Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Incremental Planning with Adaptive Dimensionality

Kalin Gochev
University of Pennsylvania

Alla Safonova
University of Pennsylvania

Maxim Likhachev
Carnegie Mellon University

Abstract

Path planning is often a high-dimensional computationally-expensive planning problem as it requires reasoning about the kinodynamic constraints of the robot and collisions of the robot with the environment. However, large regions of the environment are typically benign enough that a much faster low-dimensional planning combined with a local path following controller suffice. Planning with Adaptive Dimensionality that was recently developed makes use of this observation and iteratively constructs and searches a state-space consisting of mainly low-dimensional states. It only introduces regions of high-dimensional states into the state-space where they are necessary to ensure completeness and bounds on sub-optimality. However, due to its iterative nature, the approach relies on running a series of weighted A^* searches. In this paper, we introduce and apply to Planning with Adaptive Dimensionality a simple but very effective incremental version of weighted A^* that reuses its previously generated search tree if available. On the theoretical side, the new algorithm preserves guarantees on completeness and bounds on sub-optimality. On the experimental side, it speeds up 3D (x,y,heading) path planning with a full-body collision checking by up to a factor of 5. Our results also show that it tends to be much faster than applying alternative incremental graph search techniques such as D^* to Planning with Adaptive Dimensionality.

Keywords: Path Planning, Planning Algorithms, Heuristic Search, Incremental Graph Search

Introduction

Path planning is frequently done in high-dimensional state-spaces in order to represent a high degree of freedom robotic system and to account for the system's various kinodynamic constraints and collisions with the environment. Unfortunately, the high dimensionality of the state-space makes the problem much more computationally expensive. However, while planning in a high-dimensional state-space is often necessary, large portions of the environment are typically benign enough that a much faster low-dimensional planning, combined with a local path following controller suffice. For example, a path for a 3-DoF (x,y,$heading$) nonholonomic robot typically contains large portions that are straight-line segments and therefore do not necessarily require 3-dimensional planning. On the other hand, sections of the path that include turning or involve moving through cluttered spaces do require full-dimensional planning, since the turning radius of the vehicle must be taken into account, and also, exact collision checking must be performed on the full configuration of the robot to ensure a collision-free path. The algorithm for Planning with Adaptive Dimensionality (Gochev et al. 2011; Gochev, Safonova, and Likhachev 2012) that was recently developed makes use of this observation and iteratively constructs and searches a state-space consisting of mainly low-dimensional states. It only introduces regions of high-dimensional states into the state-space where they are necessary to ensure completeness and bounds on sub-optimality. The algorithm has been shown to provide strong theoretical guarantees, such as completeness with respect to the underlying graph encoding the search problem and bounds on solution cost sub-optimality.

However, due to its iterative nature, the algorithm for Planning with Adaptive Dimensionality relies on running a series of weighted A^* searches. In this paper, we introduce and apply to Planning with Adaptive Dimensionality a simple but very effective incremental version of weighted A^* that reuses the valid portion of its previously generated search tree if available. On the theoretical side, the new algorithm preserves guarantees on completeness and bounds on sub-optimality. On the experimental side, we apply the algorithm in the context of 3-DoF (x,y,$heading$) path planning for Willow Garage's PR2 robot, performing full-body collision checking. Our results suggest that the algorithm improves planning times by up to a factor of 5 over the original algorithm for Planning with Adaptive Dimensionality. We also observe that the the simple incremental weighted A^* tends to work better in the context of Planning with Adaptive Dimensionality than alternative incremental graph search techniques such as D^*.

Related Work

In this paper we present an algorithm for performing incremental weighted A^* search and apply it to Planning with

This research was sponsored by ONR grant N00014-09-1-1052, DARPA CSSG program D11AP00275 and the Army Research Laboratory Cooperative Agreement Number W911NF-10-2-0016.

Adaptive Dimensionality. As such, the related work can be split it two groups—work relating to Planning with Adaptive Dimensionality and work relating to incremental heuristic search.

Researchers have used a variety of techniques to avoid performing high-dimensional global planning in order to improve planning times. Often, planners implement a two layer planning scheme, where a low-dimensional global planner provides input to a higher-dimensional local planner, which operates only on a small local region of the environment. 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 sub-optimal paths and even paths that are infeasible due to mismatches in the assumptions made by the higher and lower level planners.

Rather than splitting the planning into two fixed layers, Planning with Adaptive Dimensionality mixes the dimensionalities of the planning problem within a single planning process. This is similar to the hierarchical planners, which use different methods of state abstraction to make better informed heuristics to guide the search (Bulitko et al. 2007).

Researchers have also developed various methods for performing incremental heuristic searches, based on the observation that information computed during previous search queries can be used to perform the current search faster. Generally, incremental heuristic search algorithms fall into three categories.

The first class of algorithms, such as Incremental A^* (Koenig and Likhachev 2002a), D^* (Stentz 1995), and D^*-Lite (Koenig and Likhachev 2002b), aim to identify and repair inconsistencies in a previously-generated search tree. These approaches are very general and don't make limiting assumptions about the structure or behavior of the underlying graph. They also demonstrate excellent performance by repairing search tree inconsistencies that are relevant to the current search task.

The second class of algorithms, such as Fringe-Saving A^* (Sun, Yeoh, and Koenig 2009) and Differential A^* (Trovato and Dorst 2002), also try to re-use a previously-generated search tree, but rather than attempting to repair it, these approaches aim to identify the largest valid portion of the search tree, so that it does not contain any modified states, and resume searching from there. These approaches tend not to be general and to make limiting assumptions. The Fringe-Saving A^*, for example, only works on 2D grids with unit cost transitions between neighboring cells. However, the assumptions made by these algorithms allow them to perform very well in scenarios that meet these assumptions. The algorithm presented in this work falls into this class of incremental heuristic search algorithms.

The third class of incremental heuristic search algorithms, such as Generalized Adaptive A^* (Sun, Koenig, and Yeoh 2008), aim to compute more accurate heuristic values by using information from previous searches. As the heuristic becomes more informative, search tasks are performed faster.

The approach taken in (Holte et al. 1996) is similar to ours in that it combines techniques of abstraction and incremental planning.

Problem Definition

The focus of this work is on path planning for robotic systems, which usually operate in continuous state-spaces. In order to be able to represent the path planning problem in a graph-theoretical context, we discretize and bound the continuous state-space in which the system operates, to obtain a discretized finite state-space S of dimensionality d. d is the number of degrees of freedom considered in the path planning problem. Every state X of the system can then be expressed as a state-vector of size d. We also assume a set of transitions $T = \{(X_i, X_j) | X_i, X_j \in S\}$. Each pair $(X_i, X_j) \in T$ corresponds to a feasible for the robotic system transition between the corresponding state vector values. Each transition is associated with a positive cost $c(X_i, X_j)$. The state-space S and the transition set T define an edge-weighted graph $G = (S, T)$ with a vertex set S and edge set T. We will use the notation $\pi_G(X_i, X_j)$ to denote a path from state X_i to state X_j in G, and its cost will be denoted by $c(\pi_G(X_i, X_j))$. We will use $\pi^*_G(X_i, X_j)$ to denote a least-cost path and $\pi^\epsilon_G(X_i, X_j)$ for $\epsilon \geq 1$ to denote a path of bounded cost sub-optimality: $c(\pi^\epsilon_G(X_i, X_j)) \leq \epsilon \cdot c(\pi^*_G(X_i, X_j))$. The goal of the planner is to find a least-cost path in G from a given start state X_S to a goal state X_G. Alternatively, given a desired sub-optimality bound $\epsilon \geq 1$, the goal of the planner is to find a path $\pi^\epsilon_G(X_S, X_G)$.

Planning with Adaptive Dimensionality

In this section we will provide a brief overview of the algorithm for Planning with Adaptive Dimensionality. For a more detailed explanation of the algorithm, we refer the reader to the original work on Planning with Adaptive Dimensionality (Gochev et al. 2011; Gochev, Safonova, and Likhachev 2012).

Overview

The algorithm for Planning with Adaptive Dimensionality builds on the fact that many high-dimensional path planning problems have lower-dimensional projections that represent the problem very well in most areas. For example, path planning for a non-holonomic vehicle needs to consider the planar position of the vehicle (x,y), but also the heading angle to ensure that system constraints, such as minimum turning radius, are obeyed. However, a two-dimensional representation of the problem, only considering the planar position of the vehicle (x,y), can work well in many areas of the state-space.

Thus, the algorithm for Planning with Adaptive Dimensionality considers two graphs as defined by their corresponding state-spaces and transition sets—a high-dimensional $G^{hd} = (S^{hd}, T^{hd})$ with dimensionality h, and a low-dimensional $G^{ld} = (S^{ld}, T^{ld})$ with dimensionality l. S^{ld} is a projection of S^{hd} onto a lower dimensional manifold ($h > l, |S^{hd}| > |S^{ld}|$) through a projection function

λ.

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

The projection function λ^{-1} maps low-dimensional states to their high-dimensional pre-images:

$$\lambda^{-1} : S^{ld} \rightarrow \mathcal{P}(S^{hd})$$

and is defined as

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

where $\mathcal{P}(S^{hd})$ denotes the power set of S^{hd}.

Each of the two state-spaces may have its own transition set. However, in order to provide path cost sub-optimality guarantees, the algorithm requires that the costs of the transitions be such that for every pair of states X_i and X_j in S^{hd},

$$c\left(\pi^*_{G^{hd}}(X_i, X_j)\right) \geq c\left(\pi^*_{G^{sd}}(\lambda(X_i), \lambda(X_j))\right) \qquad (1)$$

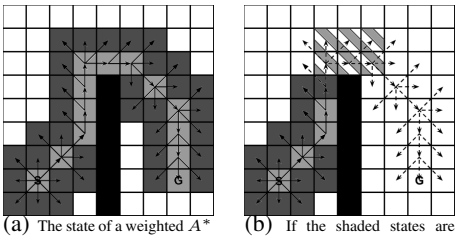
In other words, it is required that the cost of a least-cost path between any two states in the high-dimensional state-space to be at least the cost of a least-cost path between their images in the low-dimensional state-space.

Algorithm

The algorithm for Planning with Adaptive Dimensionality iteratively constructs and searches a graph $G^{ad} = (S^{ad}, T^{ad})$ consisting mainly of low-dimensional states and transitions. The algorithm only introduces regions of high-dimensional states and transitions into the graph where it is necessary in order to ensure the feasibility of the resulting path and maintain path cost sub-optimality guarantees. Each iteration of the algorithm consists of two phases: planning phase and tracking phase.

In the planning phase, the current instance of G^{ad} is searched for a path $\pi^{\epsilon_{plan}}_{G^{ad}}(X_S, X_G)$. Any graph search algorithm that provides a bound on path cost sub-optimality can be used to compute $\pi^{\epsilon_{plan}}_{G^{ad}}$. The original implementation of the algorithm for Planning with Adaptive Dimensionality used weighted A^* algorithm (Gochev et al. 2011; Gochev, Safonova, and Likhachev 2012).

In the tracking phase, a high-dimensional tunnel τ (a subgraph of G^{hd}) is constructed around the path found in the planning phase. Then τ is searched for a path $\pi_\tau(X_S, X_G)$. If $c(\pi_\tau) \leq \epsilon_{track} \cdot c(\pi^{\epsilon_{plan}}_{G^{ad}})$, then π_τ is returned as the path computed by the algorithm. If no path through τ is found or $c(\pi_\tau)$ does not satisfy the above constraint, the algorithm identifies locations in G^{ad}, where the search through τ got stuck or where large cost discrepancies between $\pi^{\epsilon_{plan}}_{G^{ad}}$ and π_τ are observed. The algorithm then introduces new high-dimensional regions in G^{ad} centered at the identified locations. For more details on how the locations of new high-dimensional regions are computed and how high-dimensional regions are introduced in G^{ad}, please refer to (Gochev et al. 2011; Gochev, Safonova, and Likhachev 2012). The algorithm then proceeds to the next iteration.



(a) The state of a weighted A^* search at the end of a successful search episode

(b) If the shaded states are modified, a part of the back-pointer tree remains valid.

Figure 1: Simple 8-connected grid weighted A^* example (assuming a perfect heuristic for simplicity). Light gray: $CLOSED$ list (expanded states), dark gray: $OPEN$ list, striped: modified states, black: obstacles/invalid states, solid arrows: valid back-pointer tree, dashed arrows: invalid back-pointer tree.

Theoretical Properties

If the high-dimensional state-space S^{hd} is finite, the algorithm for Planning with Adaptive Dimensionality is complete with respect to the underlying graph G^{hd} encoding the search problem and is guaranteed to terminate. If a path π is found by the algorithm, then π satisfies

$$c(\pi) \leq \epsilon_{plan} \cdot \epsilon_{track} \cdot \pi^*_{G^{hd}}(X_S, X_G)$$

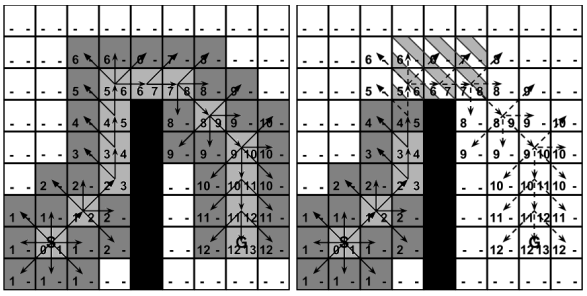
In other words, the cost of a path returned by the algorithm is bounded by $\epsilon_{plan} \cdot \epsilon_{track}$ times the cost of an optimal path from X_S to X_G in the high-dimensional graph G^{hd}. These theoretical properties are proven in (Gochev et al. 2011).

Tree-Restoring Weighted A^* Search

In this section we describe a technique for performing weighted A^* graph search in an incremental fashion and how it is beneficial in the context of Planning with Adaptive Dimensionality.

Motivation

The algorithm for Planning with Adaptive Dimensionality performs multiple iterations of searches of the graph G^{ad}. In the original implementation of the algorithm (Gochev et al. 2011; Gochev, Safonova, and Likhachev 2012) the planning phase of each iteration performs a weighted A^* search of G^{ad} from scratch. However, the structure of G^{ad} only changes inside the high-dimensional regions that have been introduced by the previous algorithm iteration. Therefore, large portions of G^{ad} remain the same between subsequent iterations. Starting a new weighted A^* search with each iteration leads to many redundant expansions of unmodified states, while portions of the search tree constructed by the previous weighted A^* search episode need not be recomputed and can be re-used (Fig. 1). Moreover, as the algorithm for Planning with Adaptive Dimensionality progresses, new high-dimensional regions tend to be introduced closer and closer to the goal, which means that larger and larger portions of the search trees generated during the previous iterations remain unmodified and need not be recomputed. Thus, the performance of the algorithm for Planning with Adaptive Dimensionality could be improved significantly by using an incremental weighted A^* search. An important property of Planning with Adaptive Dimensionality is that upon introducing a new high-dimensional region



(a) Tree-restoring A^* search showing the creation time (bottom left) and expansion step 5. Restoring the weighted A^* search time (bottom right) of each state. A dash indicates ∞.

(b) The first modified state is generated at ***state*** at step 4 produces a valid ***state***.

Figure 2: Simple 8-connected grid tree-restoring weighted A^* example (assuming a perfect heuristic for simplicity). Light gray: $CLOSED$ list (expanded states), dark gray: $OPEN$ list, striped: modified states, black: obstacles/invalid states, solid arrows: valid back-pointer tree, dashed arrows: invalid back-pointer tree.

into G^{ad}, costs of edges cannot decrease as ensured by Eq. 1. The tree-restoring weighted A^* search algorithm makes use of this property.

Algorithm

The ***state*** of a weighted A^* search can be defined by the $OPEN$ list, the $CLOSED$ list, the $g-values$ of all states, and the back-pointer tree. Note the distinction between a ***state*** of a search and a state in the graph; we will use "***state***" when referring to a state of a search. The idea of our approach to incremental weighted A^* planning is to keep track of the ***state*** of the search, so that when the graph structure is modified, we can restore a valid previous search ***state*** and resume searching from there.

We call a ***state*** of a weighted A^* search valid with respect to a set of modified states, if the $OPEN$ and $CLOSED$ lists, and the back-pointer tree do not contain any of the modified states and the g-values of all states are correct with respect to the back-pointer tree.

At any one time during a weighted A^* search, each state falls in exactly one of the following categories:

- $unseen$ - the state has not yet been encountered during the search; its g-value is infinite; the state is not in the back-pointer tree.
- $inOPEN$ - the state is currently in the $OPEN$ list; the state has been encountered (generated), but has not yet been expanded; it's g-value is finite (assuming that when states with infinite g-values are encountered, they are not put in the $OPEN$ list); the state is in the back-pointer tree.
- $inCLOSED$ - the state is currently in the $CLOSED$ list; the state has been generated and expanded; it's g-value is finite; the state is in the back-pointer tree.

One important assumption that we make when developing the tree-restoring weighted A^* algorithm is that edge cost cannot decrease between search episodes. This is certainly true in the context of Planning with Adaptive Dimensionality as mentioned above. We also assume that the weighted A^* search expands each state at most once, which preserves the sub-optimality guarantees of the algorithm (Likhachev, G. Gordon, and Thrun 2003). The tree-restoring weighted A^* algorithm keeps a discrete time variable $step$ that is initialized at 1 and incremented by 1 after every state expansion. Thus, if we record the step $C(X)$ in which a state X is generated (first placed in the $OPEN$ list, $C(X) = \infty$ if state has not yet been generated) and the step $E(X)$ in which a state is expanded (placed in the $CLOSED$ list, $E(X) = \infty$ if the state has not yet been expanded), we can reconstruct the $OPEN$ and $CLOSED$ lists at the end of any step s.

$$CLOSED_s = \{X | E(X) \leq s\}$$

$$OPEN_s = \{X | C(X) \leq s \text{ and } E(X) > s\}$$

Note that $C(X) < E(X)$ (i.e. a state's creation time is before the state's expansion time), and if $E(X) = E(X')$ then $X \equiv X'$ (i.e. no two states could have been expanded during the same step).

In order to be able to reconstruct the back-pointer tree and g-values for all states at the end of a previous step s, each state must store a history of its parents and g-values. Every time a better g-value g and parent X_p are found for a state X (when X_p is being expanded), a pair (X_p, g) is stored for the state X. Note that the pair stores the g-value of the state X itself, not the g-value of its parent X_p. Thus, we can compute the parent $P_s(X)$ and g-value $g_s(X)$ of a state X at the end of a previous step s by going through X's list L_X of stored parent/g-value pairs.

$$(P_s(X), g_s(X)) =$$

$$(X_p, g)_{\in L_X} | \forall (X', g')_{\in L_X} : E(X') \leq E(X_p) \leq s$$

In other words, the valid parent/g-value pair of X at step s is the pair containing the parent that was expanded last (most recently), but before or during step s. Storing the history in a list or array and searching it backwards seems to be very effective in quickly identifying the most recent valid parent and g-value.

When a set of states M get modified between search episodes, we identify the earliest step c_{min} in which a modified state was created: $c_{min} = min(C(X) | X \in M)$. If we then restore the search ***state*** at the end of step $c_{min} - 1$, we will end up with a valid search ***state*** with respect to the modified states, and thus, we can resume searching from there.

Algorithm 1 gives the pseudo code for all the important functions in the tree-restoring weighted A^* algorithm. Figure 3 shows an example of the tree-restoring weighted A^* algorithm used in the context of Planning with Adaptive Dimensionality.

Theoretical Properties

Theorem 1 *All states X with $C(X) > c$ will become $unseen$ after the function $restoreSearch(c)$ is called.*

Proof The function will not insert X into the $OPEN$ or $CLOSED$ lists since $C(X) > c$. $g(X)$ will be set to ∞ and the parent pointer of X will be cleared, making X $unseen$. Also, any descendant X_d of X in the back-pointer tree must have been created after X ($C(X_d) > C(X) > c$). Thus, the call to $restoreSearch(c)$ will make X_d $unseen$ as well.

Algorithm 1 Tree-Restoring Weighted A^* Search Algorithm

```
Data:
CLOSED : Set
OPEN : MinHeap
CREATED : Array
step : Integer

function INITIALIZESEARCH(X_S X_G)
    CLOSED ← ∅
    OPEN ← {X_S}
    g(X_S) ← 0
    f(X_S) ← g(X_S) + ε · h(X_S)
    step ← 1
    C(X_S) ← 0
    insert(CREATED, X_S)
    E(X_S) ← ∞
end function

function RESUMESEARCH( )
    if heuristic has changed then
        recompute heuristic
        update f-values and re-order OPEN
    end if
    while OPEN ≠ ∅ do
        X ← extractMin(OPEN)
        if X = X_G then
            return reconstructPath()
        end if
        Expand(X)
    end while
end function

function EXPAND(X)
    for all X' ∈ successors of X do
        if X' ∉ CLOSED and isValidState(X') then
            g' ← g(X) + cost(X, X')
            if (X' ∉ OPEN or g' ≤ g(X')) and g' ≠ ∞ then
                g(X') ← g'
                storeParent(X',(X, g'),step)
                f(X') ← g' + ε · h(X')
                if X' ∉ OPEN then
                    insertOPEN(X', f(X'))
                    C(X') ← step        ▷ record when state is encountered first
                    insert(CREATED, X')
                else
                    updateOPEN(X', f(X'))
                end if
            end if
        end if
    end for
    E(X) ← step                         ▷ record when state is expanded
    insert(CLOSED, X)
    step ← step + 1
end function

function RESTORESEARCH(s)
                ▷ restores the search state to just after the expansion at step s
    OPEN ← ∅
    CLOSED ← ∅
    CREATED ← ∅
    if s ≤ 0 then
        initializeSearch(X_S, X_G)
        return
    end if
    for all X ∈ CREATED do
        if E(X) ≤ s then
                ▷ the state has been created before and expanded before or during
step s
            (X_p, g) ← updateParents(X, s)
            g(X) ← g
            parent(X) ← X_p
            insert(CLOSED, X)
        else if C(X) ≤ s then
                ▷ the state has been created, but not expanded at step s
            (X_p, g) ← updateParents(X, s)
            g(X) ← g
            parent(X) ← X_p
            f(X) ← g + ε · h(X')
            insertOpen(X, f(X))
            E(X) ← ∞
        else
                ▷ the state has not been created at step s
            clearParents(X)
            g(X) ← ∞
            parent(X) ← ∅
            C(X) ← ∞
            E(X) ← ∞
        end if
    end for
    step ← s + 1
end function

function UPDATEPARENTS(X, s)
    latestG ← 0
    latestParent ← ∅
    latestParentStep ← 0
    for all (X_p, g_p) in stored parent/g-value pairs of X do
        if E(X_p) ≤ s then
                ▷ X_p is a valid parent for step s
            if E(X_p) > latestParentStep then
                ▷ Found more recent parent
                latestParentStep ← E(X_p)
                latestParent ← X_p
                latestG ← g_p
            end if
        else
            ▷ X_p is not a valid parent for step s as it has not been expanded before
or during step s
            Remove (X_p, g_p) from stored parent/g-value pairs
        end if
    end for
    return (latestParent, latestG)
end function
```

Theorem 2 *The contents of the $OPEN$ and $CLOSED$ lists after the function $restoreSearch(c)$ is called are identical to what they were at the end of step c of the algorithm.*

Proof Let $OPEN_c$ and $CLOSED_c$ be the $OPEN$ and $CLOSED$ lists at the end of step c of the algorithm. Let $OPEN'$ and $CLOSED'$ be the $OPEN$ and $CLOSED$ lists after the function $restoreSearch(c)$ is called. Let $X \in CLOSED_c$, then X has been created and expanded before or during step c. Thus, $C(X) < c$ and $E(X) \leq c$. Then X will be placed in $CLOSED'$ by $restoreSearch(c)$. Let $X \in CLOSED'$, then $C(X) < c$ and $E(X) \leq c$. Thus, X has been created and expanded before or during step c of the algorithm and $X \in CLOSED_c$. Let $X \in OPEN_c$, then X has been created, but not yet expanded at the end of step c. Thus, $C(X) \leq c$ and $E(X) = \infty$. Then X will be placed in $OPEN'$ by $restoreSearch(c)$. Let $X \in OPEN'$, then $C(X) \leq c$ and $E(X) > c$. Thus X has been created, but not yet expanded at the end of step c. Then $X \in OPEN_c$. Thus, $OPEN_c \equiv OPEN'$ and $CLOSED_c \equiv CLOSED'$.

Theorem 3 *All states X with $C(X) \leq c$ will have correct g-values and parent pointers after $restoreSearch(c)$ is called.*

Proof We proceed with a proof by contradiction. Suppose a state X has an incorrect parent pointer. In other words, there exists a state $P' \in CLOSED$ such that $g(P') + cost(P', X) < g(P) + cost(P, X)$ (a better parent P' for X exists in the $CLOSED$ list).

Suppose P was expanded before P'. Then $E(P) < E(P') \leq c$. The call to $updateParents(c)$, then should have found P' as the parent of X as P' has been expanded more recently than P, but still before or during step c—a contradiction. Then, P must have been expanded after P' and $E(P') < E(P) \leq c$. However, since the g-value obtained through P is larger than the g-value obtained through P', P would not have been recorded as a parent of X when P was expanded because a better parent had been found already. Thus, P could not be a parent of X—contradiction.

Thus, the parent pointers for all states are correctly computed by $restoreSearch(c)$. Since parent pointers are stored together with their corresponding g-values, then $restoreSearch(c)$ also computes the correct g-values for all states.

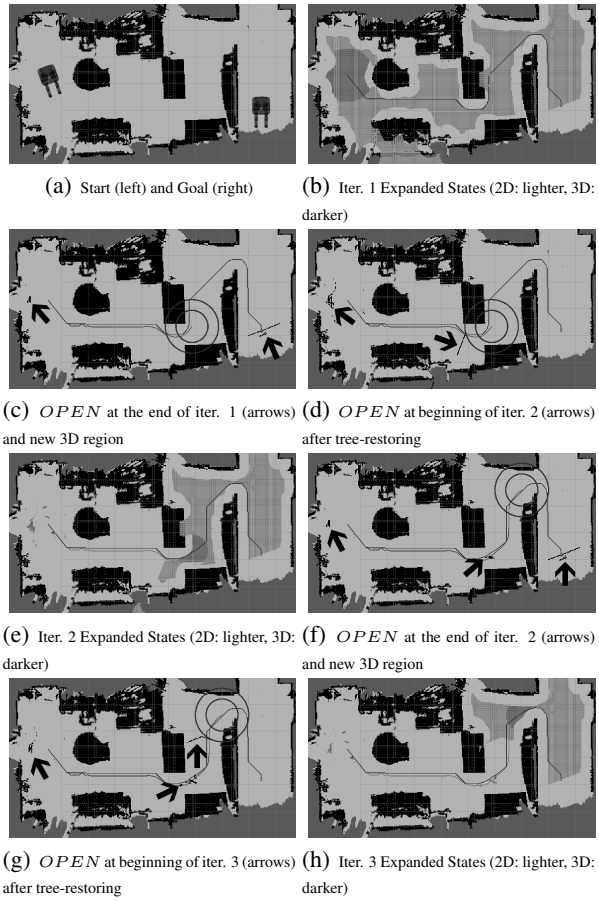
Theorem 4 *Let M be the set of all modified states after a successful incremental A^* search episode. Let $c_{min} = min(C(X)|X \in M)$. $restoreSearch(c_{min} - 1)$ results in a search **state** that is valid.*

Proof The result follows directly from the above theorems.

Since introducing high-dimensional regions into the adaptive graph G^{ad} cannot decrease edge costs, the heuristic function remains admissible (underestimating the actual cost to goal) for all search episodes of the incremental search. Therefore, the tree-restoring weighted A^* graph search algorithm preserves the theoretical properties of weighted A^* such as termination, completeness, and bounds on solution cost sub-optimality in the context of Planning with Adaptive Dimensionality.

Experimental Results

The domain we chose to experimentally validate our algorithm was path planning for non-holonomic vehicles in three dimensions—(x,y,$heading$) with full-body collision checking. We used Willow Garage's PR2 robot as our experimental platform. We compared three implementations of



(a) Start (left) and Goal (right) (b) Iter. 1 Expanded States (2D: lighter, 3D: darker) (c) $OPEN$ at the end of iter. 1 (arrows) and new 3D region (d) $OPEN$ at beginning of iter. 2 (arrows) after tree-restoring (e) Iter. 2 Expanded States (2D: lighter, 3D: darker) (f) $OPEN$ at the end of iter. 2 (arrows) and new 3D region (g) $OPEN$ at beginning of iter. 3 (arrows) after tree-restoring (h) Iter. 3 Expanded States (2D: lighter, 3D: darker)

Figure 3: Example of Planning with Adaptive Dimensionality using tree-restoring A^* search (with no heuristic for illustration purposes). New high-dim. regions introduced in the graph are represented by the inner circles. The outer circles represent states that are affected by the introduction of the new region (modified states). Dark cells indicated by arrows represent the $OPEN$ list (search frontier). Note the reduction of the number of expanded states as iterations progress.

the algorithm for Planning with Adaptive Dimensionality—the non-incremental version of the algorithm, an incremental version using tree-restoring weighted A^*, and an incremental version using D^*-Lite planner (Koenig and Likhachev 2002b). We used the same approach to 3-DoF planning as in (Gochev et al. 2011)—we used lattice-based graphs of uniform resolution (2.5cm×2.5cm) and heading angle values were uniformly discretized into 16 on the interval $(-\pi, \pi]$. We used a set of pre-computed transitions for a non-holonomic robot for 3D states and simple 8-connected 2D grid transitions for the 2D states. The costs of 2D transitions were representative of the distance traveled and the costs of 3D transitions were computed based on the distance traveled, inflated by a pre-computed penalty factor: 3D transitions that required the robot to move backwards had higher penalty factors than transitions moving forward. We used a 2D 8-connected grid-based distance-to-goal heuristic, accounting for obstacles. The heuristic values were computed by a single backward Dijkstra search on the 2D grid. In the incremental versions of the algorithm, every time a new high-dimensional region was introduced, all states falling inside the region and all states on the boundary of the region (states that have valid high-dimensional transitions into the

Algorithm	Sub-opt. Bound	Time (s) mean	std dev	min	max	# Iterations mean	std dev	# 3D Expands mean	std dev	# 2D Expands mean	std dev	Total Expands mean	std dev	Successful Searches
3D Weighted A^*	5.0	39.41	34.45	2.42	118.57	n/a		37.29K	32.53K	n/a		37.29K	32.53K	23 of 30
Non-incremental Adaptive	5.0	14.43	15.92	0.89	48.69	2.07	1.10	13.92K	15.52K	1.41K	0.99K	15.31K	16.39K	30 of 30
Tree-restoring A^* Adaptive	5.0	6.86	2.75	0.89	21.75	2.07	1.10	6.83K	6.34K	0.69K	0.29K	7.51K	6.59K	30 of 30
Incremental D^* Adaptive	5.0	10.40	10.80	0.89	34.97	2.07	1.10	7.35K	8.96K	2.22K	1.76K	9.55K	9.96K	30 of 30
Bi-directional RRT	n/a	22.56	20.48	0.03	87.87	n/a		n/a		n/a		n/a		286 of 300

Table 1: Experimental results on 30 scenarios for 3-DoF (x,y,$heading$) path planning (weighted A^* planner vs. non-incremental adaptive-dimensionality planner vs. adaptive-dimensionality planner using tree-restoring weighted A^* vs. adaptive-dimensionality planner using D^*-Lite vs. sampling-based bi-directional RRT planner). The deterministic search-based planners were run only once on each scenario. RRT results are averaged over 10 runs on each scenario. The reported times for RRT do not include trajectory post-smoothing. A search was reported as successful if it took less than 60 seconds to compute a path to the goal.

Algorithm	Sub-opt. Bound	Time (s) mean	std dev	min	max	# Iterations mean	std dev	Time spent in planning phase mean	std dev	min	max
Non-incremental Adaptive	5.0	22.91	15.50	6.41	48.69	2.78	0.83	15.35	13.28	2.11	40.21
Incremental A^* Adaptive	5.0	12.03	5.67	4.96	21.75	2.78	0.83	5.16	3.30	1.42	9.75
Inremental D^* Adaptive	5.0	17.08	10.87	5.29	34.97	2.78	0.83	10.94	8.94	1.69	27.27

Table 2: Statistical data for the 18 scenarios that required more than one iteration of planning demonstrating the benefits of using incremental graph searches in the context of Planning with Adaptive Dimensionality. Using tree-restoring weighted A^* reduced the time spent in the planning phase of the algorithm by a factor of 3.

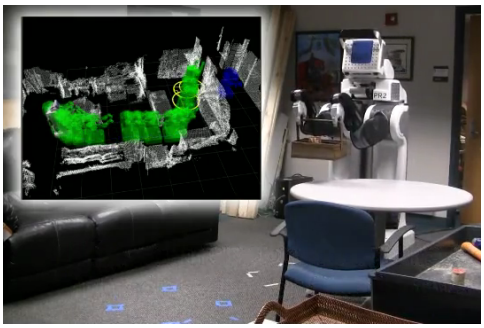


Figure 4: Example run of an Adaptive-Dimensionality planner on an indoor environment. The high-dimensional regions introduced by the algorithm, represented by circles, and the computed path are shown in the embedded figure. 3D planning is performed inside the circles and 2D planning is performed everywhere else in the environment.

region) were tagged as modified.

We also compared the three Adaptive-Dimensionality algorithms with a 3-DoF weighted A^* lattice-based planner and a 3-DoF sampling-based bi-directional RRT planner based on the approach taken in (LaValle and Kuffner 2001). The RRT planner used controllers for a non-holonomic robot with the same parameters (minimum turning radius and nominal velocity) as the 3D transitions used by the search-based planners. We ran all algorithms on 30 indoor environments of varying degree of difficulty (example can be seen in Fig. 4). Most scenarios exhibited challenging features such as pronounced heuristic local minima and narrow passages. All algorithms performed full-body collision checking (base, torso, arms and head) to ensure that the computed paths were completely collision-free. This is much more computationally expensive (orders of magnitude) than collision-checking just the footprint of the robot, but is much more precise. The Adaptive-Dimensionality planners used simpler collision checking for 2D states, treating the robot as a point and inflating the obstacles by the robot's inscribed circle radius.

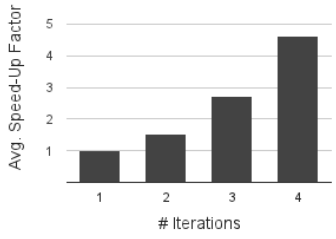


Figure 5: Relationship between the number of iterations performed and the average speed-up factor between non-incremental Adaptive-Dimensionality planner and incremental Adaptive-Dimensionality planner using tree-restoring weighted A^* observed in our 30 experimental scenarios. The incremental algorithm demonstrates better speed-up as the difficulty of the problem increases.

As seen in Table 1, both the 3-DoF weighted A^* lattice planner and the bi-directional RRT planner are outperformed by the Adaptive-Dimensionality algorithms. The poor performance of the RRT algorithm can be attributed to the many narrow passages (such as doorways and narrow gaps between furniture) present in our test environments. A significant drawback of the bi-directional RRT algorithm was the fact that it frequently produced highly sub-optimal paths and paths that required the robot to drive backwards for long periods, which we consider undesirable. The performance of the 3-DoF weighted A^* lattice planner was reasonable only in a few scenarios that did not exhibit local minima of the heuristic function. We observed that the Adaptive-Dimensionality algorithm using tree-restoring weighted A^* performed best on average, improving performance over the non-incremental version by a factor of 2.

Table 2 compares the performance of the incremental and non-incremental versions of the Adaptive-Dimensionality algorithm on 18 of the 30 scenarios, which required multiple search iterations to produce a path. On scenarios that required only a single iteration of planning, all three versions of the algorithm behaved identically, since no re-planning was needed. The Adaptive-Dimensionality algorithm using D^*-Lite performed significantly better than the non-incremental version of the algorithm, improving the overall planning time by a factor of 1.35. However, using D^*-Lite seems to introduce significantly more overhead than using the simple tree-restoring technique for incremental weighted A^* planning. This can be explained by the fact that D^*-Lite needs to generate both successor and predecessor states for all modified states in the graph in order to propagate the inconsistencies in its search tree. This involves expensive collision-checking and some book-keeping

overhead. Also, in the context of Planning with Adaptive Dimensionality, edge costs only increase when a new high-dimensional region is added, which results in underconsistent states ($g(X) < rhs(X)$, defined in (Koenig and Likhachev 2002b)) in the D^* search. D^*-Lite propagates the consistency by expanding these underconsistent states to make them overconsistent ($g(X) > rhs(X)$, defined in (Koenig and Likhachev 2002b)), after which it may have to expand these states again to make them consistent ($g(X) = rhs(X)$). Thus, while attempting to correct its search tree, D^*-Lite might have to expand many states twice, which introduces significant overhead. On the other hand, the tree-restoring weighted A^* does not attempt to correct its search tree, but rather quickly identifies a usable portion of the search tree and resumes searching from there. In our experiments we observed that the tree-restoring weighted A^* algorithm needed an average of 5 milliseconds to restore itself to a valid previous search ***state*** and resume searching. As a result, tree-restoring weighted A^* improves the performance of the planning phase of the algorithm for Planning with Adaptive dimensionality by a factor of 3 and seems to be a better incremental search alternative than D^*-Lite in this context. As shown in Fig. 5, the performance benefit of using tree-restoring weighted A^* increases as the difficulty of the search problem increases and more iterations are needed to solve it.

Discussion and Future Work

To verify that the Adaptive-Dimensionality algorithm using tree-restoring weighted A^* scales well with increasing the dimensionality of the problem, we have begun work on applying it in the context of 4-DoF (x,y,z,yaw) navigation for an unmanned areal vehicle and 11-DoF mobile manipulation planning on the PR2 platform as in (Gochev, Safonova, and Likhachev 2012). Our preliminary results are very similar to the results presented in this work and suggest that the performance of the algorithm does not deteriorate significantly in larger state-spaces.

From the results shown in Table 2 we can see that the Adaptive-Dimensionality algorithm using incremental weighted A^* spends roughly 43% of its overall search time in planning phases and 57% in tracking phases. We have seen that incremental search techniques can significantly improve the performance of the planning phases of the algorithm. In the future, we would like to investigate the possibility of performing incremental searches to speed up the tracking phases as well. This problem is challenging, since the tunnel τ being searched during the tracking phase is only a subgraph of G^{hd} and can change drastically between subsequent iterations.

Conclusion

In this work we have presented an incremental version of the previously-developed algorithm for Planning with Adaptive Dimensionality using a simple, but effective technique for performing incremental weighted A^* graph searches. We have proven that this technique preserves the theoretical properties of weighted A^*, such as completeness and bounds on solution cost sub-optimality. Experimentally, we have demonstrated that using incremental weighted A^* graph search can improve the performance of the algorithm for Planning with Adaptive Dimensionality by up to a factor of 5 in the context of 3-DoF (x, y, $heading$) path planning for a non-holonomic vehicle. We also believe that tree-restoring weighted A^* can be used more generally than in the context of Planning with Adaptive Dimensionality. We plan to investigate this further.

References

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.

Gochev, K.; Cohen, B.; Butzke, J.; Safonova, A.; and Likhachev, M. 2011. Path planning with adaptive dimensionality. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*.

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

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: a graph oriented approach. *Artif. Intell.* 85(1-2):321–361.

Koenig, S., and Likhachev, M. 2002a. Incremental A*. In Dietterich, T. G.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems (NIPS) 14*. Cambridge, MA: MIT Press.

Koenig, S., and Likhachev, M. 2002b. D*-lite. In *Eighteenth national conference on Artificial intelligence*, 476–483. Menlo Park, CA, USA: American Association for Artificial Intelligence.

LaValle, S., and Kuffner, J. 2001. Randomized kinodynamic planning. *International Journal of Robotics Research* 20:378–400.

Likhachev, M.; G. Gordon; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS)*. Cambridge, MA: MIT Press.

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

Stentz, A. 1995. The focussed D* algorithm for real-time replanning. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, 1652–1659. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Sun, X.; Koenig, S.; and Yeoh, W. 2008. Generalized adaptive A*. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, 469–476. Richland, SC: Interna-

tional Foundation for Autonomous Agents and Multiagent Systems.

Sun, X.; Yeoh, W.; and Koenig, S. 2009. Dynamic fringe-saving A*. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, 891–898. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

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.

Trovato, K. I., and Dorst, L. 2002. Differential A*. *IEEE Trans. on Knowl. and Data Eng.* 14(6):1218–1229.