phase collision detection enabled. Without a broad phase, performance is quadratic. Something to note is that increasing the number of DOFs per agent provides no additional benefit to the quality of the simulation since



(a) Pre-smile flash mob (b) Post-smile

Figure 20: (a) group of people in a flash mob instructed to form a smily face. Agents follow unintuitive paths through space and time to maintain the fluidity of their motion. (b) a smily face created by controlling the motion of the mob.

the actual physics of the space-time rods are not the end result of our method. As long as agents trajectory curves have enough DOFs to traverse the environment smoothly, the end results will be good. Our third plot measures the time per iteration for an increasing complexity in the environment mesh on the corn maze (Figure 18a) example with three agents and 200 trajectory nodes per agent. Again, performances is nearly linear.

As mentioned before there is no gold-standard crowd simulation algorithm since each scenario is so unique and intricate. Rather than focus on timing performance for large crowds, our method focuses on solving previously overlooked path planning scenarios with asymmetric interactions for small to medium sized groups in highly constrained environments. Our asymptotics show linear to near-linear performance, but there is plenty of room for improvement in wall-clock-times through optimization and parallelization.

5. Conclusion and Future Work

In this paper we show that modeling the motion of agents through space and time using a 3D curve resolves a number of difficulties with multi-agent path planning. Assigning physical characteristics such as a radius and mass to the agent's trajectory curve lets us intuitively simulate asymmetric interactions between agents. Our method outputs agent paths that are smooth, can navigate through highly constrained environments, and are parameterized by intuitive controls.

In the future we hope to improve our agent model to include limited environmental perception (rather than the omniscience our agents enjoy currently) as well as additional dynamics. As mentioned before, performance depends on two factors: number of agents and their temporal support. While we can currently push along these axes independently, a future work is to be able to push along both axes together, i.e. handle complex environments with large numbers of agents with intricate motion. We also hope to extend the method in order to handle scenes with conflicting and changing goals, e.g. a predator-prey scenario. We would also like to extend our space-time approach to fully 3D environments which requires performing our optimization in four dimensions. Finally, while our focus is on robustness in path planning, there is still further room for improvement in performance. We are excited to explore fast space-time multi-agent path planning to scale our approach to the large dense crowds that continuum approaches excel at, while maintaining our unique advantages.