Path Planning with Adaptive Dimensionality

Kalin Gochev
Univ. of Pennsylvania
kgochev@seas.upenn.edu

Benjamin Cohen
Univ. of Pennsylvania
bcohen@seas.upenn.edu

Jonathan Butzke
Univ. of Pennsylvania
jbutzke@seas.upenn.edu

Alla Safonova
Univ. of Pennsylvania
alla@seas.upenn.edu

Maxim Likhachev
Carnegie Mellon Univ.
maxim@cs.cmu.edu

Abstract

Path planning quickly becomes computationally hard as the dimensionality of the state-space increases. In this paper, we present a planning algorithm intended to speed up path planning for high-dimensional state-spaces such as robotic arms. The idea behind this work is that while planning in a high-dimensional state-space is often necessary to ensure the feasibility of the resulting path, large portions of the path have a lower-dimensional structure. Based on this observation, our algorithm iteratively constructs a state-space of an adaptive dimensionality–a state-space that is high-dimensional only where the higher dimensionality is absolutely necessary for finding a feasible path. This often reduces drastically the size of the state-space, and as a result, the planning time and memory requirements. Analytically, we show that our method is complete and is guaranteed to find a solution if one exists, within a specified suboptimality bound. Experimentally, we apply the approach to 3D vehicle navigation (x, y, heading), and to a 7 DOF robotic arm on the Willow Garage's PR2 robot. The results from our experiments suggest that our method can be substantially faster than some of the state-of-the-art planning algorithms optimized for those tasks.

Keywords: Motion and Path Planning, Planning Algorithms, Heuristic Search

1 Introduction

Path planning is frequently done in high-dimensional state-spaces in order to represent a high degree of freedom robotic system or to account for various kinodynamic constraints of the system. Unfortunately, the high dimensionality of the state-space often leads to a dramatic increase in the time and memory required to find a path. However, while planning in a high-dimensional state-space is often necessary, large portions of the computed paths are still low-dimensional. For example, a 3D (x, y, θ) path for a non-holonomic robot typically contains large portions that are straight-line segments and do not therefore require 3-dimensional planning. Sections of the path that include turning do require full-dimensional planning. Similarly, planning for manipulation can often be reduced to 3D planning for an end-effector and then just running an inverse kinematics solver to find the full-dimensional path that corresponds to the found end-effector path. At the same time, there are relatively infrequent situations where the planner does need to consider the full configuration of the arm in trying to figure out the feasibility of the end-effector path.

In this paper, we present an algorithm that exploits this observation. It iteratively constructs a state-space that is low-dimensional everywhere except for the areas where low-dimensional planning fails. This results in substantial speedups and lower memory requirements. On the theoretical side, we show that the method is complete with respect to the state-space discretization and can provably guarantee to find a solution, if one exists, within a given suboptimality bound. On the experimental side, we apply our algorithm to a 3D (x, y, θ) vehicle navigation problem, planning adaptively in 3D/2D, and also to a 7 DOF robot arm on the Willow Garage's PR2 robot, planning adaptively in 7D/3D, where 3D corresponds to 3D (x, y, z) planning for the end effector. In both scenarios, our experiments suggest that our approach can be substantially faster than other methods optimized for these tasks.

2 Related Work

In order to improve planning times, researchers have used a variety of techniques to avoid performing global planning in high-dimensions. Many path planners implement a two layer planning scheme where a low-dimensional global planner provides input to a higher-dimensional local planner. Since these local planners can operate on a small subset of the entire environment, they can afford to include more dimensions while still meeting planning time constraints. The local planners have been implemented using reactive obstacle avoidance planners (Thrun and others 1998) and dynamic windows (Philippsen and Siegwart 2003; Brock and Khatib 1999) to produce feasible paths from an underlying low-dimensional global planner. However these techniques can result in highly suboptimal paths and even paths that are infeasible to follow due to mismatches in the assumptions made by the higher and lower level planners.

Our approach does not split the planning process into two fixed layers but rather mixes the dimensionalities of the planning problem within a single planning process. The tech-

This research was sponsored by ONR grant N00014-09-1-1052 and the Army Research Laboratory Cooperative Agreement Number W911NF-10-2-0016.

nique most similar to ours is the hierarchical planners using homomorphic abstraction which have shown excellent runtime reductions by grouping adjacent states together and pre-computing costs for traversing the set from all possible entry and exit points (Botea, Müller, and Schaeffer 2004). This approach requires significant pre-processing in order to be effective. Other hierarchical planners use different methods of abstraction to make better informed heuristics to guide the search (Bulitko et al. 2007). Our method differs from these in that we change the dimensionality of a state-space where necessary as opposed to combining states to have connecting edges.

Our approach is also somewhat relevant to planners that use very accurate pre-computed heuristic values (Knepper and Kelly 2006). The heuristics are often derived by solving a lower-dimensional problem. As a result, these methods can be viewed as a full-dimensional planning that uses the results of the lower dimensional planning. Unlike our approach however, these methods do not explicitly decrease the dimensionality and, as a result, can run into severe computational problems when the heuristic is incorrect.

3 Planning with Adaptive Dimensionality

3.1 Definitions, Notations, Assumptions

We are assuming that the planning problem is represented by a discretized finite state-space S of dimensionality d, consisting of state vectors $X = (x_1, ..., x_d)$, and a set of transitions $T = \{(X_i, X_j) | X_i, X_j \in S\}$. Each transition (X_i, X_j) corresponds to a feasible transition between the corresponding state vector values and is associated with a cost $c(X_i, X_j)$ which is bounded from below by some positive δ, that is, $c(X_i, X_j) > \delta > 0$. Thus, we have an edge-weighted graph G with a vertex set S and edge set T. The goal of the planner is to find a least-cost path in G from the start state X_S to the goal state X_G. We will use the notation $\pi(X_i, X_j)$ to denote a path in graph G from state X_i to state X_j. We will use $\pi^*(X_i, X_j)$ to denote a least-cost path. The cost of any path $\pi(X_i, X_j)$ is the cumulative costs of the transitions along it and will be notated by $c(\pi(X_i, X_j))$.

Consider two state-spaces—a high-dimensional S^{hd} with dimensionality h, and a low-dimensional S^{ld} with dimensionality l, which is a projection of S^{hd} onto a lower dimensional manifold ($h > l, |S^{hd}| > |S^{ld}|$). We define a many-to-one mapping

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

from the high-dimensional state-space S^{hd} to the low-dimensional state-space S^{ld}. For example, in the case of 3D/2D navigation planning we used the simple mapping $\lambda((x, y, \theta)) = (x, y)$, just dropping the heading information θ.

Each of the two state-spaces may have its own transition set. For example, in the 3D/2D navigation planning scenario we used 8-connected grid transitions for the 2D state-space, and a set of precomputed feasible atomic actions that capture the kinodynamic constraints of the vehicle, called motion primitives, as transitions for the 3D state-space (Figure 1). We require that the costs of the transitions be such that for

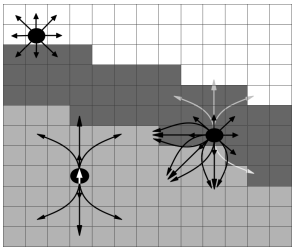


Figure 1: Example state transitions for a 3D/2D state-space–white cells are 2D states (x, y), dark gray cells are 2D states with feasible 3D transitions to 3D states $(x, y,$ heading$)$, and the light gray cells are 3D states. On the upper left is shown a 2D state with all of its feasible transitions (only 2D transitions). The state in the middle right is in the boundary area, so its feasible transitions include all 2D transitions that end in a 2D state and all 3D transitions (from all possible heading values) that end in a 3D state. In light gray are shown some of the disallowed 3D transitions, since they lead to 2D states. In the lower left is a 3D state with all of its 3D transitions (heading indicated by the white arrow).

every pair of states X_i and X_j in S^{hd},

$$c\left(\pi^*\left(X_i, X_j\right)\right) \geq c\left(\pi^*\left(\lambda(X_i), \lambda(X_j)\right)\right) \qquad (1)$$

That is, we require that the cost of a least-cost path between any two states in the high-dimensional state-space to be at least the cost of a least-cost path between their images in the low-dimensional state-space.

We also define the mapping $\lambda^{-1} : S^{ld} \rightarrow (S^{hd})^*$ from the low-dimensional state-space S^{ld} to subsets of the high-dimensional state-space S^{hd}, defined by

$$\lambda^{-1}(X^{ld}) = \{X \in S^{hd} | \lambda(X) = X^{ld}\}$$

Notice that λ^{-1} is a one-to-many mapping.

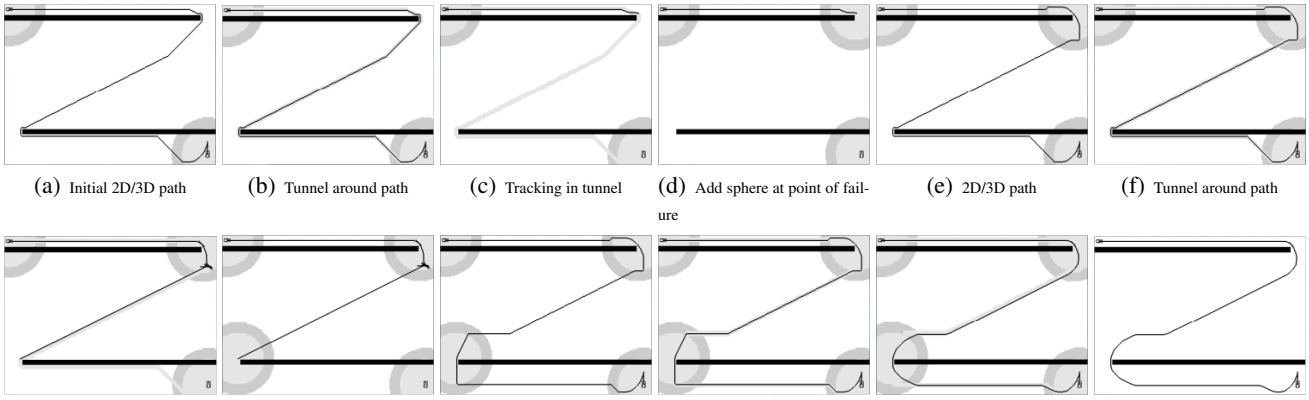
Let G^{hd} and G^{ld} represent the corresponding graphs defined by S^{hd} and S^{ld} and their respective transition sets T^{hd} and T^{ld}.

The idea of our algorithm is to iteratively construct and search an adaptively-dimensional state-space S^{ad}. We discuss the structure and the construction of S^{ad} in the next section.

3.2 Algorithm

Structure of S^{ad}: Recall that the goal of our algorithm was to use the faster low-dimensional planning, except for areas of the environment where high-dimensional planning is necessary to ensure the feasibility of the resulting path. We want our adaptively-dimensional state-space to capture this property—namely, we want to have largely low-dimensional states in S^{ad}, except for the areas where high-dimensional planning needs to be done, represented by high-dimensional states in S^{ad}. To ensure path feasibility in the high-dimensional regions of S^{ad}, we have to use high-dimensional transitions. In the low-dimensional areas we can use simpler low-dimensional transitions. However, recall that the transitions we have in T^{hd} and T^{ld} connect two states of the same dimensionality, which do not allow us to transition from the low-dimensional to the high-dimensional regions. Therefore, we have to construct a transition set T^{ad} that allows for transitions between states of different dimensionalities.

Construction of S^{ad}: Our algorithm iteratively constructs S^{ad}, beginning with the low-dimensional-state space S^{ld} and introducing a set of high-dimensional regions R in it. We will first explain how the high-dimensional regions



(a) Initial 2D/3D path (b) Tunnel around path (c) Tracking in tunnel (d) Add sphere at point of failure (e) 2D/3D path (f) Tunnel around path

(g) Tracking in tunnel (h) Add sphere at point of failure (i) 2D/3D path (j) Tunnel around path (k) Tracking in tunnel (l) Final trajectory

Figure 2: Example of iterative process for simple map. The light gray circles are the states that exist in 3D, while the darker gray outer circles represent the border of 2D states which have valid 3D transitions going into the 3D areas. The black bars are obstacles, white areas are 2D only, and the dark gray lines are the path from the 2D/3D search and the forward simulation.

are being introduced into S^{ad} and connected with the low-dimensional regions. The algorithm that decides when and where to introduce these regions will be explained later.

Once a high-dimensional region r is introduced, the following changes are done to S^{ad}. If a low-dimensional state X_i^{ld} falls inside a high-dimensional region $r \in R$, we replace it with its high-dimensional projection states in $\lambda^{-1}(X_i^{ld})$. Thus, S^{ad} contains both low-dimensional and high-dimensional states. Notice that if a high-dimensional state X^{hd} is in S^{ad}, then its low-dimensional projection $\lambda(X^{hd})$ is not in S^{ad}, and also if $X^{hd} \notin S^{ad}$, then $\lambda(X^{hd}) \in S^{ad}$.

Next we define the transition set T^{ad} for the adaptively-dimensional state-space as follows. 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 $(X_i, X_j^{hd}) \in T^{ad}$. If $X_j^{hd} \notin S^{ad}$, then $(X_i, \lambda(X_j^{hd})) \in T^{ad}$. That is, for high-dimensional states we allow only high-dimensional transitions to other high-dimensional states if they fall inside S^{ad}, or their low-dimensional projections (Fig. 1 lower left).
- 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 $(X_i, X_j^{hd}) \in T^{ad}$. That is, for low-dimensional states we allow low-dimensional transitions if they lead to another low-dimensional state in S^{ad} (Fig. 1 upper left), and high-dimensional transitions from their high-dimensional projections if they lead to a high-dimensional state in S^{ad} (Fig. 1 right).

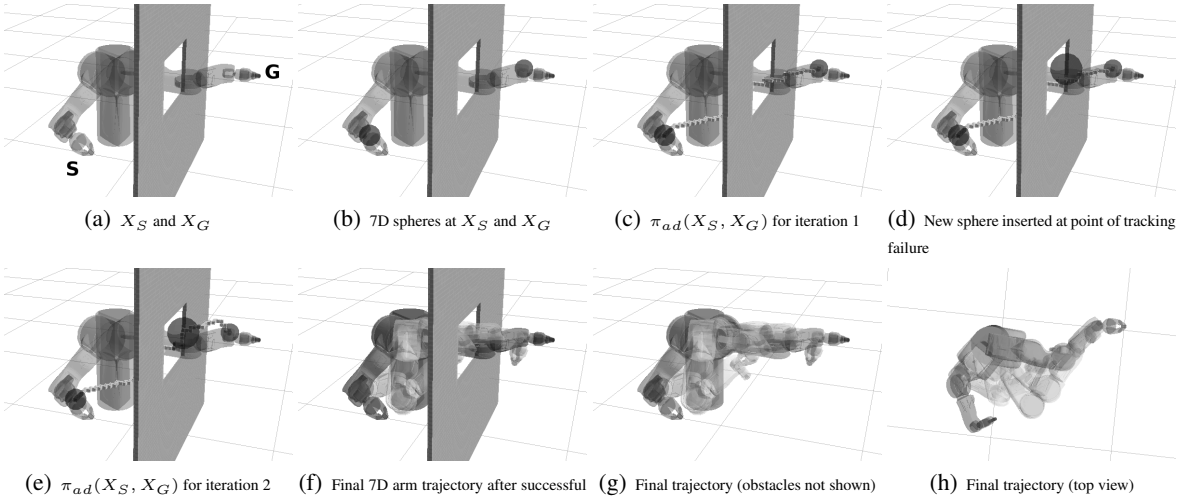
Notice, that the above definition of T^{ad} allows for transitions between states of different dimensionalities. Figure 1 illustrates the set of transitions in the adaptive graph in the case of 3D (x, y, θ) path planning.

The adaptively-dimensional state-space S^{ad} and the transition set T^{ad} give us a graph G^{ad} of adaptive dimensionality. Adding new high-dimensional regions or increasing the sizes of existing regions requires the reconstruction of S^{ad} and T^{ad}, and thus, will produce a new instance of G^{ad}.

We also define a tunnel τ of radius w around an adaptively-dimensional path π_{ad} as follows: τ is a subgraph of G^{hd}, and thus consists of high-dimensional states and transitions. A high dimensional state $X^{hd} \in \tau$ if there exists a state $X_i \in \pi_{ad}$ such that the distance from $\lambda(X^{hd})$ to X_i (or $\lambda(X_i)$ if X_i is high-dimensional) is no larger than w, for some pre-defined distance metric in S^{ld}. We include all transitions (X_j, X_k) from T^{hd} such that both X_j and X_k are in τ.

We continue this section with an intuitive description of our proposed algorithm, in particular the algorithm for deciding when and where to introduce the high-dimensional regions within S^{ad}. Figure 2 provides an illustration of a run of the algorithm for 3D (x, y, θ) path planning, that completed in 3 iterations. Algorithm 1 gives the pseudo code for our algorithm. Each iteration of the algorithm consists of two phases—an adaptive planning phase (Fig. 2(a)) and a path tracking phase (Fig. 2(b) - 2(d)). In the adaptive planning phase, the current instance of the adaptively-dimensional graph G^{ad} is searched for a least-cost path of adaptive dimensionality from start to goal. The tracking phase, then attempts to construct a high-dimensional executable path to match (or track) the adaptive path computed in the adaptive planning phase.

Initially, G^{ad} is the same as G^{ld}, with two high-dimensional regions added around the start and goal states (Algorithm 1, lines 1-3), which are necessary since the start and goal states are high-dimensional. At each iteration, a new instance of G^{ad} is constructed based on the set of high-dimensional regions, and is searched for a least-cost path π_{ad}^* from X_S to X_G. Notice that π_{ad}^* consists of both low-dimensional and high-dimensional states, so it is not an executable path. If no path is found in the adaptive planning phase, then no feasible path exists from start to goal and the algorithm terminates. If an adaptive path π_{ad}^* is found, then the path tracking phase constructs a tunnel τ of radius w around the adaptive path π_{ad}^* (Fig. 2(b)). Then τ is searched for a least-cost path π_τ^* from start to goal (Fig. 2(c)). Note that since τ consists of only high-dimensional states and transitions, π_τ^* is a fully high-dimensional path,



(a) X_S and X_G (b) 7D spheres at X_S and X_G (c) $\pi_{ad}(X_S, X_G)$ for iteration 1 (d) New sphere inserted at point of tracking failure

(e) $\pi_{ad}(X_S, X_G)$ for iteration 2 (f) Final 7D arm trajectory after successful tracking (g) Final trajectory (obstacles not shown) (h) Final trajectory (top view)

Figure 3: Example environment for robotic arm motion planning. A trajectory is computed of how the arm can be maneuvered from the start configuration to reach through the opening to the goal arm configuration in two iterations of our algorithm. 3(c) and 3(e) show the adaptively-dimensional paths computed at each iteration

and thus, it is executable. If no path is found in τ, then a new high-dimensional region is introduced in G^{ad} or the sizes of the existing regions are increased, and the algorithm proceeds to the next iteration. If a path is found in τ, but its cost $c(\pi^*_\tau) > \epsilon_{\text{track}} \cdot c(\pi^*_{ad})$, then a new high-dimensional region is introduced or the sizes of existing high-dimensional regions are increased, and another iteration is started. If $c(\pi^*_\tau) \leq \epsilon_{\text{track}} \cdot c(\pi^*_{ad})$, then the algorithm returns π^*_τ as a feasible path from start to goal and terminates. The returned path is guaranteed to have cost that is no more than ϵ_{track} times the cost of an optimal path in G^{hd}.

Identifying the places where high-dimensional regions need to be introduced is a non-trivial problem in itself. In both of our experiments, the search within the tunnel during the path tracking phase keeps a record of how far along the tunnel states have been expanded. Thus, if the search in τ fails, we are able to reconstruct a path to the point where the search had failed, and we introduce a new high-dimensional region there, as seen in Fig. 2(c),2(d),2(g), and 2(h). Line 17 of algorithm 1 is obscure about how exactly the state X_r where a new high-dimensional region needs to be introduced is being computed. There are a number of approaches that can be taken in identifying such a state. Perhaps the simplest one is to pick a random location along the path where to introduce a new region. A more sophisticated technique, which we implemented, is to approximate the location, where the largest cost discrepancy between π^*_{ad} and π^*_τ is observed. Introducing a new high-dimensional region at that location tends to remedy the cost discrepancy, and generally works well in identifying the regions that require high-dimensional planning. The approach taken in computing X_r does not affect the theoretical properties of the algorithm, such as algorithm termination and suboptimality guarantees.

3.3 Theoretical Properties

The presented algorithm is complete with respect to G^{ad} and provides guarantees on the suboptimality related to the ϵ_{track} constant.

Algorithm 1 Path Planning with Adaptive Dimensionality

```
1:  G^ad = G^ld
2:  AddFullDimRegion(G^ad, λ(X_S))
3:  AddFullDimRegion(G^ad, λ(X_G))
4:  loop
5:      search G^ad for least-cost path π*_ad(X_S, X_G)
6:      if π*_ad(X_S, X_G) is not found then
7:          return no path from X_S to X_G exists
8:      construct a tunnel τ around π*_ad(X_S, X_G)
9:      search τ for least-cost path π*_τ(X_S, X_G)
10:     if π*_τ(X_S, X_G) is not found then
11:         let π(X_S, X_end) be the returned path
12:         if X_end is already within FullDimRegion in G^ad then
13:             GrowFullDimRegion(G^ad, λ(X_end))
14:         else
15:             AddFullDimRegion(G^ad, λ(X_end))
16:     else if c(π*_τ(X_S, X_G)) > ε_track · c(π*_ad(X_S, X_G)) then
17:         identify a state X_r where a new FullDimRegion needs to
            be introduced
18:         if X_r is already within FullDimRegion in G^ad then
19:             GrowFullDimRegion(G^ad, X_r)
20:         else
21:             AddFullDimRegion(G^ad, X_r)
22:     else
23:         return π*_τ(X_S, X_G)
```

Theorem 3.1 *The cost of a least-cost path from X_S to X_G, $\pi^*_{ad}(X_S, X_G)$, in G^{ad} is a lower bound on the cost of a least-cost path from X_S to X_G, $\pi^*_{hd}(X_S, X_G)$, in G^{hd}.*

$$c(\pi^*_{ad}(X_S, X_G)) \leq c(\pi^*_{hd}(X_S, X_G))$$

Proof Consider the projection of the path $c(\pi^*_{hd}(X_S, X_G))$ onto the adaptive dimensionality state-space S^{ad}. In this projection, every state X in $\pi^*_{hd}(X_S, X_G)$ is mapped onto itself if $X \in S^{ad}$ and onto $\lambda(X)$ otherwise. Then according to equation 1, every transition T_i in the projected version of the path $\pi^*_{hd}(X_S, X_G)$ will either be bounded from above by the cost of the corresponding transition in

$\pi^*_{hd}(X_S, X_G)$ if T_i is a low-dimensional transition, or will be exactly equal to the cost of the corresponding transition if T_i is a high-dimensional transition. Consequently, the cost of the projected version of $\pi^*_{hd}(X_S, X_G)$ will be no larger than $c(\pi^*_{hd}(X_S, X_G))$. Furthermore, since $\pi^*_{ad}(X_S, X_G)$ is a least-cost path from X_S to X_G in S^{ad}, its cost is no larger than the cost of any other path including the cost of the projected version of $\pi^*_{hd}(X_S, X_G)$. As a result, $c(\pi^*_{ad}(X_S, X_G)) \leq c(\pi^*_{hd}(X_S, X_G))$.

Theorem 3.2 *If we have a finite state-space, algorithm 1 terminates and at the time of its termination, the cost of the returned path $\pi(X_S, X_G)$ is no more than ϵ_{track} times the cost of an optimal path from state X_S to state X_G in G^{hd}.*

Proof The termination of the algorithm is ensured by the fact that after each iteration we are introducing new high-dimensional states to G^{ad}. Since we have a finite state-space, after finitely many iterations, G^{ad} will become identical to G^{hd}, containing only high-dimensional states. G^{ad} will then be searched for a least-cost path in a finite time. If a path is successfully computed by the adaptive planning phase, it will be fully high-dimensional and the tracking phase will be able to track the computed path exactly, causing the algorithm to terminate. If no path is found in G^{ad}, the algorithm again terminates stating that no feasible path exists from start to goal.

The second statement of Theorem 3.2 follows from Theorem 3.1. By Theorem 3.1, the adaptive planning phase produces an underestimate of the real cost from start to goal, that is $c(\pi^*_{ad}(X_S, X_G)) \leq c(\pi^*_{hd}(X_S, X_G))$. Upon algorithm termination, the tracking phase succeeds in finding a path of cost no more than ϵ_{track} times the cost of the computed adaptive path. Thus, we have $c(\pi_\tau(X_S, X_G)) \leq \epsilon_{\text{track}} \cdot c(\pi^*_{ad}(X_S, X_G)) \leq \epsilon_{\text{track}} \cdot c(\pi^*_{hd}(X_S, X_G))$. Hence, the cost of the tracked path is no larger than ϵ_{track} times the cost of an optimal path from start to goal in G^{hd}.

ϵ-suboptimal graph searches such as weighted-A* are often used by researchers (Likhachev and Ferguson 2008), since they provide the flexibility of quickly finding paths of cost no more than ϵ times the cost of an optimal path. The following result can be proven if we modify algorithm 1 to use such ϵ-suboptimal graph searches:

Theorem 3.3 *If ϵ_{plan}-suboptimal searches are used in lines 5 and 9 of algorithm 1, the cost of the path returned by our algorithm is no larger than $\epsilon_{plan} \cdot \epsilon_{track} \cdot \pi^*_{hd}(X_S, X_G)$.*

Proof If we use an ϵ-suboptimal search in the adaptive planning phase, we know that that the cost of the produced path $c(\pi_{ad})$ is no larger than $\epsilon \cdot c(\pi^*_{ad})$. Then we have $c(\pi_{ad}) \leq \epsilon \cdot c(\pi^*_{ad}) \leq \epsilon \cdot c(\pi^*_{hd})$. Then we know that the tracking phase produced a path π_τ of cost no larger than $\epsilon_{\text{track}} \cdot c(\pi_{ad})$. Hence, we have $c(\pi_\tau) \leq \epsilon_{\text{track}} \cdot c(\pi_{ad}) \leq \epsilon_{\text{track}} \cdot \epsilon \cdot c(\pi^*_{hd})$.

3.4 Algorithm Parameters

Our algorithm has several important parameters that directly affect its execution time:

- tracking suboptimality parameter $\epsilon_{\text{track}} \geq 1$ – affects the number of iterations of the algorithm. Larger values produce more suboptimal final paths with fewer algorithm iterations, trading off path suboptimality for planning time.
- adaptive search suboptimality parameter $\epsilon_{\text{plan}} \geq 1$ – affects the time spent in the adaptive planning phase of each iteration. Larger values produce more suboptimal paths quicker, trading off path suboptimality for planning time.
- size of high-dim. regions – affects the number of iterations and also the time spent in the adaptive planning phase of the algorithm. Larger regions tend to reduce the number of iterations but many unnecessary high-dimensional states may be introduced, which increases adaptive planning time. The parameter generally trades off between number of algorithm iterations and time required per iteration.
- width of the tracking tunnel τ – affects the amount of time taken by the tracking phase of the algorithm and the chances of successful tracking. The parameter trades off the number of iterations for tracking time per iteration.

4 Implementation and Experimental Analysis

The domains we chose to validate our algorithm were robotic path planning for non-holonomic robots done in three dimensions—$(x, y, heading)$, and arm motion planning for 7 DOF robotic arm on the Willow Garage's PR2 robot. In both cases our algorithm implementation kept track of the high-dimensional regions of the environment as spheres: 2D (x, y) circles in the case of 3D path planning, and 3D (x, y, z of the end-effector) spheres in the case of robotic arm motion planning). This allowed us to quickly check if a state falls inside a region or not, and also quickly add new regions and grow the sizes of existing ones.

In both cases the graph G representing the problem was constructed as a lattice-based graph, similar to the approach taken in (Likhachev and Ferguson 2008), except we used constant resolution for all lattices. In lattice-based planning, each state consists of a vertex encoding a state vector and edges corresponding to feasible transitions to other states. The set of edges incident to a state are computed based on a set of pre-computed motion primitives, which are executable by the robot.

4.1 3D Path Planning

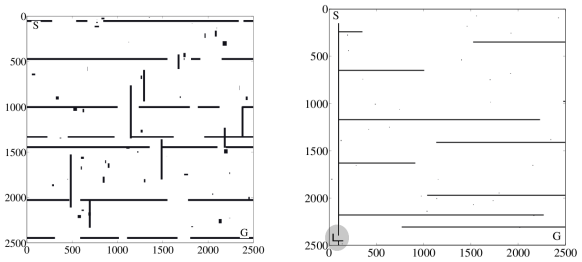
For the 3D planning, we modeled our environment as a planar world and a polygonal robot with three degrees of freedom: x, y, and θ (heading). We used a very simple projection function λ to transform 3D states to 2D states:

$$\lambda_{3D/2D}(x, y, \theta) = (x, y)$$

We used a 16-discretized value for the heading angle, thus, our λ^{-1} mapping was:

$$\lambda^{-1}_{3D/2D}(x, y) = \{(x, y, 0), ..., (x, y, 15)\}$$

The set of motion primitives used for 3D states consisted of long straight, short straight, left and right turn elements for both forward and reverse motion, as can be seen in the lower left corner of figure 1. The motion primitives used for



(a) Typical map used for 3D/2D navigation. (b) Example map for which the 2D Dijkstra heuristic is misleading for the 3D search (the opening on the lower left is not traversable using 3D motion primitives).

Figure 4: Maps of size 2500x2500 cells.

2D states were the eight neighboring states (eight-connected 2D grid), as seen in the upper left of figure 1. It should be noted that the motion primitives for 2D states do not produce feasible paths.

We compared our algorithm to the full 3D planner on several different map sizes. Small maps with few hundred cells in each dimension were quickly solved by the full 3D planner, so little benefit was seen of our algorithm. On maps with 5000 or more cells in both x and y dimensions, the full 3D planner was unable to find a solution due to memory constraints, while our algorithm, having to expand a lot fewer states, was still able to plan successfully.

As a middle ground and to prevent the results from being skewed by the 3D planner having to use the significantly slower hard drive swap space, we randomly generated 50 2500x2500 cell maps typified by figure 4(a) for our test runs.

In all the cases we used a 2D Dijkstra search as the heuristic to help guide a weighted-A* 3D planner towards the goal state. Weighted-A* multiplies the heuristic by an ϵ value to direct the planner along the heuristic path as in (Likhachev and Ferguson 2008). By weighting the heuristic in this manner the resulting path cost is guaranteed to be within ϵ of the optimal cost. In addition, for the 2D portion of our planner and the heuristic generation for the 3D planner, the obstacles on the map were inflated by the inscribed circle radius to preclude the generation of paths through areas too narrow for the robot to physically traverse.

The underlying search algorithm used in both the adaptive planning phase and the tracking phase of our algorithm was also weighted-A* using the ϵ_{plan} parameter suboptimality bound. In addition, the tunnel width we used for the tracking phase was six cells, and the radii of newly added spheres were 20 cells. Since the longest motion primitive was 10 cells long, these parameter values seemed sufficient to allow reasonable range of maneuvers to occur within a sphere and within the tracking tunnel τ.

For each map three values of the suboptimality parameter ϵ were tried: 1.1, 1.5 and 3.0 with the adaptive planner using the square root of ϵ for both ϵ_{plan} and ϵ_{track}, giving an overall suboptimality bound of the adaptive algorithm of ϵ. For both planners a maximum planning time was enforced based on the value of ϵ: $\epsilon = 1.1 : 5$ minutes, $\epsilon = 1.5 : 4$ minutes, $\epsilon = 3.0 : 3$ minutes.

Figure 5: Trajectory from Fig. 3 being executed by an actual PR2 robot

4.2 Robotic Arm Motion Planning

In the case of the robotic arm motion planning, our goal was to use a 7D/3D adaptive planning, where 3D states represented the arm's end-effector position, and 7D states represented the full arm configuration. Generally, the full arm configuration on the PR2 robot is given by its seven joint angles (shoulder pan, shoulder lift, shoulder roll, elbow flex, forearm roll, wrist flex, wrist roll). Constructing a λ mapping reducing full joint angle configuration to end effector position presented several challenges—namely discretization of the joint angle space could not be easily matched to a discretization of the end-effector position space, and λ and λ^{-1} would have needed to involve expensive FK and IK computations. Instead, we decided to transform the standard 7D robot arm configuration representation to one described in (Tolani, Goswami, and Badler 2000), which converts joint angles representations of a 7 DOF arm to 7 DOF representations consisting of the following values: (end-effector x position, end-effector y position, end-effector z position, end-effector roll, end-effector pitch, end-effector yaw, swivel angle). We are going to adopt the following short-hand notation for describing such states: ($ee_{position}$, $ee_{orientation}$, $swivel$), where $ee_{position}$ and $ee_{orientation}$ consist of 3 values each. For more details on the representation, consult (Tolani, Goswami, and Badler 2000). This alternative representation of the full arm configuration did not change the dimensionality of the high-dimensional state-space, but provided clean and easy λ and λ^{-1} mappings without any discretization inconsistencies.

$$\lambda_{7D/3D}(ee_{position}, ee_{orientation}, swivel) = (ee_{position})$$

$$\lambda^{-1}_{7D/3D}(ee_{position}) = \{(ee_{position}, ee_{orientation}, swivel)| \text{ for all feasible values of } swivel \text{ and } ee_{orientation}\}$$

We used very simple motion primitives for the 7D arm motion planning—namely we allow +/-1 change in each of the seven state-vector values. This produces 14 possible transitions for 7D states and 6 possible transitions for 3D states. Due to the simplicity of the motion primitives, the resulting arm trajectory is not very smooth, but experimenting with a more complex set of motion primitives is one of our future work goals.

We chose a 2 cm. 3D grid resolution for the end-effector position, and 16-discretized values for the four angles. This produced a 3D grid of 75x75x75, or roughly 420,000 low-dimensional states, centered at the shoulder joint. In each cell of the grid we have $16^4 \sim 65,000$ possible high-dimensional states, giving us a total of about 28 billion states in the high-dimensional state-space.

We ran both the adaptive dimensionality planning algorithm and the full 7D planning algorithm on 35 environments and compared the results. Environments ranged in

degree of difficulty—some required very simple motions to navigate from start to goal, while others were more cluttered and required a set of complex maneuvers to navigate around the obstacles. Some of the types of environments we used included various table tops, bookshelves, and random cuboid obstacles. Both the adaptively-dimensional and the 7D algorithm utilized a 3D Dijkstra heuristic to guide the planners to the position constraint. We treated the end-effector as a point robot of radius equal to the radius of the largest link of the arm. More sophisticated collision checking and enforcing of joint limits were done on high-dimensional states.

We observed that new sphere radius parameter value of about 10cm. allows sufficient arm maneuvering. Also tunnel radius of 10-20cm. provides a good balance between the success rate of the tracking phase and the time needed for tracking a path. Since we have a large number of high-dimensional states, we imposed time limits on both the adaptive planning phase and the tracking phase. The time limit we used for the adaptive planning phase was 120 seconds. If the limit was reached the adaptive planning failed and the algorithm terminated, reporting that no path from start to goal could be found in the given time limit. Due to the number of states inside the tunnel τ even with a small radius, the tracking search might take a long time to find a path through the tunnel or fail. Since we require the tracking to fail before we begin a new iteration, it becomes impractical to wait long for tracking to fail before starting a new iteration. Thus, we limit the time for the tracking phase, allowing us to proceed to the next itration more quickly. The time limit on the tracking phase we used was 20 seconds.

4.3 Results

For both the 3D path planning and the 7D motion planning on robotic arm experiments, we compared the total number of states expanded, number of high-dimensional states expanded, final path cost, and execution time of the adaptively-dimensional planner compared to the high-dimensional planner, for each of the maps tested. Our results are summarized in table 1 for 3D vehicle navigation and table 2 for the robotic arm motion planning.

In the case of 3D path planning, while the average time for the adaptively-dimensional planner was significantly shorter than the average time for the 3D planner it is interesting to note that the 3D planner was actually faster on 54 out of 100 runs. When the map was benign, the 2D Djikstra heuristic allowed the 3D planner to expand very few states, particularly at higher ϵ_{plan} values. However, two particular cases led to very long 3D plan times: the case of a map with no solution and the case of a map where the solution required a route very different from the one computed by the heuristic. Of the 18 runs where neither algorithm was able to find a solution in the allowed time the adaptively-dimensional planner recognized no solution was available in an average of 12 seconds with a maximum of 25 seconds. On the other hand, the 3D planner in all but two cases ran out of allowable execution times (determined the two cases after 177 and 175 seconds for $\epsilon = 1.5$ and $\epsilon = 3.0$ respectively).

The second case where the adaptive-dimensionality planner performed significantly better than the 3D planner is the set of maps where the heuristic for the 3D planner fails to find a good route. An example of this type of map is shown in figure 4(b). A significantly shorter path exists from start to goal going through the narrow opening depicted in the lower left. Even after inflating the obstacles, the 2D planner is capable of finding a route through the narrow passage. However, this path is not executable using the 3D motion primitives. The 3D planner cannot make use of this information and update its heuristic due to its non-iterative nature. The adaptively-dimensional algorithm initially plans a 2D path through the short cut, but after attempting to track this path, finds that it cannot negotiate the tight turn and places a sphere at that location. During the next iteration while expanding the 3D states in the sphere the adaptively-dimensional planner determines that no path through the sphere exists and reverts back to the 2D planner to explore other alternative routes. By using the lower-dimensional search to find the alternate route, this search can be performed significantly quicker than the full 3D search.

In the case of 7D motion planning on a robotic arm, we noticed results similar to those obtained in the 3D path planning experiments. For simple environments where the 3D Dijkstra heuristic provides good guidance to the goal and for high ϵ_{plan} values, 7D planning is able to quickly identify a path from start to goal satisfying the suboptimality constraint, without having to expand many states. However, in cases of complex environments, where the heuristic fails to provide good guidance to the goal, or for lower suboptimality bounds the adaptively-dimensional planner performs significantly faster. As seen in table 2, adaptive planning is able to achieve about two times speedup on the average over seven-dimensional planning for suboptimality bound of 5.0, and about ten times speedup for suboptimality bound of 2.0. We ran our algorithm with several sets of parameter values. It is interesting to note that increasing the tracking tunnel radius by a factor of 2 results in about 4 times increase in the average number of 7D states expanded during tracking, and thus, about 4 times increase in the average planning time (19.59s). On the other hand, decreasing the tracking tunnel radius by a factor of 2 results in increased number of algorithm iterations on some of the more cluttered environments, slightly increasing the average planning time (7.66s).

4.4 Comparison with Sampling-Based Planners

Algorithm	End-effector distance between a pair of trajectories		Elbow distance between a pair of trajectories	
	Avg.	Max.	Avg.	Max.
RRT (smoothed)	8.2 cm	27.5 cm	6.6 cm	18.0 cm
RRT (not smoothed)	9.7 cm	28.8 cm	6.5 cm	17.9 cm
adaptive	2.5 cm	7.7 cm	2.2 cm	7.9 cm

Table 3: Trajectory consistency comparison between our planner and an RRT planner in the 7DOF robotic arm setting.

We also compared our adaptively-dimensional planner with a sampling-based planner—RRT (LaValle and Kuffner 2001; Kuffner and LaValle 2000; Kavraki et al. 1996)—in the 7DOF robot-arm setting. The advantages of our algorithm over sampling-based planners are deterministic bounds on suboptimality guarantees, consistency in the solutions of similar problems, and applicability to any (including discrete) planning problems that can be represented as

Algorithm	Suboptimality Bound	Time (secs)		# 3D Expands (in thousands)		# 2D Expands (in thousands)		Total Expands (in thousands)		Path Cost	
		mean	std dev	mean	std dev	mean	std dev	mean	std dev	mean	std dev
3D	1.1	142.57	60.24	5218	2177	n/a		5218	2177	58763	9610
adaptive	1.1	184.99	112.93	4448	2884	2434	1793	6957	3946	59202	9856
3D	1.5	83.74	104.94	2813	3533	n/a		2813	3533	68360	11946
adaptive	1.5	25.78	48.96	648	1665	826	1332	1476	2541	66630	13400
3D	3.0	59.99	79.16	2252	3064	n/a		2252	3064	79707	13463
adaptive	3.0	15.21	35.80	396	1319	656	1145	1053	1903	71358	13372

Table 1: Testing results on randomly generated maps for 3D path planning on non-holonomic robot

Algorithm	Suboptimality Bound	Time (secs)		# Iterations		# 7D Expands		# 3D Expands		Total Expands		Path Cost		Successful Plans
		mean	std dev	mean	max	mean	std dev	mean	std dev	mean	std dev	mean	std dev	
7D	2.0	147.88	59.93	n/a		769743	1103939	n/a		769743	1103939	63417	18088	12 of 35
adaptive	2.0	14.42	41.95	1.31	6.0	47419	151391	33219	189870	79689	244112	72656	17000	33 of 35
7D	5.0	10.63	15.66	n/a		46529	65586	n/a		46529	65586	73344	19092	31 of 35
adaptive	5.0	5.23	10.45	1.06	2.0	23877	45427	113	40	23986	45439	75400	18839	34 of 35

Table 2: Testing results on 35 environments for 7D motion planning on robotic arm

a graph and have corresponding low-dimensional graph representations. Although our algorithm could not match the speed of RRT, the consistency of our planner was significantly better—it produced very similar trajectories for similar start/goal configurations within an environment.

We used the following experimental setup for measuring the consistency of the planners. We picked a random table-top environment in which the goal is to maneuver the robotic arm from under to over a table-top. We created 10 scenarios with similar (but not the same) start and goal configurations in that environment. We ran both our planner and the RRT planner on these scenarios. To measure the consistency between a pair of arm trajectories produced by a planner, we measured the average and maximum distances between end-effector positions along the trajectories and also the average and maximum distances between elbow positions along the trajectories. We calculated the consistency between all (45) pairs of the 10 trajectories produced by our planner and compared it with the consistency between all (45) pairs of the 10 RRT trajectories (we compared with both RRT with post-smoothing and RRT without smoothing; smoothing operations included shortcutting and quintic spline smoothing). Table 3 shows the maximum and average end-effector and elbow distances averaged over the 45 pairwise comparisons of the 10 trajectories for each planner.

5 Conclusion

While many path planning problems are seemingly high-dimensional, they are often low-dimensional in most of the state-space. In this paper, we have presented an algorithm that tries to exploit this observation and constructs a state-space of adaptive dimensionality: high dimensionality is introduced only where it is necessary. This results in a significant speedup over the full-dimensional planning alternatives without sacrificing the guarantees on completeness and sub-optimality.

References

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near Optimal Hierarchical Path-Finding. *Journal of Game Development* 1(1):7–28.

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

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 30:51 – 100.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Knepper, R., and Kelly, A. 2006. High performance state lattice planning using heuristic look-up tables. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 3375–3380.

Kuffner, J., and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 995–1001.

LaValle, S., and Kuffner, J. 2001. Rapidly-exploring random trees progress and prospects. *Algorithmic and Computational Robotics New Directions* 293 – 308.

Likhachev, M., and Ferguson, D. 2008. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Proceedings of Robotics: Science and Systems (RSS)*.

Philippsen, R., and Siegwart, R. 2003. Smooth and efficient obstacle avoidance for a tour guide robot. In *ICRA*, 446–451.

Thrun, S., et al. 1998. Map learning and high-speed navigation in RHINO. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *AI-based Mobile Robots: Case Studies of Successful Robot Systems*. Cambridge, MA: MIT Press.

Tolani, D.; Goswami, A.; and Badler, N. 2000. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models* 62:353 – 388.