Proceedings of the Ninth International
Symposium on Combinatorial Search (SoCS 2016)

Path Planning in Dynamic Environments with Adaptive Dimensionality

Anirudh Vemula, Katharina Muelling, and **Jean Oh**
Robotics Institute
Carnegie Mellon University
avemula1@andrew.cmu.edu, {kmuelling, jeanoh}@nrec.ri.cmu.edu

Abstract

Path planning in the presence of dynamic obstacles is a challenging problem due to the added time dimension in the search space. In approaches that ignore the time dimension and treat dynamic obstacles as static, frequent re-planning is unavoidable as the obstacles move, and their solutions are generally sub-optimal and can be incomplete. To achieve both optimality and completeness, it is necessary to consider the time dimension during planning. The notion of adaptive dimensionality has been successfully used in high-dimensional motion planning such as manipulation of robot arms, but has not been used in the context of path planning in dynamic environments. In this paper, we apply the idea of adaptive dimensionality to speed up path planning in dynamic environments for a robot with no assumptions on its dynamic model. Specifically, our approach considers the time dimension only in those regions of the environment where a potential collision may occur, and plans in a low-dimensional state-space elsewhere. We show that our approach is complete and is guaranteed to find a solution, if one exists, within a cost sub-optimality bound. We experimentally validate our method on the problem of 3D vehicle navigation (x, y, heading) in dynamic environments. Our results show that the presented approach achieves substantial speedups in planning time over 4D heuristic-based A*, especially when the resulting plan deviates significantly from the one suggested by the heuristic.

1 Introduction

It is important for mobile robots to be able to generate collision-free paths in environments that contain both static and dynamic obstacles. In static environments, robots can efficiently generate a collision-free path using the occupancy gridmap of the environment. But in dynamic environments, to account for the dynamic nature of obstacles, the robot needs to predict the future trajectories of these obstacles to plan its own path accordingly. These predictions involve a high degree of uncertainty due to sensor limitations and incorrect dynamic models. As a result, the predicted trajectories are subject to frequent changes due to incorrect predictions, which makes it necessary to generate new plans in a timely manner.

To account for dynamic obstacles in an environment, we need to include the time dimension into consideration. For example, planning a path for a non-holonomic robot in a dynamic environment involves a 4D state-space, i.e., $(x, y,$ heading, time). Due to the curse of dimensionality, adding the time dimension substantially increases the number of states to be searched, e.g., from 3D state-space considering only $(x, y,$ heading), leading to long planning times especially, since there are potentially an unbounded number of timesteps for each spatial location.

The Adaptive Dimensionality (AD) approach, (Gochev et al. 2011), exploits the observation that while planning in a high dimensional space is needed to satisfy kinematic constraints and collision-free criteria, large portions of the path are still low dimensional. For instance, in planning for a non-holonomic robot, an optimal path generally includes straight-line segments that do not involve any turns or collisions with dynamic obstacles. This observation implies that high dimensional path planning is required only in the sections of the path where turning is required or where there is a potential collision with a dynamic obstacle.

Following this insight, we consider the time dimension only in those regions where a potential collision could occur and ignore it elsewhere. In this paper, we develop an approach that can achieve speedups over full-dimensional heuristic-based A* without any assumptions on robot capabilities by employing a variant of the Adaptive Dimensionality approach.

In the remainder of this paper, we will give an overview of relevant existing work in Section 2. The planning problem is formally defined in Section 3 and the motivation for our approach is presented in Section 4. Sections 5 and 6 will describe our approach and prove the theoretical guarantees. The efficiency of the method is demonstrated by applying it to a 3D non-holonomic robot navigation problem in the presence of dynamic obstacles, showing a significant increase in speed over 4D heuristic-based A* planner for this task, in Sections 7 and 8.

2 Related Work

Our work is relevant to path planning in dynamic environments and works on coping with high dimensionality. In general, we divide the existing approaches into three categories: works that deal with planning in dynamic environments, that deal with high-dimensional planning using adaptive dimensionality and that use hybrid dimensionality path

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

planning in dynamic environments.

2.1 Path Planning in Dynamic Environments

A common approach used for efficient path planning in dynamic environments involves modeling moving obstacles as static objects with a small window of high cost around the beginning of their projected trajectories (Likhachev and Ferguson 2009; Rufli, Ferguson, and Siegwart 2009).

By avoiding the additional time dimension, these approaches can efficiently find paths that do not collide with any obstacles in the near future. However, they can suffer from severe sub-optimality or even incompleteness due to the uncertainty of moving obstacles in the future.

To plan and re-plan online, several approaches have been suggested that sacrifice near-optimality guarantees for efficiency (Van Den Berg, Ferguson, and Kuffner 2006), including sampling-based planners such as RRT-variants that can quickly obtain kinodynamically feasible paths in a high dimensional space (Bekris and Kavraki 2007; Petti and Fraichard 2005). However, these sampling-based approaches do not provide any global optimality guarantees that we require in most cases.

Other approaches (Fox, Burgard, and Thrun 1997; Brock and Khatib 1999) delegate the dynamic obstacle avoidance problem to a local reactive planner which can effectively avoid collision with dynamic obstacles. These methods have the disadvantage that they can get stuck in local minima and are generally not globally optimal.

Among works that provide global optimality guarantees that are relevant to our work, HCA* (Silver 2005) is an approach that plans in the full space-time search space for a path from start to goal, taking dynamic obstacles into account under the guidance of a low-dimensional heuristic. In dynamic environments, HCA* provides guarantees on optimality and can be applied to path planning for a robot without any assumptions on its motion model.

Recently, approaches such as SIPP and its variants (Phillips and Likhachev 2011; Narayanan, Phillips, and Likhachev 2012), have been introduced that obtain fast, globally optimal paths in dynamic environments. But, SIPP assumes that the robot is capable of waiting in place. In cases where this assumption doesn't hold, SIPP is essentially a full space-time A* planner. Thus, the advantages from this algorithm are restricted to only those robots which have the capability of waiting in place, unlike a fixed wing aircraft or a motorcycle. Besides, when fuel efficiency is included in the cost, fuel consumption is generally higher during idling than moving.

We use HCA* as our baseline algorithm as it doesn't make any assumptions on the motion model of the robot and provides optimality guarantees, similar to our approach.

2.2 Adaptive Dimensionality

To accelerate planning, a variety of algorithms try to avoid global planning in high-dimensional state-space. In these algorithms, planning is split into a two-layer process where a global planner deals with a low-dimensional state-space and provides an input to a high-dimensional local planner (Philippsen and Siegwart 2003). The local planner is a reactive planner that avoids obstacles locally and hence, is fast and efficient. However, these approaches can result in paths that are highly suboptimal or that cannot be executed, due to mismatches in the assumptions made by the global and local planners.

In (Knepper and Kelly 2006), highly accurate heuristic values are computed by solving a low-dimensional problem and are then used to direct high-dimensional planning. However, this approach does not explicitly decrease the dimensionality of the state-space and can lead to long planning times when the heuristic is incorrect.

By contrast, the Adaptive Dimensionality (AD) approach, (Gochev et al. 2011), explicitly decreases the dimensionality of the state-space in regions where full-dimensional planning is not needed. This approach introduces a strategy for adapting the dimensionality of the search space to guarantee a solution that is still feasible with respect to a high dimensional motion model while making fast progress in regions that exhibit only low-dimensional structure. In (Gochev, Safonova, and Likhachev 2012), path planning with adaptive dimensionality has been shown to be efficient for high-dimensional planning such as mobile manipulation. The AD approach has been extended in (Gochev, Safonova, and Likhachev 2013), to get faster planning times by introducing an incremental planning algorithm. (Zhang, Butzke, and Likhachev 2012) extends this method in the context of mobile robots by using adaptively dimensional state-space to combine the global and local path planning problem for navigation. Our approach builds on the AD approach and applies it to path planning in dynamic environments.

2.3 Hybrid Dimensionality in Dynamic Environments

Some approaches only plan in full-dimensional space-time search space until the end of an obstacle's trajectory and then finish the plan in a low-dimensional state-space. Time-bounded lattice planning, (Kushleyev and Likhachev 2009), neglects dynamic obstacles and the time dimension in the search space after a certain point in the time. Several works, (Petereit, Emter, and Frey 2013; 2014), have extended this algorithm to account for kinematic and dynamic feasibility in the resulting paths by using a hybrid dimensionality state-space. These approaches sacrifice optimality for faster planning times and don't provide theoretical guarantees on the sub-optimality of the solution. In addition, our algorithm doesn't prune the dynamic obstacle trajectories, instead takes the entire obstacle trajectories into account and returns a bounded sub-optimal collision-free path. Considering the entire trajectory of the obstacles ensures a globally optimal solution.

3 Problem Definition

In this paper, we follow the simplifying assumptions used in (Phillips and Likhachev 2011) that the trajectories of moving obstacles are known and that obstacles move at a constant speed. Based on these assumptions, path planning in a dynamic environment can be formulated more generally as path planning in a high-dimensional space as follows. A path planning problem is defined as a tuple $\Phi = [G = (S, T), c, X_s, X_g]$, where G denotes a graph consisting of S, a set of discretized states in a d-dimensional space, and T, a set of feasible transitions between each pair of states $X_i, X_j \in S$; c, a function encoding a non-negative cost $c(X_i, X_j)$ for each pair of transitions $(X_i, X_j) \in T$; $X_s \in S$, a start state, and $X_g \in S$, a goal state. For instance, the target problem for a ground vehicle can be defined in 4-D state space (x, y, θ, t) where each variable denotes x-coordinate, y-coordinate, vehicle heading, and time, respectively. Note that transitions that result in collision with an obstacle are assigned infinite cost, making them invalid.

A path between states X_i and X_j is denoted by $\pi(X_i, X_j)$, and the cost of a path is defined as the sum of all transition costs in the path.

Given a planning problem $\Phi = [G, c, X_s, X_g]$, the goal is to find a minimum cost path between the two states X_s and X_g, denoted by $\pi^*(X_s, X_g)$. Alternatively, given a suboptimality bound ϵ, the goal of the planner can be relaxed to find a path $\pi(X_S, X_G)$ such that its cost $c(\pi(X_S, X_G)) \leq \epsilon \cdot c(\pi^*(X_S, X_G))$.

4 Motivation

Given a path planning problem in a high dimensional space, it is possible to find an optimal solution through a complete search. For example, heuristic-based A* variant algorithms exist that are guaranteed to find an optimal solution (Silver 2005). Because these algorithms rely on low-dimensional heuristics, search can be counter-intuitive.

Consider the example shown in Figure 1, where the resulting path (dash-dot path), in the absence of the dynamic obstacle (disc), is towards the heuristic. But, in the presence of the dynamic obstacle, this path is in collision and cannot be executed. Hence, we need to come up with the alternative path (dashed path), which is against the heuristic. Heuristic-based A* would expand a large number of states and will take a long time to generate the new path whereas our approach generates the alternative path quickly, because it plans in a lower dimensional space.

More generally, we observe that substantial sections of paths found are not in collision with any dynamic obstacles, implying that we need not consider the time dimension in such regions. We can obtain quicker planning times by planning in low dimensional state-space for those regions and in full dimensional state-space only where it is necessary to reason about a potential collision with an obstacle.

Based on these observations, we explore the idea of adaptive dimensionality to solve the target problem.

5 Approach

We describe the adaptive dimensionality approach used for path planning in dynamic environments, and the algorithm for finding a bounded cost sub-optimal path.

5.1 Adaptive Dimensionality for Dynamic Environments

Our approach follows the algorithm for planning with adaptive dimensionality introduced in (Gochev et al. 2011). Following their notation, the target problem in Section 3 can be rewritten as follows. Graph G is substituted with the adaptive-dimensionality graph $G^{ad} = (S^{ad}, T^{ad})$. G^{ad} is constructed from two graphs: a high dimensional graph $G^{hd} = (S^{hd}, T^{hd})$ with dimensionality h and a low dimensional graph $G^{ld} = (S^{ld}, T^{ld})$ with dimensionality l. The state-space S^{ld} is a projection of S^{hd} onto a lower dimensional manifold ($h>l$) through a projection function:

$$\lambda : S^{hd} \rightarrow S^{ld} \tag{1}$$

Similarly, an inverse projection function $\lambda^{-1} : S^{ld} \rightarrow \mathcal{P}(S^{hd})$ is defined to map low-dimensional states to the set of all their high-dimensional pre-images,where $\mathcal{P}(S^{hd})$ denotes the power set of S^{hd}.

Both state spaces S^{hd} and S^{ld} can have their own transition sets T^{hd} and T^{ld}, with a constraint that transitions in a high-dimensional space are more expensive than the corresponding transitions in a low-dimensional space, that is for every pair of states X_i and X_j in S^{hd},

$$c(\pi^*_{G^{hd}}(X_i, X_j)) \geq c(\pi^*_{G^{ld}}(\lambda(X_i), \lambda(X_j))) \tag{2}$$

We note that this constraint is important for bounding the suboptimality that will be discussed later in Section 6.

In our target problem of planning in a dynamic environment, the low-dimensional state-space S^{ld} consists of only spatial state variables, e.g., xy-coordinates, and the high-dimensional state-space S^{hd} consists of states with spatio-temporal variables including a time dimension. Theoretically, the time dimension is unbounded and thus, the high-dimensional graph G^{hd} is an infinite graph. For practical purposes, we bound the time dimension by a upper bound T, which would slightly modify the goal of planning problem into finding a least-cost path that can reach the goal from start within time T. For any high dimensional state $X^{hd} \in S^{hd}$, we will use the notation $t(X^{hd})$ to denote the value of time dimension associated with that state.

The projection function λ projects the high-dimensional state X^{hd} to a low-dimensional state X^{ld} with only the spatial variables. If we follow the original definition of λ in

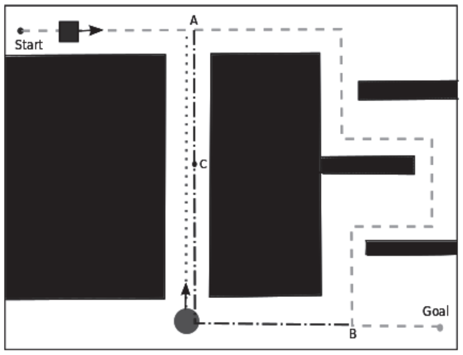
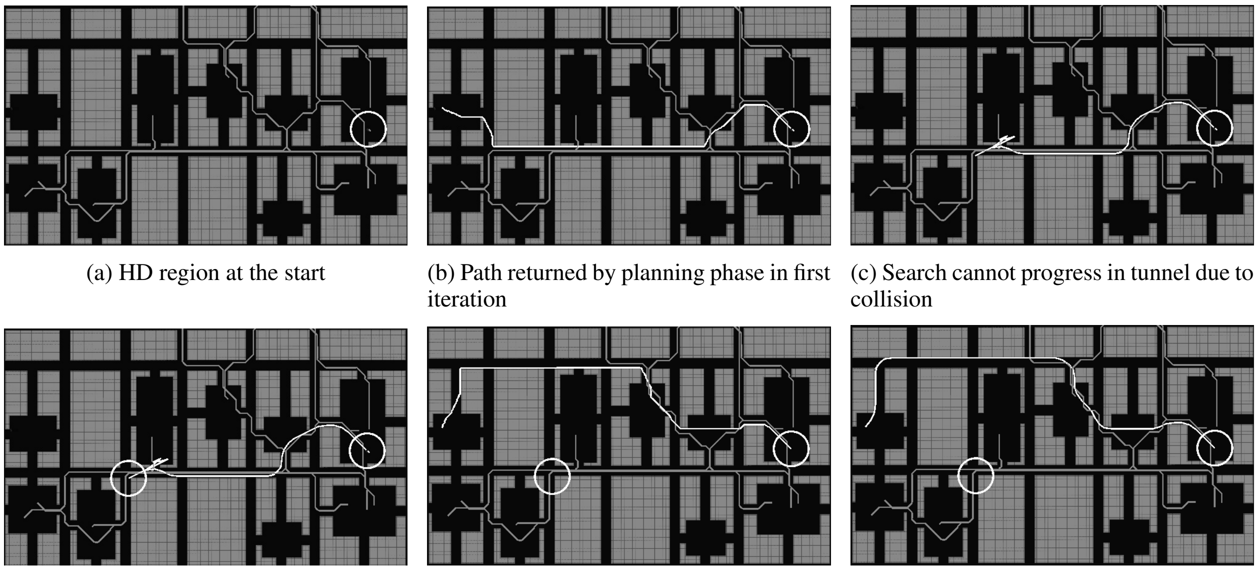


Figure 1: Example of a dynamic environment where the heuristic leads the robot (square) into collision (on dash-dot path) with the dynamic obstacle (disc) at C. We need to find an alternate path (dashed path) from A to B without expanding a large number of states.



(a) HD region at the start

(b) Path returned by planning phase in first iteration

(c) Search cannot progress in tunnel due to collision

(d) HD region introduced at point of collision

(e) Path returned by planning phase in second iteration

(f) Tracking successful and path returned as solution

Figure 2: Example run of the algorithm on a sample map. HD regions are indicated by white circles, paths of dynamic obstacles by gray lines and path found using our approach by white lines.

Equation 5.1 then, for a given low-dimensional state X^{ld}, the inverse projection function λ^{-1} would map state X^{ld} to the set of all X^{hd} where the spatial configuration of X^{hd} is the same as X^{ld} and $0 \leq t(X^{hd}) \leq T$. Thus, for each low-dimensional state, there are T corresponding high-dimensional states, which is quite a large number as T is usually a high value.

Here, we introduce a pruning technique based on an observation that not all of high-dimensional states are reachable from the start state. For example, consider a low-dimensional state $X \in S^{ld}$ which is mapped to T high-dimensional states. If the time-optimal path from start to this state, ignoring dynamic obstacles, reaches at time t_f then all the states X^{hd} with $\lambda(X^{hd}) = X$ and $t(X^{hd}) < t_f$ are essentially unreachable and hence, can be pruned away from the search space.

Taking advantage of this fact, we decrease the size of search space by performing a low-dimensional time-optimal Dijkstra search in G^{ld}, which ignores dynamic obstacles, initially from the start state to all low-dimensional states and keep track of the time at which we reach each state. We store this time as a dependent variable t_{dep} of the low-dimensional state and ignore all the corresponding high-dimensional states whose time value is less than t_{dep} in the inverse projection mapping. Note that this dependent time variable need not be the exact optimal time obtained from the Dijkstra search, it just needs to be a lower bound on the optimal time. Pruning the search space in this way is necessary as it speeds up the planning by a considerable amount while still maintaining the completeness property of the planning algorithm.

Thus, we define the inverse projection function, λ^{-1} as:

$$\lambda^{-1}(X^{ld}) = \{X^{hd} \mid \lambda(X^{hd}) = X^{ld}, t_{dep} \leq t(X^{hd}) \leq T\}$$

where t_{dep} is the dependent time variable associated with the low-dimensional state X^{ld}.

The low-dimensional transition set is $T^{ld} = \{(X_i, X_j) | X_i, X_j \in S^{ld}\}$ where it is feasible for the robot to move from the spatial configuration of X_i to X_j according to its motion model. The transition set $T^{hd} = \{(X_i, X_j) | X_i, X_j \in S^{hd}\}$ where, $t(X_j) \geq t(X_i)$ and it is feasible for the robot to move from spatial configuration of X_i to X_j in time $(t(X_j) - t(X_i))$ according to its motion model. Note that we can check for collisions with any dynamic obstacle only in the high-dimensional transitions as we have the time information.

5.2 Algorithm

The planning algorithm follows that of (Gochev et al. 2011). Here, we sketch the general algorithm and describe how it has been applied to our target problem of handling dynamic obstacles.

Adaptive Dimensionality Graph Construction The algorithm iteratively constructs S^{ad}, starting with S^{ld} and introducing high-dimensional regions in subsequent iterations. Once a high-dimensional region is introduced we replace all the low-dimensional states that fall inside it with their high-dimensional counterparts as given by λ^{-1} to get the re-constructed S^{ad} for the next iteration. The transition set T^{ad} is also iteratively constructed, starting with T^{ld} and re-constructed as follows in subsequent iterations. For any state $X_i \in S^{ad}$:

- If X_i is high-dimensional then, for all high-dimensional transitions $(X_i, X_j^{hd}) \in T^{hd}$, if $X_j^{hd} \in S^{ad}$ then $(X_i, X_j^{hd}) \in T^{ad}$. Otherwise, $(X_i, \lambda(X_j^{hd})) \in T^{ad}$.
- If X_i is low-dimensional then, for all low-dimensional transitions $(X_i, X_j^{ld}) \in T^{ld}$, if $X_j^{ld} \in S^{ad}$ then $(X_i, X_j^{ld}) \in T^{ad}$, and for all high-dimensional transitions $(X, X_j^{hd}) \in T^{hd}$ where $X \in \lambda^{-1}(X_i)$, if $X_j^{hd} \in S^{ad}$ then $(X_i, X_j^{hd}) \in T^{ad}$.

Main Loop We start with G^{ad} same as G^{ld} and a high-dimensional region added at the start, which is necessary as the start state X_S is high-dimensional with $t(X_S) = 0$. Note that the goal state X_G is not high-dimensional as we don't know the value of the time dimension for the goal state.

AD planning phase. At the start of each iteration, the current graph G^{ad} is searched for a path $\pi^*_{G^{ad}}$ from the start to the goal, using a suboptimal graph search algorithm like weighted A* with a suboptimality bound ϵ_{plan}. During the search for this path, we consider the dynamic obstacles only in high-dimensional regions of S^{ad} and not in the low-dimensional regions. Hence, the path found could potentially be in collision with a dynamic obstacle in the low-dimensional regions. If no path $\pi^*_{G^{ad}}$ is found, we return that there exists no feasible path that can reach the goal from start within time T, and the algorithm terminates.

Tracking phase. If a path is found, then in the tracking phase, a high-dimensional tunnel τ is constructed around the path $\pi^*_{G^{ad}}$ and searched for the least-cost path $\pi^*_\tau(X_S, X_G^{hd})$ where $X_G^{hd} \in \lambda^{-1}(X_G)$. The tunnel is constructed by projecting all the states within to their high-dimensional counterparts. Notice that since the tunnel is entirely high-dimensional and is a subgraph of G^{hd}, we consider dynamic obstacles in the entire tunnel and hence, the path found is guaranteed to be feasible and collision-free. If a path π^*_τ is found and its cost is less than $\epsilon_{track} * c(\pi^*_{G^{ad}})$, then it is returned as the solution by the algorithm and the algorithm terminates. If no path is found, then we identify the farthest location in the tunnel until which the planner has progressed (i.e. the path with most progress), introduce a high-dimensional region there and move onto next iteration. If a path is found and its cost is greater than $\epsilon_{track} * c(\pi^*_{G^{ad}})$, then we identify the location where the largest cost discrepancy (cost difference) between the path π^*_τ and $\pi^*_{G^{ad}}$ is observed and a high-dimensional region is introduced there. In both cases, if we identify a location which is already high-dimensional, then the size of the high-dimensional region at that location is increased.

Graph updating phase. The algorithm re-constructs G^{ad} based on the new high-dimensional regions introduced and moves onto the next iteration of planning and tracking, and keeps repeating until it finds a feasible, collision-free path or returns that there is no such path. An example run of the algorithm is shown in Figure 2. The algorithm is presented in Algorithm 1.

6 Theoretical Properties

The given algorithm is complete with respect to the underlying graph G^{hd} and provides a sub-optimality bound on the cost of the returned path.

Algorithm 1 Planning with AD in dynamic environments

```
1: $G^{ad} = G^{ld}$
2: AddFullDimRegion($G^{ad}$, $\lambda(X_S)$)
3: loop
4:     Search $G^{ad}$ for the path $\pi^*_{G^{ad}}(X_S, X_G)$
5:     if no $\pi^*_{G^{ad}}(X_S, X_G)$ is found then
6:         return no path from $X_S$ to $X_G$ within time $T$ exists
7:     end if
8:     Construct tunnel $\tau$ around $\pi^*_{G^{ad}}(X_S, X_G)$
9:     Search $\tau$ for the least-cost path $\pi^*_\tau(X_S, X_G^{hd})$ where $X_G^{hd} \in \lambda^{-1}(X_G)$
10:    if no $\pi^*_\tau(X_S, X_G^{hd})$ is found then
11:        Let $\pi(X_S, X_{end})$ be the path with most progress
12:        if $X_{end}$ is high-dimensional then
13:            GrowFullDimRegion($G^{ad}$, $\lambda(X_{end})$)
14:        else
15:            AddFullDimRegion($G^{ad}$, $X_{end}$)
16:        end if
17:    else if $c(\pi^*_\tau(X_S, X_G^{hd})) > \epsilon_{track} * c(\pi^*_{G^{ad}}(X_S, X_G))$ then
18:        Identify state $X_r$ with largest cost discrepancy
19:        if $X_r$ is already high-dimensional then
20:            GrowFullDimRegion($G^{ad}$, $\lambda(X_r)$)
21:        else
22:            AddFullDimRegion($G^{ad}$, $X_r$)
23:        end if
24:    else
25:        return $\pi^*_\tau(X_S, X_G^{hd})$
26:    end if
27: end loop
```

Theorem 5.1 *If a path $\pi^*_\tau(X_S, X_G^{hd})$ is found in the tracking phase, it is guaranteed to be collision-free with respect to all obstacles*

Proof The tunnel τ constructed around $\pi^*_{G^{ad}}$ is entirely high-dimensional and is a subgraph of G^{hd} therefore, the search space considers the transition set T^{hd}. In T^{hd}, transitions that are in collision with dynamic obstacles are assigned infinite cost, essentially making them invalid.

Hence, the path found in the tunnel π^*_τ is guaranteed to be collision-free with respect to all obstacles $\square$

Theorem 5.2 *If no path $\pi^*_{G^{ad}}(X_S, X_G)$ is found in the planning phase, then no collision-free, feasible path exists from start to goal in G^{hd} that can reach the goal from the start within time T*

Proof If no path is found during the planning phase of the first iteration, no feasible path exists in the absence of dynamic obstacles (Note that we only consider dynamic obstacles in the high-dimensional regions during the planning phase). Therefore, there is no collision-free path. If no path is found during the planning phase of any subsequent iteration, then the algorithm was not able to progress in a high-dimensional region. It cannot be a low-dimensional region because in such a case, it would have terminated in a previous iteration. If the algorithm is not able to progress in a high-dimensional region, then all the transitions into or inside the region are blocked by dynamic obstacles. Because we allow transitions to all X^{hd} inside the region where $t_{dep} \leq t(X^{hd}) \leq T$ and we already know that the planner cannot reach X^{hd} at a time earlier than t_{dep}, that guarantees that there exists no path in G^{hd} that can reach the goal from start within time T. □

Theorem 5.3 *The algorithm always terminates after a finite number of iterations*
Proof If no path is found at the end of a iteration, we introduce either a new high-dimensional region or increase the size of an existing one. As the time dimension is bounded above by T, we have a finite state-space. Hence, in the worst scenario, after a finite number of iterations G^{ad} will be the same as G^{hd} and the algorithm will either terminate with a feasible path or return that there is no feasible, collision-free path. □

Theorem 5.4 *If a path $\pi(X_S, X_G^{hd})$ is found at the time of termination, its cost is no more than $\epsilon_{plan} * \epsilon_{track} * c(\pi^*_{G^{hd}}(X_S, X_G^{hd}))$ where $\pi^*_{G^{hd}}(X_S, X_G^{hd})$ is the optimal least-cost path in G^{hd}.*
Proof We obtain this bound using equation 2 and the proof is similar to that of the AD approach. For this proof, we refer the reader to (Gochev et al. 2011). □

7 Experiments

For an experimental evaluation of the presented approach we use the domain of robotic path planning in dynamic environments for 3D-(x, y, θ) non-holonomic robot. To successfully avoid dynamic obstacles in the environment, we will need to add the time dimension to the state-space while planning. Hence, the full-dimensional planning has a 4D (x, y, θ, t) state-space. Our implementation of the algorithm kept track of the high-dimensional regions in the environment as spheres: 2D (x, y) circles, in the 4D planning case, as this allowed to quickly check if a state falls inside a region or not, and also quickly add new regions or grow existing regions.

We modeled our environment as a planar world and the robot as a polygonal object with a unicycle motion model, which doesn't allow waiting in place actions. Our adaptive-dimensionality space consists of a 2D (x, y) low-dimensional state-space and a 4D (x, y, θ, t) high-dimensional state-space, where θ is the heading of the robot. Thus the projection function is:

$$\lambda(x, y, \theta, t) = (x, y)$$

We used a set of 16-discretized values for the heading angle and a maximum value of $T = 1000$ seconds at a resolution of 0.1 seconds for the time dimension. The set of motion primitives used for the 4D states consists of pre-computed kinematically feasible motion sequences as used in a lattice-type planner (Likhachev and Ferguson 2009). The motion primitives used for the 2D states were the eight neighboring states according to the eight-connected 2D grid. Note that the motion primitives for 2D states do not produce feasible paths that can be executed by the robot. The objective of the planner is to find the minimal time path from the start state to goal state. Hence, cost of each edge in the graph is the time taken to execute the respective action multiplied by a constant factor.

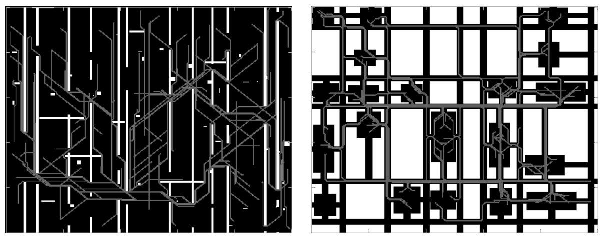

(a) Maze-like environment (b) Indoor environment

Figure 3: Example maps, with paths (gray) of dynamic obstacles shown, used in our experiments. Static obstacles are shown in white and free space in black.

We compared our algorithm to the baseline 4D HCA* planner on several different environment sizes. In small environments with a few hundred cells along each spatial dimension, the baseline planner comes up with the plan quickly, so there is no advantage from our approach. In very large environments with more than 4000 cells along each spatial dimension, the baseline planner runs out of memory to find a solution, while our approach, since it deals with a low-dimensional state-space, was still able to plan successfully. To effectively compare the two approaches at the same level, we chose a moderate environment size of 2500 cells along each spatial dimension and generated 50 maze-like random environments like the one shown on the left in Figure 3. We also generated 50 random indoor environments of the same size like the one shown on the right in Figure 3. These indoor environments are composed of a series of narrow hallways and rooms on a grid placed randomly, while the maze-like environments are composed of a series of walls with small gaps in them to allow the robot to pass through.

In each of these environments, we randomly generate 30 dynamic obstacles. Each dynamic obstacle could come in a large or small size, randomly chosen, and started at a random configuration in the environment. To generate the trajectory for a dynamic obstacle, random goals were chosen and 2D A* is used to generate the paths between the goal points. We chose the start and goal configuration for each dynamic obstacle so that the resulting path is long enough, ensuring that they cover a significant area of the environment. In the indoor environments, the large dynamic obstacles fill the entire width of the hallways, so there is no way to pass them while the narrow dynamic obstacles fill half the width of the hallway, so they can be passed. In the maze-like environments, the dynamic obstacles traverse through the small gaps that the robot tries to pass through, resulting in congestion at such gaps. For each set of environments, we execute

Algorithm	Number of Success	Epsilon	time (secs)		# 4D expands		# 2D expands		path cost	
			mean	std dev	mean	std dev	mean	std dev	mean	std dev
Adaptive	49	1.1	6.4	0.7	3160	1105	10810	9361	39442	4438
4D	7	1.1	99.3	67.7	127393	87024	0	0	37142	5766
Adaptive	50	1.5	20.9	48.5	16688	42804	33276	88880	55342	14668
4D	40	1.5	67.1	75.8	85324	96306	0	0	49150	11568
Adaptive	50	2.0	18.4	39.2	16029	42418	23193	62865	60050	17148
4D	44	2.0	36.5	61.5	45172	76970	0	0	54090	15339

Table 1: Results on 50 indoor environments with 10 dynamic obstacles.

Algorithm	Number of Success	Epsilon	time (secs)		# 4D expands		# 2D expands		path cost	
			mean	std dev	mean	std dev	mean	std dev	mean	std dev
Adaptive	41	1.1	6.7	0.8	3705	1379	12524	9901	40740	2200
4D	5	1.1	91.0	71.2	111165	87228	0	0	38320	6522
Adaptive	40	1.5	11.7	14.0	7318	9285	21454	38559	54690	16811
4D	21	1.5	70.3	86.7	88576	109859	0	0	47566	9916
Adaptive	46	2.0	18.5	26.6	16000	31148	13672	21383	57760	19450
4D	23	2.0	35.8	69.8	43546	86677	0	0	50039	12256

Table 2: Results on 50 indoor environments with 30 dynamic obstacles.

two sets of runs - one with 10 dynamic obstacles in each environment and the other with an additional 20 dynamic obstacles (making it a total of 30 moving obstacles) in each environment.

The underlying search algorithm used in both the planning and tracking phase is weighted A* with the ϵ_{plan} sub-optimality bound. The tunnel width used for the tracking phase, was 10 cells, and the radii of the newly added spheres were 20 cells. For the heuristic used by the weighted A* planners, in our approach and the baseline HCA*, we use a 2D Dijkstra search from the goal state to all the (x, y) cells in the environment ignoring the dynamic obstacles. During the computation of heuristic, static obstacles are inflated by the inscribed circle radius of the robot to preclude paths through areas that are too narrow for the robot to physically traverse.

For each environment, we try three values of ϵ: 1.1, 1.5, and 2 with the adaptive planner using the square root of ϵ for both ϵ_{plan} and ϵ_{track}, thus giving an overall sub-optimality bound of ϵ for the adaptive planner. We use the same set of ϵ values for the baseline planner and compare their performance. For both planners, we enforce a maximum planning time of 5 minutes for all ϵ values.

8 Results

For both sets of environments, we compare the planning time, number of high-dimensional (4D) states expanded, number of low-dimensional (2D) states expanded and resulting path cost, between our approach and the baseline 4D HCA* approach. We also list out the number of cases among the set of 50 environments that our approach could come up with a solution within 5 minutes of planning time and the number of cases the baseline approach could. Note that statistics like mean planning time, number of HD expansions, number of LD expansions and path cost, are computed only on cases where both approaches could come up with a solution within 5 minutes.

Tables 1 and 2 present the results for 50 randomly generated indoor environments with 10 and 30 dynamic obstacles respectively. In these environments, the low-dimensional heuristic used is very often misleading as it cannot account for dynamic obstacles and leads the search into a blocked hallway. For $\epsilon = 1.1$, the planning problem is hard and the baseline could solve only 5 environments with 30 dynamic obstacles (7 in the case of 10 dynamic obstacles). In comparison, our approach could solve 41 of the 50 environments with 30 dynamic obstacles (49 in the case of 10 dynamic obstacles) with a substantially smaller mean planning time. As the ϵ value increases, the planning problem becomes easier and performance of the baseline approach improves. Even in these easier cases, our approach has a comparable performance, if not better than that of baseline. Results across tables 1 and 2 show that our approach performs well even when the number of obstacles increases, whereas the performance of baseline degrades substantially.

The results for the 50 randomly generated maze-like environments with 10 and 30 dynamic obstacles are presented in tables 3 and 4. These environments are characterized by tight turns and potential dynamic obstacle collisions at the gaps in the walls. In most cases, the robot would have to swerve around the obstacle to avoid collision. Hence, the resulting path doesn't deviate significantly from the one suggested by heuristic. From the results we can observe that when the planning problem is difficult (for example, when $\epsilon = 1.1$), our approach could solve a large number of cases (47 and 46 of 50) when compared to the baseline (4 and 2 of 50). But as the ϵ value increases and the planning problem becomes easier, performance of baseline quickly catches up with our approach and in one of the case, outperforms our approach as well ($\epsilon = 2$ in the 30 dynamic obstacles case). But in most runs, our approach performs better than the baseline in mean planning time and the path cost. Results across tables 3 and 4 show that there is not as much decrease in the performance of baseline with increase in number of obstacles, when compared to the indoor environments.

9 Discussion

Interestingly, in indoor environments our approach returns paths with higher costs (but still within the suboptimality bound) when compared to the baseline approach. This is due to the fast low-dimensional 2D planning used in our ap-

Algorithm	Number of Success	Epsilon	time (secs)		# 4D expands		# 2D expands		path cost	
			mean	std dev	mean	std dev	mean	std dev	mean	std dev
Adaptive	47	1.1	7.9	1.4	5105	2177	26665	9660	432425	43683
4D	4	1.1	161.5	59.8	195758	79955	0	0	416650	40272
Adaptive	48	1.5	14.2	11.2	9622	8566	22588	21539	532652	91234
4D	45	1.5	41.3	43.8	51515	56488	0	0	562109	95470
Adaptive	48	2.0	12.6	16.6	11771	13531	25630	34279	537873	89790
4D	48	2.0	18.6	35.8	22485	45048	0	0	622739	104136

Table 3: Results on 50 maze-like environments with 10 dynamic obstacles.

Algorithm	Number of Success	Epsilon	time (secs)		# 4D expands		# 2D expands		path cost	
			mean	std dev	mean	std dev	mean	std dev	mean	std dev
Adaptive	46	1.1	18.2	16.1	12565	12843	50012	12470	441100	91923
4D	2	1.1	192.7	159.0	236515	196442	0	0	424250	81529
Adaptive	48	1.5	25.4	33.8	16558	18703	29679	28739	524451	83024
4D	45	1.5	46.7	46.0	59116	59870	0	0	553686	89470
Adaptive	48	2.0	22.9	36.2	18922	22002	44550	107434	538370	92375
4D	48	2.0	21.3	34.7	25986	43153	0	0	623997	103201

Table 4: Results on 50 maze-like environments with 30 dynamic obstacles.

proach which when a hallway is blocked finds an alternative path through an adjacent hallway, even if it is against heuristic. In contrast, the baseline tries to go through the blocked hallway suggested by the heuristic by wasting time before and entering the hallway once the obstacle comes out. Hence, the path returned by baseline often has low cost.

Generally, in environments where dynamic obstacles do not block the path suggested by heuristic, the baseline approach is fast and often outperforms our approach. This is the case in maze-like environments where the robot has to just swerve around the obstacle to avoid it. Hence, we see a good performance of baseline in such cases. But in cases where the solution required a path significantly different from the one computed by heuristic, baseline performs poorly and our approach outperforms it. This is the case in indoor environments where the entire hallway is blocked by an obstacle and the planner has to find an alternative path which might be against the heuristic. In such environments, as the number of dynamic obstacles increases, the heuristic becomes less informative and performance of baseline degrades. Our approach circumvents this through its iterative nature and fast low-dimensional planning.

10 Conclusion and Future Work

In this work we have presented a new approach to path planning in dynamic environments that doesn't make any assumptions on the robot's motion model, but still achieves significant speedups in planning time over heuristic-based A*. Our algorithm builds on the previously-developed algorithm for path planning with adaptive dimensionality to explicitly decrease the dimensionality of the search space in an adaptive manner. The algorithm plans in full dimensional state-space in regions of the environment where there is a potential dynamic obstacle collision and in low-dimensional state-space elsewhere, thereby obtaining quicker planning times. We have proven that our approach returns feasible, collision-free paths in dynamic environments with bounds on solution cost sub-optimality. As shown in our results, we outperform full-dimensional planning algorithms such as HCA* by a substantial margin in tasks like navigation of non-holonomic robot in dynamic environments.

As a part of future work, we plan to verify performance of the algorithm on a real robot navigating in a realistic environment with dynamic obstacles. We are exploring the possibility of using an incremental planner to reuse the search information from previous iterations to speed up planning in subsequent iterations. Currently, the planning algorithm starts from scratch at the start of each iteration and does not reuse search tree from previous iterations. We are also interested in relaxing our assumption of complete knowledge regarding the trajectories of dynamic obstacles and handle uncertainty in the predicted trajectories within the algorithm.

Acknowledgments

This work was conducted in part through collaborative participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016, and in part by ONR under MURI grant "Reasoning in Reduced Information Spaces" (no. N00014-09-1-1052). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The authors would also like to thank Venkatraman Narayanan and Kalin Gochev for their valuable inputs, and the anonymous reviewers for their suggestions.

References

Bekris, K. E., and Kavraki, L. E. 2007. Greedy but safe replanning under kinodynamic constraints. In *IEEE International Conference on Robotics and Automation, 2007*, 704–710. IEEE.

Brock, O., and Khatib, O. 1999. High-speed navigation using the global dynamic window approach. In *IEEE International Conference on Robotics and Automation, 1999*, volume 1, 341–346. IEEE.

Fox, D.; Burgard, W.; and Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* 4(1):23–33.

Gochev, K.; Cohen, B.; Butzke, J.; Safonova, A.; and Likhachev, M. 2011. Path planning with adaptive dimensionality. In *Fourth annual symposium on combinatorial search*.

Gochev, K.; Safonova, A.; and Likhachev, M. 2012. Planning with adaptive dimensionality for mobile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA), 2012*, 2944–2951. IEEE.

Gochev, K.; Safonova, A.; and Likhachev, M. 2013. Incremental planning with adaptive dimensionality. In *ICAPS*.

Knepper, R. A., and Kelly, A. 2006. High performance state lattice planning using heuristic look-up tables. In *IROS*, 3375–3380.

Kushleyev, A., and Likhachev, M. 2009. Time-bounded lattice for efficient planning in dynamic environments. In *IEEE International Conference on Robotics and Automation, 2009*, 1662–1668. IEEE.

Likhachev, M., and Ferguson, D. 2009. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research* 28(8):933–945.

Narayanan, V.; Phillips, M.; and Likhachev, M. 2012. Anytime safe interval path planning for dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012*, 4708–4715. IEEE.

Petereit, J.; Emter, T.; and Frey, C. W. 2013. Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality. In *Proceedings of the IFAC Intelligent Autonomous Vehicles Symposium*, 546–563.

Petereit, J.; Emter, T.; and Frey, C. W. 2014. Combined trajectory generation and path planning for mobile robots using lattices with hybrid dimensionality. In *Robot Intelligence Technology and Applications 2*. Springer. 145–157.

Petti, S., and Fraichard, T. 2005. Safe motion planning in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.(IROS 2005).*, 2210–2215. IEEE.

Philippsen, R., and Siegwart, R. 2003. Smooth and efficient obstacle avoidance for a tour guide robot. In *None*, number LSA-CONF-2003-018.

Phillips, M., and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA), 2011*, 5628–5635. IEEE.

Rufli, M.; Ferguson, D.; and Siegwart, R. 2009. Smooth path planning in constrained environments. In *IEEE International Conference on Robotics and Automation, 2009.*, 3780–3785. IEEE.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

Van Den Berg, J.; Ferguson, D.; and Kuffner, J. 2006. Anytime path planning and replanning in dynamic environments. In *IEEE International Conference on Robotics and Automation, 2006.*, 2366–2371. IEEE.

Zhang, H.; Butzke, J.; and Likhachev, M. 2012. Combining global and local planning with guarantees on completeness. In *IEEE International Conference on Robotics and Automation (ICRA), 2012*, 4500–4506. IEEE.