Multi-Domain Real-time Planning in Dynamic Environments

Mubbasir Kapadia*[1], Alejandro Porres†[2], Francisco Garcia‡[3], Vivek Reddy§[1], Nuria Pelechano¶[2], and Norman I. Badler‖[1]

[1]University of Pennsylvania
[2]Universitat Politcnica de Catalunya
[3]University of Massachusetts Amherst

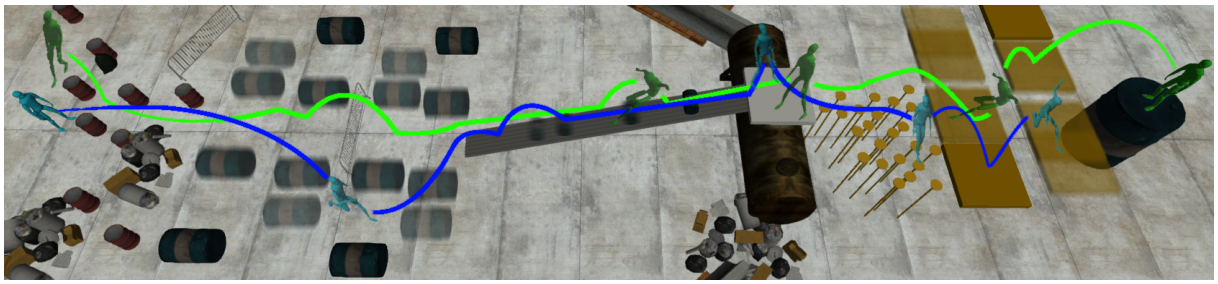


Figure 1: *Two agents navigating with space-time precision through a complex dynamic environment.*

Abstract

This paper presents a real-time planning framework for multi-character navigation that enables the use of multiple heterogeneous problem domains of differing complexities for navigation in large, complex, dynamic virtual environments. The original navigation problem is decomposed into a set of smaller problems that are distributed across planning tasks working in these different domains. An anytime dynamic planner is used to efficiently compute and repair plans for each of these tasks, while using plans in one domain to focus and accelerate searches in more complex domains. We demonstrate the benefits of our framework by solving many challenging multi-agent scenarios in complex dynamic environments requiring space-time precision and explicit coordination between interacting agents, by accounting for dynamic information at all stages of the decision-making process.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: real-time navigation, space-time planning

*mubbasir@seas.upenn.edu
†abeacco@gmail.com
‡fmaxgarcia@gmail.com
§vivreddy@seas.upenn.edu
¶npelechano@lsi.upc.edu
‖badler@seas.upenn.edu

1 Introduction

The next generation of interactive applications requires high fidelity navigation of interacting autonomous agents in non-deterministic, dynamic virtual worlds. The environment and agents are constantly affected by unpredictable forces (e.g., human input), making it impossible to accurately extrapolate the future world state to make optimal decisions. These complex domains require robust navigation algorithms that can handle partial and imperfect knowledge, while still making decisions which satisfy space-time constraints.

Different situations require different granularity of control. An open environment with no agents and static obstacles requires only coarse-grained control while cluttered dynamic environments require fine-grained character control with careful planned decisions that have spatial and temporal precision. Some situations (e.g., potential deadlocks) may require explicit coordination between multiple agents.

The problem domain of interacting autonomous agents in dynamic environments is extremely high-dimensional and continuous, with infinite ways to interact with objects and other agents. Having a rich action set, and a system that makes intelligent action choices, facilitates robust, intelligent virtual characters, at the expense of interactivity and scalability. Greatly simplifying the problem domain yields interactive virtual worlds with hundreds and thousands of agents that exhibit simple behavior. The ultimate, far-reaching goal is still a considerable challenge: a real-time system for autonomous character control that can handle many characters, without compromising control fidelity.

Previous work simulates crowds by decoupling global navigation [Sung et al. 2005; Kallmann 2010] and local collision avoidance [Pelechano et al. 2008], or demonstrates space-time planning for global navigation for a single character [Levine et al. 2011], while meeting real-time constraints. These approaches provide a tradeoff between number of agents, control fidelity, and environment complexity. To our knowledge, no proposed technique efficiently accounts for the dynamic nature of the environment at all levels of the decision-making process.

This paper proposes a real-time planning framework for multi-character navigation that uses multiple heterogeneous problem domains of differing complexities for navigation in large, complex,

dynamic virtual environments. We define a set of problem domains (spaces of decision-making) which differ in the complexity of their state representations and the fidelity of agent control. These range from a static navigation mesh domain which only accounts for static objects in the environment, to a space-time domain that factors in dynamic obstacles and other agents at much finer resolution. These domains provide different trade-offs in performance and fidelity of control, requiring a framework that efficiently works in multiple domains by using plans in one domain to focus and accelerate searches in more complex domains.

A global planning problem (start and goal configuration) is dynamically decomposed into a set of smaller problem instances across different domains, where an anytime dynamic planner is used to efficiently compute and repair plans for each of these problems. Planning tasks are connected by either using the computed path from one domain to define a *tunnel* to focus searches, or using successive waypoints along the path as start and goal for a planning task in another domain to reduce the search depth, thereby accelerating searches in more complex domains. Using our framework, we demonstrate real-time character navigation for multiple agents in large-scale, complex, dynamic environments, with precise control, and little computational overhead.

2 Related Work

There is extensive research in multi-agent simulations with many proposed techniques that differ in domain complexity and control fidelity. Global navigation approaches [Sung et al. 2005; Sud et al. 2007; van den Berg et al. 2008b; Kallmann 2010] precompute a roadmap of the global environment which is used for making efficient navigation queries, but generally regard the environment to be static. Crowd approaches [Pelechano et al. 2008; Thalmann 2008] compromise on control fidelity in an effort to efficiently simulate a large number of agents in real-time. Reactive approaches [Reynolds 1987; Lamarche and Donikian 2004; Loscos et al. 2003] avoid collisions with most imminent threats while predictive approaches [van den Berg et al. 2008a; Paris et al. 2007; Kapadia et al. 2009] approximate the trajectories of neighboring agents in choosing collision-free velocities. The work in [Singh et al. 2011a] proposes a hybrid technique that combines reactive rules, predictions, and planning for simulating crowds.

Planning based control of autonomous agents has demonstrated control of single agents with large action spaces [Choi et al. 2003; Fraichard 1999; Shapiro et al. 2007]. In an effort to scale to a large number of agents, meet real-time constraints, and handle dynamic environments, a large variety of methods [Pettré et al. 2008] have been proposed. The complexity of the domain is made simpler [Lau and Kuffner 2005; Lo and Zwicker 2008] to reduce the branching factor of the search, or the horizon of the search is limited to a fixed depth [Singh et al. 2011b; Choi et al. 2011]. Anytime planners [Likhachev et al. 2003; van den Berg et al. 2006] tradeoff optimality to satisfy strict time constraints, and have been successfully demonstrated for motion planning for a single character [Safonova and Hodgins 2007]. Randomized planners [Hsu et al. 2002; Shapiro et al. 2007] expand nodes in the search graph using sampling methods, greatly reducing search efforts to make it a feasible solution in high-dimensional, continuous domains. The work in [Phillips and Likhachev 2011] introduces safe time intervals as a novel abstraction of the temporal domain for planning in dynamic environments.

Hierarchical Planning. Hierarchical planners [Botea et al. 2004; Bulitko et al. 2007; Holte et al. 1996] reduce the problem complexity by precomputing abstractions in the state space, which can be used to speed up plan efforts. Given a discrete environment representation, neighboring states are first clustered together to precompute abstractions for high-level graphs. Different algorithms are proposed [Kring et al. 2010] which plan paths hierarchically by planning at the top level first, then recursively planning more detailed paths in the lower levels, using different methods [Lacaze 2002; Sturtevant and Geisberger 2010] to communicate information across hierarchies. These include using the plans in high-level graphs to compute heuristics for accelerating searches in low-level graphs [Holte et al. 2005], using the waypoints as intermediate goals, or using the high-level path to define a tunnel [Gochev et al. 2011] to focus the search in the low-level graph. The work in [Arikan and Forsyth 2002] demonstrates the use of randomized search in a hierarchy of motion graphs for interactive motion synthesis.

Comparison to Prior Work. Our work builds on top of excellent recent contributions [Levine et al. 2011; Lopez et al. 2012] showcasing the use of space-time planning for global navigation in dynamic environments, for a single agent. Levine *et al.* [2011] uses parameterized locomotion controllers to efficiently reduce the branching factor of the search and assumes that object motion have known trajectories, thus mitigating the need for replanning. Lopez *et al.* [2012] introduces a dynamic environment representation which is computed by deducing the evolution of the environment topology over time, thus enabling space-time collision avoidance with no prior knowledge of how the world changes. In contrast, we use multiple heterogeneous domains of control, and present a planning-based control scheme that reuses plan efforts across domains to demonstrate real-time, multi-character navigation, in constantly changing dynamic environments. Instead of automatically computing abstractions from a given representation, we develop a set of heterogeneous domains with different state and action representations that provide trade-offs in control fidelity and computational performance, and investigate different methods of communicating between domains to meet our application needs.

3 Overview

The problem domain of a planner determines its effectiveness in solving a particular problem instance. A complex domain that accounts for all environment factors such as dynamic environments and other agents, and has a large branching factor in its action space can solve more difficult problems, but at a larger cost overhead. A simpler domain definition provides the benefit of computational efficiency while compromising on control fidelity. Our framework enables the use of multiple heterogeneous domains of control, providing a balance between control fidelity and computational efficiency, without compromising either.

A global problem instance P_0 is dynamically decomposed into a set of smaller problem instances $\{P^{'}\}$ across different planning domains $\{\Sigma_i\}$. Section 4 describes the different domains, and Section 5 describes the problem decomposition across domains. Each problem instance $P^{'}$ is assigned a planning task $T(P^{'})$, and an anytime dynamic planner (Section 5.1) is used to efficiently compute and repair plans for each of these tasks, while using plans in one domain to focus and accelerate searches in more complex domains. Plan efforts across domains are reused in two ways. The computed path from one domain can be used to define a *tunnel* which focuses the search, reducing its effective branching factor. Each pair of successive waypoints along a path can also be used as start,goal pairs for a planning task in another domain, thus reducing the search depth. Both these methods are used to focus and accelerate searches in more complex domains, providing real-time efficiency without compromising on control fidelity. Section 6 describes the relationships between domains.

4 Planning Domains

A problem domain is defined as $\Sigma = \langle \mathbb{S}, \mathbb{A}, \mathtt{c}(s, s'), \mathtt{h}(s, s_{goal}) \rangle$, where the state space $\mathbb{S} = \{\mathbb{S}_{self} \times \mathbb{S}_{env} \times \mathbb{S}_{agents}\}$ includes the internal state of the agent $\mathbb{S}_{self}$, the representation of the environment $\mathbb{S}_{env}$, and other agents $\mathbb{S}_{agents}$. $\mathbb{S}_{self}$ may be modeled as a simple particle with a collision radius. $\mathbb{S}_{env}$ can be an environment triangulation with only static information or a uniform grid representation with dynamic obstacles. $\mathbb{S}_{agents}$ is defined by the vicinity within which neighboring agents are considered. Imminent threats may be considered individually or just represented as a density distribution at far-away distances. The action space $\mathbb{A}$ defines the set of all possible successors $\mathtt{succ}(s)$ and predecessors $\mathtt{pred}(s)$ at each state, as shown in Equation 1. Here, $\delta(s, i)$ describes the i^{th} transition, and $\Phi(s, s')$ is used to check if the transition from s to s' is possible. The cost function $\mathtt{c}(s, s')$ defines the cost of transition from s to s'. The heuristic function $\mathtt{h}(s, s_{goal})$ defines the estimate cost of reaching a goal state.

$$\mathtt{succ}(s) = \{s + \delta(s, i) | \Phi(s, s') = \mathtt{TRUE} \ \ \forall i = 1 \text{ to } N\} \quad (1)$$

A problem definition $P = \langle \Sigma, s_{start}, s_{goal} \rangle$ describes the initial configuration of the agent, the environment and other agents, along with the desired goal configuration in a particular domain. Given a problem definition P for domain Σ, a planner searches for a sequence of transitions to generate a plan $\Pi(\Sigma, s_{start}, s_{goal}) = \{s_i | s_i \in \mathbb{S}(\Sigma)\}$ that takes an agent from s_{start} to s_{goal}.

4.1 Multiple Domains of Control

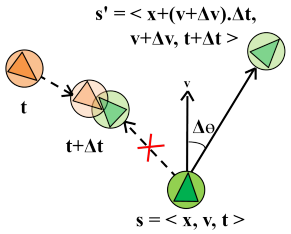
We define 4 domains which provide a nice balance between global static navigation and fine-grained space-time control of agents in dynamic environments. Figure 2 illustrates the different domain representations for a given environment.

Static Navigation Mesh Domain Σ_1. This domain uses a triangulated representation of free space and only considers static immovable geometry. Dynamic obstacles and agents are not considered in this domain. The agent is modeled as a point mass, and valid transitions are between connected free spaces, represented as polygons. The cost function is the straight line distance between the center points of two free spaces. Additional connections are also precomputed (or manually annotated) to represent transitions such as jumping with a higher cost definition. The heuristic function is the Euclidean distance between a state and the goal. Searching for an optimal solution in this domain is very efficient and quickly provides a global path for the agent to navigate. We use Recast [Mononen 2009] to precompute the navigation mesh for the static geometry in the environment.

Dynamic Navigation Mesh Domain Σ_2. This also uses triangulations to represent free spaces and coarsely accounts for dynamic properties of the environment to make a more informed decision at the global planning layer. The work in [van Toll et al. 2012] embeds population density information in environment triangulations to account for the movement of agents at the global planning layer. We adopt a similar method by defining a time-varying density field $\phi(t)$ which stores the density of moveable objects (agents and obstacles) for each polygon in the triangulation at some point of time t. $\phi(t_0)$ represents the density of agents and obstacles currently present in the polygon. The presence of objects and agents in polygons at future timesteps can be estimated by querying their plans (if available). The space-time positions of deterministic objects can be accurately queried while the future positions of agents can be approximated based on their current computed paths, assuming that they travel with constant speed along the path without deviation. $\phi(t)$ contributes to the cost of selecting a waypoint in Σ_2 during planning. The resolution of the triangulation may be kept finer than Σ_1 to increase the resolution of the dynamic information in this domain. Hence, a set of global waypoints are chosen in this domain which avoids crowded areas or other high cost regions.

Grid Domain Σ_3. The grid domain discretizes the environment into grid cells where a valid transition is considered between adjacent cells that are free (diagonal movement is allowed). An agent is modeled as a point with a radius (orientation and agent speed is not considered in this domain). This domain only accounts for the current position of dynamic obstacles and agents, and cannot predict collisions in space-time. The cost and heuristic are distance functions that measure the Eucledian distance between grid cells.

Space-Time Domain Σ_4. This domain models the current state of an agent as a space-time position with a current velocity $(\mathbf{x}, \mathbf{v}, t)$. The figure alongside illustrates the schematic illustration of the state and action space in Σ_4, showing a valid transition, and an invalid transition due to a space-time collision with a neighboring agent. The transition function $\delta(s, i)$ for Σ_4 is defined below:

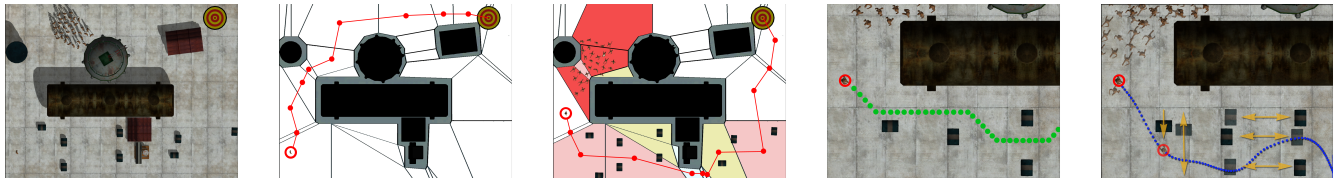


$$\delta(s, i) = \{\Delta \mathbf{v}_i \cdot \Delta t | \Delta \mathbf{v}_i = (\Delta v_i \cdot \sin \Delta\theta_i, \Delta v_i \cdot \cos \Delta\theta_i) \forall i\}$$

where $\Delta v = \{0, \pm a\}$ is the possible speed changes and $\Delta\theta = \{0, \pm\frac{\pi}{8}, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}\}$ is the possible orientation changes the agent can make from its current state. For example, $\Delta\mathbf{v} = a, \Delta\theta = \frac{\pi}{8}$ produces a transition where the agent accelerates by a for the duration of the timestep and rotates by $\frac{\pi}{8}$. The bounds of $\Delta\theta$ are limited between $\{-\frac{\pi}{2}, \frac{\pi}{2}\}$ to limit the maximum rate of turning. Transitions are also bound so that the speed and acceleration of an agent cannot exceed a given threshold. Jumps are additionally modeled as a high cost transition between two space-time points such that the region between them may be occupied or untraversable for that time interval. Inspite of the coarse discretization of $\Delta\theta$, the branching factor of this domain is much higher, providing greater degree of control fidelity with added computational overhead.

Σ_4 accounts for all obstacles (static and dynamic) and other agents. The traversability of a grid cell is queried in space-time by checking to see if moveable obstacles and agents occupy that cell at that particular point of time, by using their published paths. For space-time collision checks, only agents and obstacles that are within a certain region from the agent, defined using a foveal angle intersection, are considered. The cost and heuristic definitions have a great impact on the performance in Σ_4. We use an energy based cost formulation that penalizes change in velocity with a non-zero cost for zero velocity. Jump transitions incur a higher cost. The heuristic function penalizes states that are far away from s_{goal} in both space and time. This is achieved using a weighted combination of a distance metric and a penalty for a deviation of the current speed from the speed estimate required to reach s_{goal}.

The domains described here are *not* a comprehensive set and only serve to showcase the ability of our framework to use multiple heterogeneous domains of control in order to solve difficult problem instances at a fraction of the computation cost. Our framework can be easily extended to use other domain definitions (e.g., a footstep domain), as described in Section 7.4.



(a) (b) (c) (d) (e)

Figure 2: *(a) Problem definition with initial configuration of agent and environment. (b) Global plan in static navigation mesh domain Σ_1 accounting for only static geometry. (c) Global plan in dynamic navigation mesh domain Σ_2 accounting for cumulative effect of dynamic objects. (d) Grid plan in Σ_3. (e) Space-time plan in Σ_4 that avoids dynamic threats and other agents.*

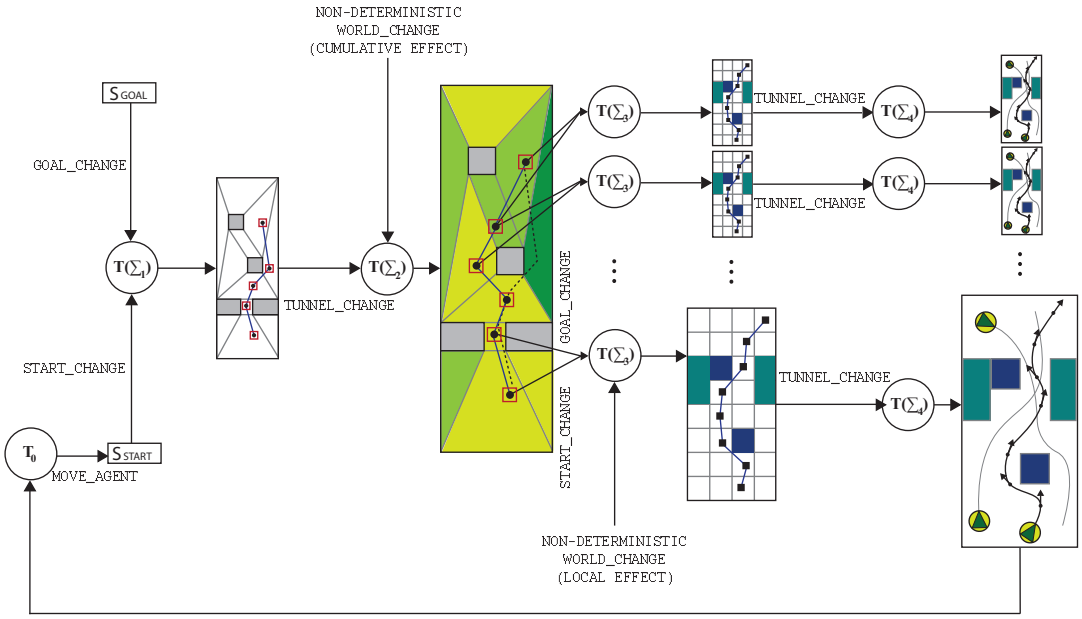


Figure 3: *Expanded illustration of domain relationship shown in Figure 4(b). A global problem instance (start and goal state) is decomposed into a set of smaller problem instances across multiple planning domains. Planning tasks $T(\Sigma)$ are assigned to each of these problems and scheduled using a dynamic priority scheme based on events from the environment and other tasks.*

5 Problem Decomposition and Multi-Domain Planning

Figure 4(a) illustrates the use of tunnels to connect each of the 4 domains, ensuring that a complete path from the agents initial position to its global target is computed at all levels. Figure 4(b) shows how Σ_2 and Σ_3 are connected by using successive waypoints in $\Pi(\Sigma_2)$ as start and goal for independent planning tasks in Σ_3. This relation between Σ_2 and Σ_3 allows finer-resolution plans being computed between waypoints in an independent fashion. Limiting Σ_3 (and Σ_4) to plan between waypoints instead of the global problem instance ensures that the search horizon in these domains is never too large, and that fine-grained space-time trajectories to the initial waypoints are computed quickly. However, completeness and optimality guarantees are relaxed as Σ_3, Σ_4 never compute a single path to the global target.

Figure 3 illustrates the different events that are sent between planning tasks to trigger plan refinement and updates for the domain relationship in Figure 4(b). Σ_1 is first used to compute a path from s_{start} to s_{goal}, ignoring dynamic obstacles and other agents. $\Pi(\Sigma_1)$ is used to accelerate computations in Σ_2, which refines the global path to factor in the distribution of dynamic objects in the environment. Depending on the relationship between Σ_2 and Σ_3,

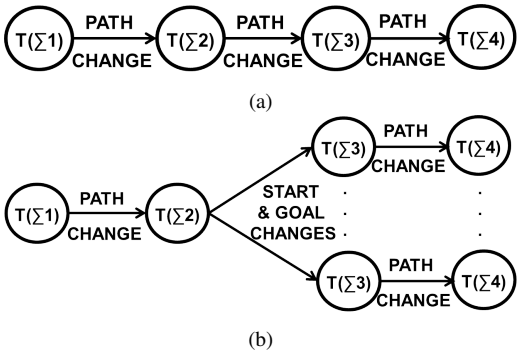


Figure 4: *Relationship between domains. (a) Use of tunnels to connect each of the 4 domains. (b) Use of successive waypoints in $\Pi(\Sigma_2)$ as start, goal pairs to instantiate multiple planning tasks in Σ_3 and Σ_4.*

a single planning task or multiple independent planning tasks are used in Σ_3. Finally, the plan(s) of $T(\Sigma_3)$ are used to accelerate searches in Σ_4.

Changes in s_{start} and s_{goal} trigger plan updates in $T(\Sigma_1)$, which

are propagated through the task dependency chain. $T(\Sigma_2)$ monitors plan changes in $T(\Sigma_1)$ as well as the cumulative effect of changes in the environment to refine its path. Each $T(\Sigma_3)$ instance monitors changes in the waypoints along $\Pi(\Sigma_2)$ to repair its solution, as well as nearby changes in obstacle and agent position. Finally, $T(\Sigma_4)$ monitors plan changes in $T(\Sigma_3)$ (which it depends on) and repairs its solution to compute a space-time trajectory that avoids collisions with static and dynamic obstacles, as well as other agents.

Events are triggered (outgoing edges) and monitored (incoming edges) by tasks, creating a cyclic dependency between tasks, with $\mathtt{T}_0$ (agent execution) monitoring changes in the plan produced by the particular $T(\Sigma_4)$, which monitors the agents most imminent global waypoint. Tasks that directly affect the agent's next decision, and tasks with currently invalid or sub-optimal solutions are given higher priority. Given the maximum amount of time to deliberate t_{max}, the agent pops one or more tasks that have highest priority and divides the deliberation time across tasks (most imminent tasks are allocated more time). Task priorities constantly change based on events triggered by the environment and other tasks.

5.1 Planning Tasks

A task $T(P)$ is a planner which is responsible for generating and maintaining a valid (and ideally optimal) solution for a particular problem definition $P = \langle \Sigma, s_{start}, s_{goal} \rangle$ where s_{start}, s_{goal}, and the search graph may be constantly changing. There are 4 types of tasks, each of which solves a particular problem in the domains described in Section 4. An additional task $\mathtt{T}_0$ is responsible for moving the agent along the path, while enforcing steering and collision constraints.

Planning tasks constantly receive events from the environment and other tasks, which render the current plan invalid, forcing it to constantly update, refine, and repair its existing plan. For this purpose, we use the Anytime Dynamic A* planner [Likhachev et al. 2005] which combines the properties of incremental planners such as D* Lite [Koenig and Likhachev 2002] and anytime algorithms such as ARA* [Likhachev et al. 2003] to provide an algorithm which efficiently repairs its solutions to accommodate world changes and agent movement, while providing solution guarantees under strict time constraints. It performs repeated backward searches (from goal to start), reusing previous search efforts to iteratively produce solutions with improved bounds on optimality, like ARA*. This is done using an inflation factor ϵ which effectively weighs the contribution of the heuristic value in estimation of node costs, thus focusing the search towards the goal, expanding fewer nodes to produce ϵ sub-optimal solutions [Pearl 1984]. We provide an overview of the algorithmic details of the planning task in Appendix B and refer the readers to a comprehensive review of the AD* algorithm here [Likhachev et al. 2005].

Plan Repair vs. Planning from Scratch. Note that there are often instances during the simulation when the start and goal changes of planning tasks change or when plans are invalidated due to obstacle movement. Plans are always recomputed for goal changes. AD* performs a backward search which allows it to efficiently update the search graph to accommodate agent movement along the path. For significant start changes or when the plan is invalidated due to obstacle movement, the choice between replanning or repairing a plan is a heuristic decision with tradeoffs in performance. Plan repair may expand lesser nodes in the current iteration but bloat the number of nodes visited, thus impacting performance in subsequent plan iterations. It is not uncommon to plan from scratch during the simulation. By resetting the inflation factor to a high value, we can quickly compute a valid sub-optimal plan while meeting time constraints and refine it in successive plan iterations.

5.2 Events and Task Priorities

Events are triggered and monitored by planning tasks in different domains, as illustrated in Figure 3. Changes in start and goal, or environment changes may potentially invalidate current plans, requiring plan refinement. Tasks that use tunnels to accelerate searches in more complex domains, monitor plan changes in other tasks. Finally, tasks observe the optimality status of their own plans to determine their task priority. Appendix A describes the different events in more detail.

The priority of a task $p(\mathtt{T}_a)$ determines the tasks that are picked to be executed at every time step, with tasks having smallest $p(\mathtt{T}_a)$ chosen for execution ($p(\mathtt{T}_a)$ is short for $p(T(\Sigma_a))$). Task $\mathtt{T}_0$, which handles agent movement always has a priority of 1. Priority of other tasks is calculated as follows:

$$p(\mathtt{T}_a) = \begin{cases} 1 \text{ if } \mathtt{T}_a = \mathtt{T}_0 \\ \mu(\mathtt{T}_a, \mathtt{T}_0) \cdot \Omega(\mathtt{T}_a) \text{ else} \end{cases} \quad (2)$$

where $\mu(\mathtt{T}_a, \mathtt{T}_0)$ is the number of edge traversals required to reach $\mathtt{T}_0$ from $\mathtt{T}_a$ in the task dependency chain (Figure 3). $\Omega(\mathtt{T}_a)$ denotes the current state of the plan of $\mathtt{T}_a$ and is defined as follows:

$$\Omega(\mathtt{T}_a) = \begin{cases} 1 \text{ if } \texttt{SOLUTION\_INVALID} \\ \epsilon \text{ if plan inflation factor, } \epsilon > 1 \\ \infty \text{ if plan inflation factor, } \epsilon = 1 \end{cases} \quad (3)$$

where ϵ is the inflation factor used to determine the optimality bounds of the current plan for that task. The agent pops one or more tasks that have highest priority and divides the deliberation time available across tasks, with execution-critical tasks receiving more time. Tasks that have the same priority are ordered based on task dependency. Hence, $\mathtt{T}_0$ is always executed at the end of every update after all planning tasks have completed.

The overall framework enforces strict time constraints. Given an allocated time to deliberate for each agent (computed based on desired frame rate and number of agents), the time resource is distributed based on task priority. In the remote event that there is no action to execute, the agent remains stationary (no impact on frame-rate) for a few frames (fractions of a second) until a valid plan is computed.

6 Relationship between Domains

The complexity of the planning problem increases exponentially with increase in dimensionality of the search space – making the use of high-dimensional domains nearly prohibitive for real-time applications. In order to make this problem tractable, planning tasks must efficiently use plans in one domain to focus and accelerate searches in more complex domains. Section 6.1 describes a method for mapping a state from a low-dimensional domain to one or more states in a higher dimensional domain. Sections 6.2 and 6.3 describe two ways in which plans in one domain can be used to focus and accelerate searches in another domain.

6.1 Domain Mapping

We define a $1 : n$ function $\lambda(s, \Sigma, \Sigma^{'})$ that allows us to maps states in $\mathbb{S}(\Sigma)$ to one or more equivalent states in $\mathbb{S}(\Sigma^{'})$.

$$\lambda(s, \Sigma, \Sigma^{'}) : s \rightarrow \{s' | s' \in \mathbb{S}(\Sigma^{'}) \wedge s \equiv s'\} \quad (4)$$

The mapping functions are defined specifically for each domain pair. For example, $\lambda(s, \Sigma_1, \Sigma_2)$ maps a polygon $s \in \mathbb{S}(\Sigma_1)$ to one or more polygons $\{s' | s' \in \mathbb{S}(\Sigma_2)\}$ such that s' is spatially contained in s. If the same triangulation is used for both Σ_1 and Σ_2, then there exists a one-to-one mapping between states. Similarly, $\lambda(s, \Sigma_2, \Sigma_3)$ maps a polygon $s \in \mathbb{S}(\Sigma_2)$ to multiple grid cells $\{s' | s' \in \mathbb{S}(\Sigma_3)\}$ such that s' is spatially contained in s. $\lambda(s, \Sigma_3, \Sigma_4)$ is defined as follows:

$$\lambda(s, \Sigma_3, \Sigma_4) : (\mathbf{x}) \rightarrow \{(\mathbf{x} + W(\Delta \mathbf{x}), t + W(\Delta t))\} \quad (5)$$

where $W(\Delta)$ is a window function in the range $[-\Delta, +\Delta]$. The choice of t is important in mapping Σ_3 to Σ_4. Since we use λ to effectively map a plan $\Pi(\Sigma_3, s_{start}, s_{goal})$ in Σ_3 to a tunnel in Σ_4, we can exploit the path and the temporal constraints of s_{start} and s_{goal} to define t for all states along the path. We do this by calculating the total path length and the time to reach s_{goal}. This allows us to compute the approximate time of reaching a state along the path, assuming the agent is traveling at a constant speed along the path.

6.2 Mapping Successive Waypoints to Independent Planning Tasks.

Successive waypoints along the plan from one domain can be used as start and goal for a planning task in another domain. This effectively decomposes a planning problem into multiple independent planning tasks, each with a significantly smaller search depth.

Consider a path $\Pi(\Sigma_2) = \{s_i | s_i \in \mathbb{S}(\Sigma_2), \forall i \in (0, n)\}$ of length n. For each successive waypoint pair (s_i, s_{i+1}), we define a planning problem $P_i = \langle \Sigma_3, s_{start}, s_{goal} \rangle$ such that $s_{start} = \lambda(s_i, \Sigma_2, \Sigma_3)$ and $s_{goal} = \lambda(s_{i+1}, \Sigma_2, \Sigma_3)$. Even though λ may return multiple equivalent states, we choose only one candidate state. For each problem definition P_i, we instantiate an independent planning task $T(P_i)$which computes and maintains path from s_i to s_{i+1} in Σ_3. Figure 4 illustrates this connection between Σ_2 and Σ_3.

6.3 Tunnels

The work in [Gochev et al. 2011] observes that a plan in a low dimensional problem domain can often be exploited to greatly accelerate high-dimensional complex planning problems by focusing searches in the neighborhood of the low dimensional plan. They introduce the concept of a tunnel $\tau(\Sigma_{hd}, \Pi(\Sigma_{ld}), t_w)$ as a sub graph in the high dimensional space Σ_{hd} such that the distance of all states in the tunnel from the low dimensional plan $\Pi(\Sigma_{ld})$ is less than the tunnel width t_w. Based on their work, we use plans from one domain in order to accelerate searches in more complex domains with much larger action spaces. A planner is input a low dimensional plan $\Pi(\Sigma_{ld})$ which is used to focus state transitions in the sub graph defined by the tunnel $\tau(\Sigma_{hd}, \Pi(\Sigma_{ld}), t_w)$.

To check if a state s lies within a tunnel $\tau(\Sigma_{hd}, \Pi(\Sigma_{ld}), t_w)$ without precomputing the tunnel itself, the low dimensional plan $\Pi(\Sigma_{ld})$ is first converted to a high dimensional plan $\Pi^{'}(\Sigma_{hd}, s_{start}, s_{goal})$ by mapping all states of Π to their corresponding states in $\Pi^{'}$, using the mapping function $\lambda(s, \Sigma_{ld}, \Sigma_{hd})$ as defined in Equation 4. Note that the resulting plan $\Pi^{'}$ may have multiple possible trajectories from s_{start} to s_{goal} due to the $1 : n$ mapping of λ. Next, we define a distance measure $\mathbf{d}(s, \Pi(\Sigma))$ which computes the distance of s from the path $\Pi(\Sigma)$. During a planning iteration, a state is generated if and only if $\mathbf{d}(s, \Pi(\Sigma_{hd})) \leq t_w$. This is achieved by redefining the $\texttt{succ}(s)$ and $\texttt{pred}(s)$ to only consider states that lie in the tunnel. Furthermore, node expansion can be prioritized to states that are closer to the path by modifying the heuristic function as shown in below.

$$h_t(s, s_{start}) = h(s, s_{start}) + |\mathbf{d}(s, \Pi(\Sigma))| \quad (6)$$

Note that the heuristic $h_t(s, s_{start})$ is an estimate of the distance from s to s_{start} since we use a backward search from s_{goal} to s_{start} to accomodate start movement. For spatial domains Σ_1, Σ_2, and Σ_3, $\mathbf{d}(s, \Pi(\Sigma))$ is the perpendicular distance between s and the line segment connecting the two nearest states in $\Pi(\Sigma)$. $\mathbf{d}(s, \Pi(\Sigma_4))$ will return a two-tuple value for spatial distance as well as temporal distance.

TunnelChangeUpdate. When the tunnel changes, previously visited nodes that are no longer within the new tunnel are assigned an infinite cost and the changes are propagated to their successors. Also, their heuristic values are updated to reflect the new tunnel distance using Equation 6, which re-prioritizes node expansion to nodes that are closer to the new path. The tunnel width $t_w$is inversely proportional to the inflation factor ϵ. Thus, a high ϵ focuses the search within a narrow tunnel, which is iteratively expanded when ϵ is reduced to increase the breadth of the search. Due to the extremely dynamic nature of the planning tasks, we find that a reasonably narrow tunnel allows solutions to be returned very quickly which can be improved, if time permits. If the tunnel is too narrow, however, no plan maybe returned, requiring a replan in a wider tunnel. Appendix B, Alg 2 [22–30] provides the algorithmic details to handle tunnel changes that are sent between planning tasks in different domains.

Completeness and Optimality Guarantees. The use of tunnels enables AD* to leverage plans across domains in order to expedite searches in high-dimensional domains. However; by modifying the definition of $\texttt{succ}(s)$ and $\texttt{pred}(s)$ to prune nodes that lie outside the tunnel, we sacrifice the strict bounds on optimality provided by AD*, as nodes that lie outside the tunnel may lead to a more optimal solution. By iteratively expanding the tunnel width t_w, when the search is unsuccessful, we ensure that a solution will be found, if one exists. For practical purposes, we find that a constantly dynamic world mitigates the need for strict optimality bounds as solutions are constantly invalidated, before their use. In our experiments (Section 7.1), we find that the computational benefit of using tunnels far outweighs its drawbacks, providing an exponential reduction in the nodes expanded, while still producing reasonable quality solutions.

7 Results

7.1 Comparative Evaluation of Domain Relationships

We randomly generate 1000 scenarios of size $100m \times 100m$, with random configurations of obstacles (both static and dynamic), start state, and goal state and record the effective branching factor, number of nodes expanded, time to compute a plan, success rate, and quality of the plans obtained. The effective branching factor is the average number of successors that were generated over the course of one search. Success rate is the ratio of the number of scenarios for which a collision-free solution was obtained. Plan quality is the ratio of the length of the static optimal path and the path obtained. A plan quality of 1 indicates that the solution obtained was able to minimize distance without any deviations. The aggregate metrics for the different domains and domain relationships are shown in Table 1. Rows 3 and 6 in Table 1 include the added time to compute plans in earlier domains for tunnel search, to provide an absolute basis of comparison.

Σ_1 and Σ_2 can quickly generate solutions but is unable to solve most of the scenarios as they don't resolve fine-grained collisions. The use of plans from Σ_1 accelerates searches in Σ_2 (Table 1, Row 3). However, the real benefit of using both Σ_1 and Σ_2 is evident when performing repeated searches across domains in large environments when an initial plan $\Pi(\Sigma_1)$ accelerates repeated refinements in Σ_2 (and other subsequent domains). Using Σ_3 in a large environment takes significantly longer to produce similar paths. Σ_4 is unable to find a complete solution for large-scale problem instances (we limit maximum number of nodes expanded to 10^4), and the partial solutions often suffer from local minima, resulting in a low success rate. The benefit of using tunnels is evident in the dramatic reduction of the effective branching factor and nodes expanded for Σ_4.

When using the complete global path from Σ_3 as a tunnel for Σ_4 (Figure 4(a) and Row 6 in Table 1), the effective branching factor reduces from 21.5 to 5.6, producing an exponential drop in node expansion and computation time, and enabling complete solutions to be generated in the space-time domain. This planning task is able to successfully solve nearly 92% of the scenarios that were generated. However, since s_{start} and s_{goal} are far apart, the large depth of the search prevents this from being used at interactive rates for many agents.

By using successive waypoints in $\Pi(\Sigma_2)$ as s_{start} and s_{goal} to create a series of planning tasks in Σ_3 and Σ_4 (Figure 4(b) and Row 7 in Table 1), we reduce the breadth *and* depth of the search, allowing solutions to be returned at a fraction of the time (6 ms), without significantly affecting the success rate. The tradeoff is that independent plans are generated between waypoints along the global path, creating a two-level hierarchy between the domains.

Domain	BF	N	T	S	Q
$T(\Sigma_1)$	3.7	43	3	0.17	0.76
$T(\Sigma_2)$	4.6	85	8	0.23	0.57
$T(\Sigma_2, \Pi(\Sigma_1))$	2.1	17	5	0.32	0.65
$T(\Sigma_3)$	7.4	187	18	0.68	0.73
$T(\Sigma_4)$	21.5	10^4	2487	0.34	0.26
$T(\Sigma_4, \Pi(\Sigma_3, \Sigma_2, \Sigma_1))$	5.6	765	136	0.92	0.64
$\sum T_i(\Sigma_4, \Pi(\Sigma_3, \Sigma_2, \Sigma_1))$	5.4	75	8	0.86	0.58

Table 1: *Comparative evaluation of the domains, and the use of multiple domains.* ***BF*** *= Effective branching factor.* ***N*** *= Average number of nodes expanded.* ***T*** *= Average time to compute plan (ms).* ***S*** *= Success rate of planner to produce collision-free trajectory.* ***Q*** *= Plan quality. Row 6,7 corresponds to the domain relationships illustrated in Figures 4(a) and (b) respectively.*

Conclusion. The comparative evaluations of domains shows that no single domain can efficiently solve the challenging problem instances that were sampled. The use of tunnels significantly reduce the effective branching factor of the search in Σ_3 and Σ_4, while mapping successive waypoints in $\Pi(\Sigma_2)$ to multiple independent planning tasks reduce the depth of the search in Σ_3 and Σ_4, without significantly impacting success rate and quality. For the remaining results in the paper, we adopt this domain relationship as it works well for our application of simulating multiple goal-directed agents in dynamic environments at interactive rates. Users may choose a different relationship based on their specific needs.

7.2 Performance

We measure the performance of the framework by monitoring the execution time of each task type, with multiple instances of planning tasks for Σ_3 and Σ_4. We limit the maximum deliberation time $t_{max} = 10$ ms, which means that the total time executing any of the tasks at each frame cannot exceed $10ms$. For this experiment, we limit the total number of tasks that can be executed in a single frame to 2 (including $\mathtt{T}_0$) to visualize the execution time of each task over different frames. Figure 6 illustrates the task execution times of a single agent over a 30 second simulation for the scenario shown in Figure 2(a). The execution task $\mathtt{T}_0$ which is responsible for character animation and simple steering takes approximately $0.4 - 0.5$ ms of execution time every frame. Spikes in the execution time correlate to events in the world. For example, a local non-deterministic change in the environment (Frames 31,157) triggers a plan update in $T(\Sigma_3)$, which in turn triggers an update in $T(\Sigma_4)$. A global change such as a crowd blocking a passage or a change in goal (Frames 39, 237,281) triggers an update in $T(\Sigma_2)$ or $T(\Sigma_1)$ which in turn propagates events down the task dependency chain.

Note that there are often instances during the simulation when the start and goal changes significantly or when plans are invalidated, requiring planning from scratch. However, we ensure that our framework meets real-time constraints due to the following design decisions: (a) limiting the maximum amount of time to deliberate for the planning tasks, (b) intelligently distributing the available computational resources between tasks with highest priority, and (c) increasing the inflation factor to quickly produce a sub-optimal solution when a plan is invalidated, and refining the plan in successive frames.

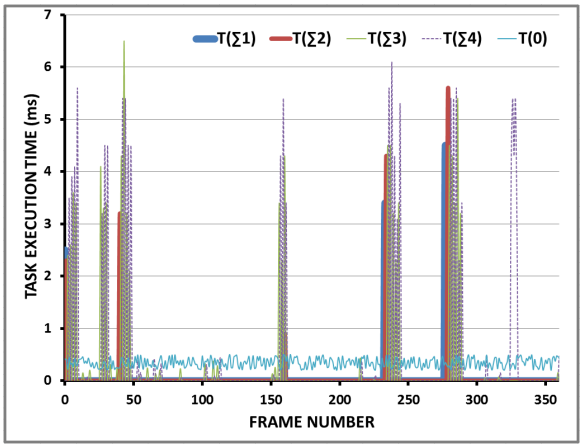
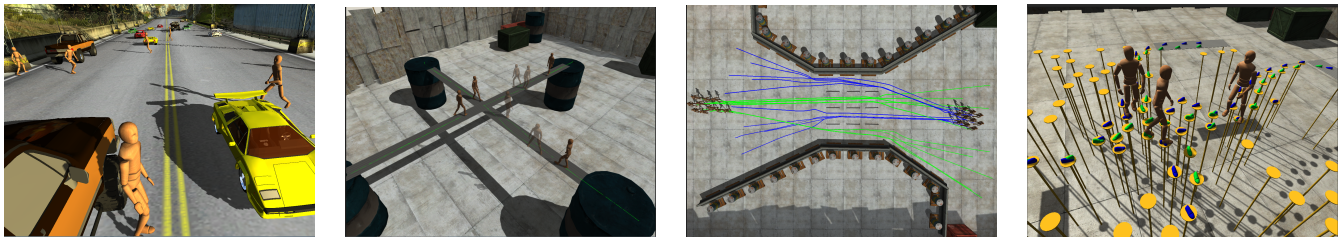


Figure 6: *Task execution times of the different tasks in our framework over the course of a* 60 *second simulation.*

Memory. $T(\Sigma_1)$ and $T(\Sigma_2)$ precomputes navigation meshes for the environment whose size depend on environment complexity, but are shared by all agents in the simulation. The runtime memory requirement of these tasks is negligible since it expands very few nodes. The memory footprint of $T(\Sigma_3)$ and $T(\Sigma_4)$ is defined by the number of nodes visited by the planning task during the course of a simulation. Since each planning task in Σ_3 and Σ_4 searches between successive waypoints in the global plan, the search horizon of the planners is never too large. On average, the number of visited nodes is 75 and 350 for $T(\Sigma_3)$ and $T(\Sigma_4)$ respectively with each node occupying $16 - 24$ bytes in memory. For 5 running instances of $T(\Sigma_3)$ and $T(\Sigma_4)$, this amounts to approximately $45KB$ of memory per agent. Additional memory for storing other plan containers such as `OPEN` and `CLOSED` are not considered in this calculation as they store only node references and are cleared after every plan iteration.

Scalability. Our approach scales linearly with increase in number of agents. The maximum deliberation time *for all* agents can be chosen based on the desired frame rate which is then distributed among agents and their respective planning tasks at each frame. The cost of planning is amortized over several frames and all agents



(a) (b) (c) (d)

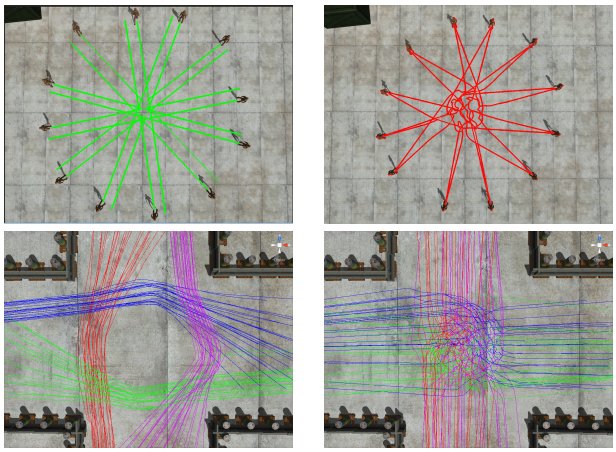
Figure 5: *Different scenarios. (a) Agents crossing a highway with fast moving vehicles in both directions. (b)* 4 *agents solving a deadlock situation at a 4-way intersection. (c)* 20 *agents distributing themselves evenly in a narrow passage, to form lanes both in directions. (d) A complex environment requiring careful foot placement to obtain a solution.*

need not plan simultaneously. Once an agent computes an initial plan, it can execute the plan with efficient update operations until it is allocated more deliberation time. If its most imminent plan is invalidated, it is prioritized over other agents and remains stationary till computational resources are available. This ensures that the simulation meets the desired framerate.

7.3 Scenarios

We demonstrate the benefits of our framework by solving many challenging scenarios (Figure 5) requiring space-time precision, explicit coordination between interacting agents, and the factoring of dynamic information (obstacles, moving platforms, user-triggered changes, and other agents) at all stages of the decision process. All results shown here were generated at 30 fps or higher, which includes rendering and character animation. We use an extended version of the ADAPT character animation system [Johansen 2009] for the results shown in the video.

Deadlocks. Multiple oncoming and crossing agents in narrow passageways cooperate with each other with space-time precision to prevent potential deadlocks. Agents observe the presence of dynamic entities at waypoints along their global path and refine their plan if they notice potentially blocked passageways or other high cost situations. Crowd simulators deadlock for these scenarios, while a space-time planner does not scale well for many agents.



Our framework Unity navigation and steering

Figure 7: *Trajectory comparison of our method with an off the shelf predictive steering algorithm in the Unity game engine. Our framework minimizes deviation and uses speed variations to avoid collisions in space-time.*

Choke Points. This scenario shows our approach handling agents arriving at a common meeting point at the same time, producing collision-free straight trajectories. Figure 7 compares the trajectories produced using our method with an off the shelf navigation and predictive collision avoidance algorithm in the Unity game engine. Our framework produces considerably smoother trajectories and minimizes deviation by using subtle speed variations to avoid collisions in space-time.

Unpredictable Environment Change. Our method efficiently repairs solutions in the presence of unpredictable world events, such as the user-placement of obstacles or other agents, which may invalidate current paths.

Road Crossing. The road crossing scenario demonstrates 40 agents using space-time planning to avoid fast moving vehicles and other crossing agents.

Lane Selection for Bi-directional Traffic. This scenario requires agents to make a navigation decision in choosing one of 4 lanes created by the dividers. Agents distribute themselves among the lanes, while bi-directional traffic chooses different lanes to avoid deadlocks. This scenario requires non-deterministic dynamic information (other agents) to be accounted for while making global navigation decisions. This is different from emergent lane formation in crowd approaches, which bottlenecks at the lanes and cause deadlocks without a more robust navigation technique.

Four-way Crossing We simulate 100 oncoming and crossing agents in a four-way crossing. The initial global plans in Σ_1 take the minimum distance path through the center of the crossing. However, Σ_2 predicts a space-time collision between groups at the center and performs plan refinement so that agents deviate from their optimal trajectories to minimize group interactions. A predictive steering algorithm only accounts for imminent neighboring threats and is unable to avoid mingling with the other groups (second row of Figure 7).

Space-Time Goals. We demonstrate a complex scenario where 4 agents in focus (additional agents are also simulated) have a temporal goal constraint, defined as an interval $(40 + / - 1second)$. Agents exhibit space-time precision while jumping across moving planes to reach their target and the temporal goal significantly impacts the decision making at all levels, where the space-time domain maybe unable to meet the temporal constraint and require plans to be modified in earlier domains. No other approach can solve this with real-time constraints.

Many of these scenarios *cannot* be solved by the current state of the art in multi-agent motion planning, which is able to either handle a single agent with great precision, or simulate many simple agents that exhibit reactive collision avoidance.

7.4 Framework Extensibility

The potential of our framework lies in the ability to use multiple domains of control, and is not limited to the domains described in Section 4.1, which only serve as a sufficient set to showcase the benefits of our method. For example, the scenario shown in Figure 5(d) requires careful control over how the character chooses its footsteps and cannot be solved by Σ_4, which does not model bipedal locomotion.

We add a footstep domain Σ_5, which models the motion of the character's center of mass and feet placement using an inverted spherical pendulum model for bipedal locomotion. The agent state $s = (\mathbf{x}, \mathbf{v}, \mathbf{f}_x, f_\phi, \mathrm{I} \in \{\mathtt{L}, \mathtt{R}\})$ includes the center of mass position and velocity, the position and orientation of the current support foot, and an indicator function for the swing foot. An action is chosen by selecting the time period of the footstep, the orientation and speed of the center of mass at the end of the step, and the orientation of the foot plant. The space of possible values of these parameters, while satisfying the constraints enforced by the inverted pendulum model, defines the action space of Σ_5. We discretize this space to keep the set of possible footstep transitions at each state to approximately 35. For implementation details of this domain, please refer to [Singh et al. 2011b].

$\lambda(s, \Sigma_4, \Sigma_5)$ maps states in Σ_4 to one or more states in Σ_5 in order to define a tunnel $\tau(\Sigma_5, \Pi(\Sigma_4), t_w)$ around $\Pi(\Sigma_4)$. We start from a default double support configuration of the character at the start and assume that the character takes a left foot stride first. The COM position is used to define a set of valid positions of the support foot at each space-time waypoint, and $\mathbf{f}_x$ is constrained based on the future COM position (where the character turns to next).

8 Discussion

Choice of Domains. The domains described in this paper represent popular solutions that are used in both academia and industry. Navigation meshes (Σ_1) are a standard solution [Mononen 2009] for representing free spaces in arbitrarily large, complex, static environments with recent proposed extensions [van Toll et al. 2012] that account for dynamic information (Σ_2). A grid-based representation (Σ_3) provides a uniform discretization of the environment, and is widely used in robot motion planning [Koenig and Likhachev 2002; Likhachev et al. 2005]. The introduction of time as a third dimension (Σ_4) enables collision checks in the future, facilitating more robust collision resolution.

These domains provide a nice balance between global navigation and space-time planning, enabling us to showcase the strength of our framework: the ability to use multiple domains of control, and leverage solutions across domains to accelerate computations while still providing a high degree of control fidelity. Additional domains can be easily integrated (e.g., a footstep domain) to meet application-specific needs, or solve more challenging motion planning problems.

Relationship Between Domains. Domains can be connected by using the plan from one domain as a tunnel for the other, or by using successive waypoints along the plan as start and goal pair for multiple planning tasks in a more complex domain. We evaluated both domain relationships based on computational efficiency and coverage, as shown in Table 1. Using waypoints from the navigation mesh domain as start, goal pairs for planning tasks in the grid and space-time domain keeps the search depth for Σ_3 and Σ_4 within reasonable bounds. The tradeoff is that a space-time plan is never generated at a global level from an agent's start position to its target, thus sacrificing completeness guarantees. This design choice worked well for our experiments where the reduction in success rate of our framework when using this scheme was within reasonable bounds, while providing a considerable performance boost, making it suitable for practical game-like applications. Users may wish to opt for different domain relationships depending on the application.

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. In *SIGGRAPH*, ACM, 483–490.

BOTEA, A., MLLER, M., AND SCHAEFFER, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development 1*, 7–28.

BULITKO, V., STURTEVANT, N., LU, J., AND YAU, T. 2007. Graph abstraction in real-time heuristic search. *J. Artif. Int. Res. 30*, 1 (Sept.), 51–100.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22*, 2 (Apr.), 182–203.

CHOI, M. G., KIM, M., HYUN, K., AND LEE, J. 2011. Deformable motion: Squeezing into cluttered environments. *Comput. Graph. Forum 30*, 2, 445–453.

FRAICHARD, T. 1999. Trajectory planning in a dynamic workspace: a'state-time space'approach. *Advanced Robotics 13 6*, 8, 7594.

GOCHEV, K., COHEN, B. J., BUTZKE, J., SAFONOVA, A., AND LIKHACHEV, M. 2011. Path planning with adaptive dimensionality. In *SOCS*.

HOLTE, R. C., PEREZ, M. B., ZIMMER, R. M., AND MACDONALD, A. J. 1996. Hierarchical A *: searching abstraction hierarchies efficiently. In *National conference on Artificial intelligence*, AAAI Press, AAAI, 530–535.

HOLTE, R., GRAJKOWSKI, J., AND TANNER, B. 2005. Hierarchical heuristic search revisited. In *Abstraction, Reformulation and Approximation*, vol. 3607 of *LNCS*. Springer Berlin Heidelberg, 121–133.

HSU, D., KINDEL, R., LATOMBE, J.-C., AND ROCK, S. 2002. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research 21*, 3, 233–255.

JOHANSEN, R. S. 2009. *Automated Semi-Procedural Animation for Character Locomotion*. Master's thesis, Aarhus University.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *ACM SIGGRAPH/Eurographics SCA*, 159–168.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Symposium on Interactive 3D graphics and games*, ACM, I3D, 215–223.

KOENIG, S., AND LIKHACHEV, M. 2002. D* Lite. In *National Conf. on AI*, AAAI, 476–483.

KRING, A. W., CHAMPANDARD, A. J., AND SAMARIN, N. 2010. DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds. In *AIIDE*, The AAAI Press.

LACAZE, A. 2002. Hierarchical planning algorithms. In *SPIE Int. Symposium on Aerospace/Defense Sensing, Simulation, and Controls*.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. In *Computer Graphics Forum 23*.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *ACM SIGGRAPH/Eurographics SCA*, 271–280.

LEVINE, S., LEE, Y., KOLTUN, V., AND POPOVIĆ, Z. 2011. Space-time planning with parameterized locomotion controllers. *ACM Trans. Graph. 30* (May), 23:1–23:11.

LIKHACHEV, M., GORDON, G. J., AND THRUN, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *NIPS*.

LIKHACHEV, M., FERGUSON, D. I., GORDON, G. J., STENTZ, A., AND THRUN, S. 2005. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *ICAPS*, 262–271.

LO, W.-Y., AND ZWICKER, M. 2008. Real-time planning for parameterized human motion. In *ACM SIGGRAPH/Eurographics SCA*, 29–38.

LOPEZ, T., LAMARCHE, F., AND LI, T.-Y. 2012. Space-time planning in changing environments : using dynamic objects for accessibility. *CAVW 23*, 2, 87–99.

LOSCOS, C., MARCHAL, D., AND MEYER, A. 2003. Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG*, IEEE, 122.

MONONEN, M., 2009. Recast: Navigation-mesh construction toolset for games. http://code.google.com/p/recastnavigation/.

PARIS, S., PETTRÉ, J., AND DONIKIAN, S. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS 2007*, vol. 26, 665–674.

PEARL, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2008. *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation.

PETTRÉ, J., KALLMANN, M., AND LIN, M. C. 2008. Motion planning and autonomy for virtual humans. In *ACM SIGGRAPH classes*, 1–31.

PHILLIPS, M., AND LIKHACHEV, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *ICRA*, 5628–5635.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH*, 25–34.

SAFONOVA, A., AND HODGINS, J. K. 2007. Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH*.

SHAPIRO, A., KALLMANN, M., AND FALOUTSOS, P. 2007. Interactive motion correction and object manipulation. In *ACM SIGGRAPH I3D*.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D, 141–150 PAGE@9.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

STURTEVANT, N., AND GEISBERGER, R. 2010. A comparison of high-level approaches for speeding up pathfinding. 76–82.

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *VRST*, ACM, 99–106.

SUNG, M., KOVAR, L., AND GLEICHER, M. 2005. Fast and accurate goal-directed motion synthesis for crowds. In *ACM SIGGRAPH/Eurographics SCA*, 291–300.

THALMANN, D. 2008. Crowd simulation. In *Wiley Encyclopedia of Computer Science and Engineering*.

VAN DEN BERG, J., FERGUSON, D., AND KUFFNER, J. 2006. Anytime path planning and replanning in dynamic environments. In *ICRA*, 2366 –2371.

VAN DEN BERG, J., LIN, M. C., AND MANOCHA, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of ICRA*, IEEE, 1928–1935.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *ACM SIGGRAPH I3D*, 139–147.

VAN TOLL, W. G., COOK, A. F., AND GERAERTS, R. 2012. Real-time density-based crowd simulation. *CAVW 23*, 1, 59–69.

A Events

Tasks monitor the following events:

START_CHANGED: A task receives this event when the start state changes. This event may be triggered when the agent moves, changing its current position – or by the propagation of changes through the task dependency chain. This event triggers an update to account for the change in start state, requiring plan refinement, and potentially invalidating the current solution if the new start position does not lie along the path.

GOAL_CHANGED: A task receives this event when the desired goal state changes. This event triggers an update to account for change in goal and invalidates the current plan. When waypoints along $\Pi(\Sigma_2)$ change, it triggers start and goal updates for tasks in Σ_3 which are responsible for generating a path between the waypoints.

WORLD_CHANGED: Different domains account for world changes at different levels. The global navigation mesh domain Σ_1 which considers only static immovable geometry does not monitor this event. The dynamic navigation mesh domain Σ_2 keeps track of the number of dynamic objects in each polygon of its triangulation which contributes to the cost of the traversal. The grid domain Σ_3 accounts for the current position of obstacles and agents in its plan vicinity. The space-time domain Σ_4 monitors for deviation in the plans of neighboring agents which it accounts for while planning. Note that event registration for WORLD_CHANGED is based on spatial and temporal locality. Tasks monitor this event only for aspects of the environment that may change the current plan or are contained in the visibility frustum of the agent. This ensures that planners only consider changes in the environment of interest which require an update.

TUNNEL_CHANGED: Planners can exploit plans in one domain in order to accelerate searches in another domain. For example, the path computed in Σ_3 can be used to focus and accelerate the search in Σ_4. Tasks with this dependency must monitor other tasks and repair its own solution when the plan changes. Section 6.3 describes the use of tunnel search.

PLAN_STATUS_CHANGED: The status of the plan is monitored by the task itself (requiring a change in task priority) and by the task that it is dependent on. An invalid or sub-optimal solution gives the task a higher priority while an optimal solution does not require any further processing. If a task is unable to come up with a solution, it requires a change in task parameters (e.g. increasing the tunnel width to increase the search focus) or it means that current problem definition cannot be solved, requiring a new problem definition from the task higher up in the task dependency chain.

B Algorithmic Details for Planning Task

ExecutePlanTask (Algorithm 1 [28–37]) is invoked each time the planning task is executed. This function monitors events and calls the appropriate event handlers, described in Algorithm 2. Given a maximum amount to deliberate t_{max}, it refines the plan and publishes the ϵ-suboptimal solution using the AD* planning algorithm [Likhachev et al. 2005]. We briefly describe our implementation of the AD* algorithm and how we handle changes in start, goal, obstacle movement, and tunnel updates, and refer the readers to [Likhachev et al. 2005] for more details.

AD* performs a backward search and maintains a least cost path from the goal s_{goal} to the start s_{start} by storing the cost estimate $g(s)$ from s to s_{goal}. However, in dynamic environments, edge costs in the search graph may constantly change and expanded nodes may become inconsistent. Hence, a one-step look ahead cost estimate $rhs(s)$ is introduced [Koenig and Likhachev 2002] to determine node consistency.

```
1  key(s)
2    if g(s) > rhs(s) then
3        return [rhs(s) + ε · h(s, s_start); rhs(s)]
4    else
5        return [g(s) + ·h(s, s_start); g(s)]
6  UpdateState(s)
7    if (s ≠ s_goal) then
8        s' = arg min_{s' ∈ pred(s)} (c(s, s') + g(s'))
9        rhs(s) = c(s, s') + g(s')
10       prev(s) = s'
11   if (s ∈ OPEN) remove s from OPEN
12   if g(s) ≠ rhs(s) then
13       if (s ∉ CLOSED) insert s in OPEN with key(s)
14       else insert s in INCONS
15   Insert s in VISITED
16 ComputeOrImprovePath (t_max)
17   while (min_{s∈OPEN}(key(s)) < key(s_start) ∨ rhs(s_start) ≠
     g(s_start) ∨ Π(s_start, s_goal) = NULL) ∧ t < t_max do
18       s = arg min_{s∈OPEN} (key(s))
19       if (g(s) > rhs(s)) then
20           g(s) = rhs(s)
21           CLOSED = CLOSED ∪ s
22       else
23           g(s) = ∞
24           UpdateState(s)
25       foreach s' ∈ succ(s) do
26           UpdateState(s')
27 ExecutePlanTask (t_max)
28   Move states from INCONS to OPEN
29   CLOSED = NULL
30   if START_CHANGED then StartChangeUpdate (s_c)
31   if GOAL_CHANGED then GoalChangeUpdate (s_new)
32   if WORLD_CHANGED then
33       foreach (obstacle change s → s' ) ObstacleChangeUpdate (s,s')
34   if TUNNEL_CHANGED then
35   TunnelChangeUpdate (Π'(Σ_ld, s_start, s_goal))
36   ComputeOrImprovePath (t_max)
37   trigger PLAN_STATUS_CHANGED
```

Algorithm 1: AD* Planner used to compute and update paths for planning tasks $T(\Sigma)$ in each of the 4 domains.

$$rhs(s) = \begin{cases} 0 \text{ if } s = s_{goal} \\ \arg\min(c(s, s') + g(s')) \text{ else} \end{cases} \quad (7)$$

The priority queue OPEN contains the states that need to be expanded for every plan iteration, with the priority defined using a lexicographic ordering of a two-tuple **key**(s), defined for each state. OPEN contains only the inconsistent states ($g(s) \neq rhs(s)$) which need to be updated to become consistent. Nodes are expanded in increasing priority until there is no state with a key value less than the start state. A heuristic function $h(s, s')$ computes an estimate of the optimal cost between two states, and is used to focus the search towards s_{start}.

Instead of processing all inconsistent nodes, only those nodes whose costs may be inconsistent beyond a certain bound, defined by the inflation factor ϵ are expanded. It performs an initial search with an inflation factor ϵ_0 and is guaranteed to expand each state only once. An INCONS list keeps track of already expanded nodes that become inconsistent due to cost changes in neighboring nodes. Assuming no world changes, ϵ is decreased iteratively and plan quality

```
1  StartChangeUpdate (s_c)
2    if s_c ∉ Π(s_start, s_goal) ∧ d(s_c, Π(s_start, s_goal)) > t_max then
3        ClearPlanData()
4        ε = ε_0
5    else
6        s_start = s_c
7        foreach s ∈ OPEN do
8            Update key(s)

9  GoalChangeUpdate (s_new)
10   ClearPlanData()
11   ε = ε_0
12   s_goal = s_new
13 ObstacleChangeUpdate (s,s')
14   if s' ∈ Π(s_start, s_goal) then
15       Π(s_start, s_goal) = Π(s_start, s_goal) - s'
16       ε = ε_0
17   if pred(s) ⋂ VISITED ≠ NULL then UpdateState(s)
18   g(s') = ∞
19   if s' ∈ CLOSED then
20       foreach s'' ∈ succ(s') do
21           if s'' ∈ VISITED then UpdateState(s'')
22 TunnelChangeUpdate (Π(s_start, s_goal))
23   foreach s ∈ VISITED do
24       if |d(s, Π(s_start, s_goal))| > t_w then
25           g(s) = ∞
26           if s ∈ CLOSED then
27               foreach s' ∈ succ(s) do
28                   if s' ∈ VISITED then UpdateState(s')
29       else
30           h_t(s, s_start) = h(s, s_start) + |d(s, Π(s_start, s_goal))|
```

Algorithm 2: Event handlers for change in start state, goal state, environment, and tunnel.

is improved until an optimal solution is reached ($\epsilon = 1$). Each time ϵ is decreased, all states made inconsistent due to change in ϵ are moved from INCONS to OPEN with **key**(s) based on the reduced inflation factor, and CLOSED is made empty. This improves efficiency since it only expands a state at most once in a given search and reconsidering the states from the previous search that were inconsistent allows much of the previous search effort to be reused, requiring only a minor amount of computation to refine the solution. **ComputeOrImprovePath** (Algorithm 1 [16–26]) gives the routine for computing or refining a path from s_{start} to s_{goal}.

When change in edge costs are detected, new inconsistent nodes are placed into OPEN and node expansion is repeated until a least cost solution is achieved within the current ϵ bounds. When the environment changes substantially, it may not be feasible to repair the current solution and it is better to increase ϵ so that a less optimal solution is reached more quickly.

An increase in edge cost may cause states to become under-consistent ($g(s) < rhs(s)$) where states need to be inserted into OPEN with a key value reflecting the minimum of their old cost and their new cost. In order to guarantee that under-consistent states propagate their new costs to their affected neigbhors, their key values must use uninflated heuristic values. This means that different key values must be computed for under- and over-consistent states, as shown in Algorithm 1 [1 – 5]. This key definition allows AD* to efficiently handle changes in edge costs and changes to inflation factor.

AD* uses a backward search to handle agent movement along the plan by recalculating key values to automatically focus the search repair near the updated agent state. It can handle changes in edge costs due to obstacle and start movement, and needs to plan from scratch each time the goal changes. The routines to handle change in start, goal, and world changes are described below.

StartChangeUpdate. When the start moves along the current plan, the key values of all states in OPEN are recomputed to re-prioritize the nodes to be expanded. This focuses processing towards the updated agent state allowing the agent to improve and update its solution path while it is being traversed. When the new start state deviates substantially from the path, it is better to plan from scratch. Alg 2 [1–8] provides the routine to handle start movement.

GoalChangeUpdate. Alg 2 [9–12] clears plan data and resets ϵ whenever the goal changes and plans from scratch at the next step.

ObstacleChangeUpdate. Alg 2 [13–21] handles change in obstacles. An obstacle movement from s to s' results in a free state at s and an invalidation of the previously valid state s'. Nodes in the vicinity of the obstacle movement (i.e., successors of s and s') become inconsistent and may have invalid references to s', which is no longer free, requiring them to be updated. If the obstacle movement invalidates the current plan, we reset ϵ to quickly produce a valid path at the next step, which can be refined in subsequent iterations.

TunnelChangeUpdate. This routine is used when the planning task monitors the computed path of another planning task $T(\Sigma_{ld})$ in a lower-dimensional domain to focus and accelerate its own searches, as described in Section 6.3.

C Performance of Tunnel Search

We evaluate the performance of tunnel-based search on 100 randomly sampled problem definitions (environment configuration, start and goal state) in the Σ_4 with a constraint enforcing the maximum Euclidean distance between s_{start} and s_{goal} to be 20 grid units. This corresponds to comparable problem definitions for these planners in our multi-domain framework. For a given problem instance, we first execute $T(\Sigma_3)$ to generate a spatial path $\Pi(\Sigma_3)$ which is used to focus the search in $T(\Sigma_4, \Pi(\Sigma_3))$. In addition, we solve the problem instance without a tunnel constraint to provide a basis for comparison.

Table 2 provides the number of nodes expanded and the total planning time for the three planning tasks. The aggregate performance of using $T(\Sigma_3)$ and $T(\Sigma_4, \Pi(\Sigma_3))$ is provided for reference. We notice that tunnel greatly expedites the search process by expanding $4X$ fewer nodes, and providing a $3X$ performance boost on average. Out of the 100 scenarios, 8 scenarios resulted in the tunnel search not being able to initially find a solution and had to increase its tunnel width and replan. Even in these cases, the tunnel search outperformed $T(\Sigma_4)$. For 3 scenarios, $T(\Sigma_4)$ could not generate a solution within the maximum time allotment of $100ms$. These were problem instances with a local minima where the search heuristic alone falsely focused the search down a wrong path but the use of the tunnel mitigated the need of the exploration of local minima in the space-time domain.

Planning Task	# of nodes	Time (ms)
$T(\Sigma_3)$	47	3.5
$T(\Sigma_4, \Pi(\Sigma_3))$	246	7.3
$T(\Sigma_3) + T(\Sigma_4, \Pi(\Sigma_3))$	293	10.8
$T(\Sigma_4)$	1041	21.2

Table 2: *Performance evaluation of using tunnel based search.*