# Search-Based Planning with Provable Suboptimality Bounds for Continuous State Spaces

**Juan Pablo Gonzalez**
Autonomous Perception Research
General Dynamics Robotic Systems
Pittsburgh, PA 15221
jpgonzal@gdrs.com

**Maxim Likhachev**
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
maxim@cs.cmu.edu

## Abstract

Search-based planning is widely used for mobile robot motion planning because of its guarantees of optimality and completeness. In continuous state-spaces, however, most existing approaches have significant limitations in terms of optimality and completeness because of the underlying grid used. We propose an approach that eliminates the dependency on grids by using more general equivalence classes to quickly find an initial solution and instead of pruning states that fall within an equivalence class and have higher cost, we use an inflated heuristic to lower the priority of these states in the search. In further iterations, we reduce the inflated heuristic in a principled way, thus providing fast solutions with provable suboptimality bounds that can be improved as time allows. The proposed approach produces smooth paths with the resolution dictated by the action set. Finer action sets produce higher resolution paths that are more computationally intensive to calculate and coarser action sets produce lower resolution paths that are faster to compute. To the best of our knowledge, this is the first algorithm that is able to plan in *continuous* state-spaces with provable guarantees on completeness and bounds on suboptimality for a given action set. Experimental results on 3D (x,y,$\theta$) path planning show that, on average, this approach is able to find paths in less than two seconds that are within 2% of the optimal path cost in worlds of up to 1000x1000 m with a minimum step size of one meter.

## Introduction

Search-based planning is widely used for mobile robot motion planning because of its guarantees of optimality and completeness. In continuous state-spaces, however, most existing approaches have significant limitations in terms of optimality and completeness. Existing approaches to search-based planning in continuous state-spaces generally belong to two categories: grid-based planners and lattice-based planners. Grid-based planners for planning in continuous state-spaces were first introduced by Barraquand and Latombe (Barraquand and Latombe 1993). Their approach uses a fixed-cell decomposition that defines implicit grid-like equivalence classes. States are generated as part of an A* search driven by a forward simulation through an action set. States that fall within cells with states that have already been expanded (*CLOSED*) are pruned. This approach produces high-quality paths and is relatively fast. However, it can prune states that are needed to find optimal solutions in constrained environments, as can be seen in Figure 1.

Lattice-based planners (Pivtoraiko and Kelly 2005;

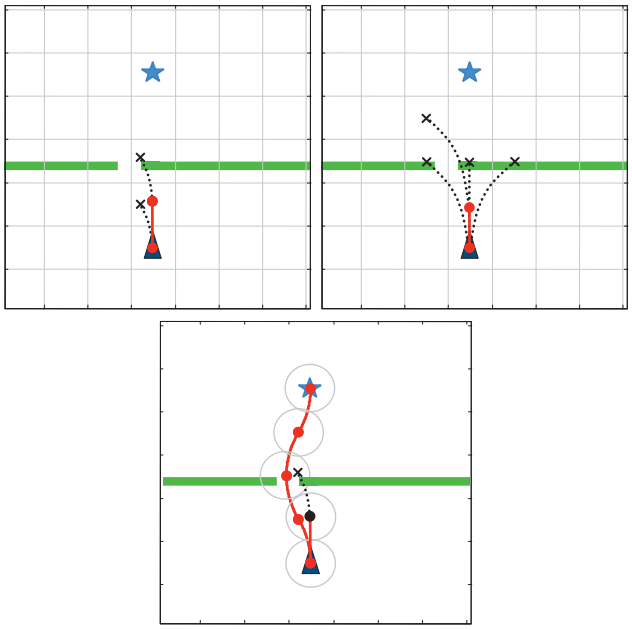

Figure 1. Comparison of grid-based planning (top left), lattice-based planning (top right) and our approach (bottom). Dashed transitions with cross at the end correspond to pruned transitions. In grid-based planning, states that are within an existing equivalence class and have higher cost are pruned. In lattice-based planning the action set is modified to always land on the center of grid locations. Our approach overcomes both limitations and plans with provable guarantees on suboptimality, allowing the path to the goal to be found independently of the characteristics of the equivalence classes

Likhachev and Ferguson 2009) use state lattices, a discretization of the configuration space into a set of states that represent configurations and connections between these states, and where every connection represents a feasible path. They guarantee feasibility of the paths while allowing search-based planning on the resulting graph. The main drawback of lattice-based planners is that they need to convert the desired action set into an action set that always starts and ends on a node in the grid. This requirement increases the design complexity of the planners and artificially restricts the paths to go through a series of specific points. Figure 1 also shows an example where the constraints in the location of the path points prevents the planner from finding a solution. In this example, in order to find a path to the goal using a lattice-based planner, a higher resolution would be needed for the lattice since the obstacles in the world are not aligned with the cell centers.

Sampling-based planners (Kavraki et al 1996; LaValle 1998) use forward search in continuous coordinates and are able to find smooth paths in high-dimensional spaces without the need for an underlying grid. Sampling based approaches are probabilistic-complete, but usually produce highly suboptimal paths that require post-processing to find locally-optimal solutions, and cannot optimize continuous cost functions. RRT* (Karaman and Frazzoli 2010) is a unique variant of sampling-based planners that reconnects vertices within a d-dimensional ball whose radius shrinks as the search progresses. RRT* guarantees asymptotic optimality, but does not provide any deterministic suboptimality bounds if the search terminates before finding an optimal path.

We propose an approach that improves upon grid-based planners by using equivalence classes to find an initial solution, but instead of pruning states that fall within an equivalence class and have higher cost, we use an inflated heuristic to lower the priority of these states in the search. In further iterations, we use ARA* (Likhachev, Gordon, and Thrun 2003b) to reduce the inflated heuristic in a principled way. This allows us to provide fast solutions with provable suboptimality bounds that can be improved as time allows. The proposed approach produces smooth paths with the resolution dictated by the action set. Finer action sets produce higher resolution paths that are more computationally intensive to calculate and coarser action sets produce lower resolution paths that are faster to compute. To the best of our knowledge, this is the first algorithm that is able to plan in continuous state-spaces with provable guarantees on suboptimality for a given action set.

Experimental results on 3D (x,y,$\theta$) path planning show that, on average, this approach is able to find paths in less than two seconds that are within 2% of the optimal path cost in worlds of up to 1000x1000 m with a minimum step size of 1m.

### Outline

To simplify the presentation we present three variants of the algorithm in order of complexity. The first one, A* with equivalence classes, prunes higher-cost states that are already represented in an equivalence class (equivalence-class state dominance). It is somewhat similar to the Barraquand-Latombe approach, although it does not use the grid to define equivalence classes. This version does not have explicit guarantees on suboptimality bounds. The second version, $\varepsilon$-optimal A* with equivalence classes, does not prune states and instead uses an inflated heuristic $\varepsilon > 1$ to penalize dominated states. This algorithm is guaranteed to find a solution that is within $\varepsilon$ of the optimal solution for the given action set. The final version, Anytime Repairing A* with equivalence classes, combines the previous two versions in an anytime fashion to quickly obtain an initial solution with large $\varepsilon$ and then reduces $\varepsilon$ while reusing previous results in order to achieve the minimum $\varepsilon$ possible within the time and space available. The solution found is also guaranteed to be within $\varepsilon$ of the optimal solution for the given action set.

## Anytime Repairing A* with Equivalence Classes

### A* with Equivalence Classes – AE*

In order to plan paths that don't depend on a grid, we use a dynamically generated graph created by applying an action set to each state expanded. As in regular A*, we use two lists, *OPEN* and *CLOSED*. The *OPEN* list contains states that are candidates for expansion, ordered by their *f* value $f(s) = g(s) + h(s)$, where $g(s)$ is the accumulated cost from the start, and $h(s)$ is an *admissible* heuristic that underestimates the cost to the goal. The *CLOSED* list contains states that have already been expanded. The nodes of the graph are made up of the states belonging to the OPEN and CLOSED lists together, and the edges are created as nodes are generated.

Initially $s_{start}$ is placed in the *OPEN* list (Figure 2, line 02) with $f(s_{start}) = h(s_{start})$. The state with the lowest *f* value is popped off the *OPEN* list, put in the *CLOSED* list, and expanded, with its successors *s' generated and* determined by the action set (lines 04 to 06). A state *s'* can lie anywhere in the state space (as determined by the action set) and therefore it is not discretized. We calculate the cost to transition from *s* to *s'* and update $g(s')$ if the state *s'* has not been generated before or if its *g* value can be improved (lines 07 to 09). Then, out of all the states in the graph *G* that we have constructed so far, we look for the

```
01  g(s_start) = 0; OPEN = ∅ ; CLOSED = ∅
02  insert s_start into OPEN with f(s_start) = h(s_start)
03  while s_goal is not expanded
04   remove s with the smallest f value from OPEN
05   CLOSED =CLOSED ⋃ {s}
06   for each succesor s' of s
07    if ( s' hasn't been generated ) OR
08        (g(s) + c(s,s') < g(s'))
09      g(s') = g(s) + c(s,s')
10      s_nearest=Nearest(OPEN ⋃ CLOSED, s')
11      if (s_nearest = { ∅ })
12        insert s' into OPEN with f(s') = g(s') + h(s')
13      else if g(s') < g(s_nearest)
14        if s_nearest ∈ OPEN
15            remove s_nearest from OPEN
16        insert s' into OPEN with f(s') = g(s') + h(s')
```

Figure 2. Algorithm 1: A* with equivalence classes

nearest neighbor $s_{nearest}$ within the equivalence class of $s'$ according to a given distance function $D(s,s')$.

We define the equivalence relationship between two states as

$$s \sim s' \leftrightarrow D(s,s') < \delta \tag{1}$$

where $D$ is the same distance function used to evaluate the nearest neighbor, and $\delta$ defines the size of the equivalence class. If there are no neighbors within the same equivalence class, then $s'$ is considered a new state and it's placed in the *OPEN* list with $f(s') = g(s') + h(s')$ (lines 11 and 12). If there is a neighbor within the same equivalence class, then $s'$ is only added to the open list (and to the graph) if its $g$ value is lower than that of the nearest neighbor (lines 13 to 16). Figure 3 illustrates this process. Every time a state is placed in the *OPEN* list, backpointers representing connecting edges to its parent are also stored with it. Lines 03 to 16 are repeated until a state in the same equivalence class as the goal state is expanded or until the *OPEN* list is empty. The size of the equivalence class together with the action set determines the resolution of the solution. Smaller equivalence classes achieve higher resolution at the expense of computation time and space. The upper bound on the size of the equivalence class is determined by the action set since the successors of a node

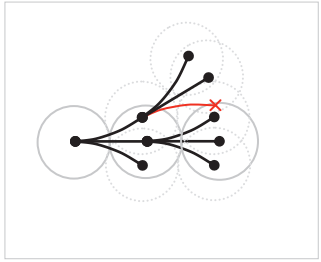

Figure 3. Using equivalence classes, states are allowed to lie where the action set dictates. The solid gray circles show the equivalence classes for $\theta$ = 0º, and dashed circles show the equivalence classes for other angles. The red "x" shows a state that is dominated (pruned) because it is within an existing equivalence class and has a higher $g$ value.

need to belong to a separate equivalence class than its parent

$$\delta \le D(s,s'). \tag{2}$$

Equivalence classes affect the quality of the solution in an indirect way (Gonzalez and Stentz 2009). Within an equivalence class states are dominated based on their $g$ value (total cost from the start). If there is a state with a higher $g$ *value* within the same equivalence class that would be needed to achieve a better solution later in the search, then this state would be pruned, and the better solution would not be achieved. The likelihood of this event depends on the topology of the problem and the size of the equivalence class. As the equivalence class gets smaller, this is less likely to occur and in the limit (if the size of the equivalence class is zero), it will never happen. In practice we usually set the size of the equivalence classes at the upper bound determined by the action set, as the resulting paths are of high quality and this reduces planning time.

For example, if we are planning paths in $\{x,y,\theta\}$ for a car-like robot moving forward with velocity $v$ and minimum turning radius $\rho_{\min}$ (Dubins car), the action set U in $\{v,\theta\}$ is

$$\mathrm{U} = v \cdot \Delta t \times \{-v/\rho_{\min}, 0, v/\rho_{\min}\} \tag{3}$$

where $\Delta t$ is the time step in the action set. The longitudinal motion determines the maximum size of the equivalence class in xy ($\delta_{xy} = v \cdot \Delta t$) and the heading change determines the maximum size of the equivalence class in $\theta$ ($\delta_\theta = v \cdot \Delta t / \rho_{\min}$). For a given speed and turning radius, increasing $\Delta t$ increases the size of the equivalence class and produces a coarser solution in (x,y, $\theta$). However, since the successors are generated based on a feasible action set and there is no quantization, the resulting path is always continuous in all variables.

Figure 4 shows the resulting paths for $v$ = 1m/s and $\Delta t$ = 1s and $\Delta t$ = 5s, as well as a path using A* on a grid in $xy$. For $\Delta t$ =1, each motion on the $xy$ plane is 1 meter long, and the maximum change of heading in the action set corresponds to approximately 6 degrees. For $\Delta t$ = 5s, each motion on the $xy$ plane is 5 meters long, and the maximum change in heading in the action set is 30 degrees. Although a quantization in heading is not imposed, the action set selected effectively limits the possible angles to multiples of 6 degrees for $\Delta t$ = 1s and to multiples of 30 degrees for $\Delta t$ = 5s. The $xy$ values are not quantized by the action set selected.

This approach is similar to the approach proposed by Barraquand and Latombe, but it uses a more general definition of equivalence classes. Where their approach was defined for grid-like equivalence classes, the approach presented here is not restricted in the shape of the equivalence class. Like the Barranquand-Latombe approach, this approach lacks completeness guarantees and

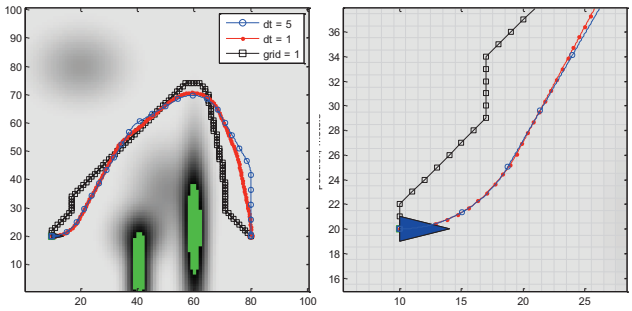


Figure 4. Comparison of A* with equivalence classes in $(x,y,\theta)$ for a car-like robot with v=1, $\rho_{min}$=10 and $\Delta t$=1 s (dots) and $\Delta t$=5 s (circles) as well as A* in $(x,y)$ with a 1-meter grid (squares). The robot starts in the lower left corner, heading east. Light gray areas are low cost, dark dray areas are higher cost and green areas are non-traversable. Notice the smaller steps and smaller heading changes for $\Delta t$=1. For $\Delta t$=5 steps and heading changes are larger, but the path is still continuous.

provable suboptimality bounds, as it prunes states within equivalence classes that could be necessary later on in the search.

## $\varepsilon$-optimal A* with Equivalence Classes

Equivalence class state-dominance can prune states that are necessary to achieve truly optimal paths. In order to obtain provable suboptimality bounds and to allow the solution to be improved beyond the limits of the equivalence class defined, we modify it as follows.

Instead of pruning states that are dominated within the equivalence class, we put them in the *OPEN* list with an inflated heuristic such that $f(s) = g(s) + \varepsilon h(s)$, with $\varepsilon \geq 1$. This is similar to implementing A* with a weighted heuristic, but only the dominated nodes are weighted. Figure 5 shows the modified algorithm, highlighting the changes that implement the inflated heuristic. Since the heuristic is at most $\varepsilon$ times the admissible heuristic $h(s)$, the solution will be at most $\varepsilon$ times the optimal solution (Likhachev, Gordon, and Thrun 2003a). With large $\varepsilon$ the solution is the same as that found with algorithm 1. With smaller $\varepsilon$ more states will be considered in the solution, overcoming the limitations imposed by the equivalence classes but also requiring more time and space.

In order to implement weighted A* with equivalence classes, a number of additional changes are required. We create a new list *NOTDOM* to contain all the non-dominated states generated so far. Each non-dominated state defines an equivalence class, and has lower $g$ value than any other states within the same equivalence class. As such, in line 10, we only check for the nearest neighbor within *NOTDOM*. We maintain this list by adding new states that don't have a non-dominated state nearby (line 12) and by checking each generated state against its nearest non-dominated neighbor. If a successor *s'* has a lower $g$ value than the nearest non-dominated neighbor within its equivalence class, then the nearest neighbor is removed from *NOTDOM* and *s'* inserted instead (lines 15 and 16). If $s_{nearest}$ was in the *OPEN* list, then its $f$ value changed to that of dominated states (lines 19 and 20). If a successor *s'* has a greater or equal $g$ value than $s_{nearest}$ then *s'* is put in the *OPEN* list as a dominated state (line 24), unless, *s'* is equal to $s_{nearest}$. If *s'* is equal to $s_{nearest}$. then *s'* is non-dominated and its $g$ value was updated to a better value in line 09. In such case, *s'* is put in the *OPEN* list as a non-dominated state (line 26).

**Theorem: If there exists a finite-cost path for the given action set, then the algorithm is guaranteed to terminate and to return a path whose cost is no more than $\varepsilon$ times the cost of the least-cost path for the given action set.**

We prove termination by contradiction. Let us consider a least-cost path from $s_{start}$ to $s$, and in particular the state $s'$ on it that has never been expanded and is closest to $s_{start}$. $s'$ was generated and inserted into OPEN with a finite priority. Given a finite set of actions for each state and strictly positive costs, there is only a finite number of states whose $g$ values will be smaller than any finite $g$ value including the $g$ value of $s'$. Thus, $s'$ must have been selected for expansion, which is a contradiction.

According to the theoretical analysis of weighted A* (Likhachev, Gordon, and Thrun 2003a), inflating $h$ values of states with $\varepsilon \geq 1$ guarantees that whenever a state $s$ is expanded, the cost of the found path from $s_{start}$ to $s$ is bounded from above by the $g$ value of $s$, which in turn is no more than $\varepsilon$ times the cost of an optimal path from $s_{start}$ to $s$. This means that the cost of the path returned by our algorithm is no more than $\varepsilon$ times the cost of a least-cost path.

## Anytime Repairing A* with Equivalence Classes

Since different environments have very different values of $\varepsilon$ that can be solved in a given amount of time and space, it is desirable to have an approach that explores different values of $\varepsilon$ in an efficient manner. Anytime Repairing A* (ARA*, Likhachev, Gordon, and Thrun 2003b) is an anytime algorithm similar to weighted A*, but rather than having a fixed value of $\varepsilon$, it starts with a large $\varepsilon$ and it decreases it while reusing previous search results. In Figure 6 we introduce ARAE*, which extends ARA* to use equivalence classes. The procedure *ImprovePath()* is very similar to $\varepsilon$-optimal A* with equivalence classes, except for the differences highlighted. Like ARA*, ARAE* uses *local inconsistencies* to propagate improvements in the solution in an efficient manner. ARAE* uses three mutually exclusive lists: *OPEN, CLOSED* and *INCONS*. *OPEN* contains all the states that have never been expanded within the current iteration of *ImprovePath*. These states have been discovered for the first time within the current search iteration or had their $g$-

```
01  g(s_start) = 0; OPEN = ∅ ; CLOSED = ∅
02  insert s_start into OPEN with f(s_start) = h(s_start)
03  while s_goal is not expanded
04   remove s with the smallest f value from OPEN
05   CLOSED =CLOSED ⋃ {s}
06   for each succesor s' of s
07    if ( s' hasn't been generated ) OR
08        (g(s) + c(s,s') < g(s'))
09      g(s') = g(s) + c(s,s')
10      s_nearest = Nearest(NOTDOM, s')
11      if (s_nearest = { ∅ })
12        NOTDOM = NOTDOM ⋃ {s'}
13        insert s' into OPEN with f(s') = g(s') + h(s')
14        else if g(s') < g(s_nearest)
15        remove s_nearest from NOTDOM
16        NOTDOM = NOTDOM ⋃ {s'}
17        if s_nearest ∈ OPEN
18          remove s_nearest from OPEN
19          insert s_nearest into OPEN with
20              f(s_nearest) = g(s_nearest) + ε·h(s_nearest)
21        insert s' into OPEN with f(s') = g(s') + h(s')
22      else // g(s') ≥ g(s_nearest)
23        if ( s' ≠ s_nearest)
24          insert s' into OPEN with f(s') = g(s') + ε·h(s')
25        else
26          insert s' into OPEN with f(s') = g(s') + h(s')
```

Figure 5. Algorithm 2: $\varepsilon$-optimal A* with equivalence classes. Lines in gray are functionally equivalent to the previous algorithm, while lines in black are unique to this algorithm.

values decreased within the current search iteration-. *CLOSED* contains all states that have been expanded within the *current* iteration of *ImprovePath* and whose *g*-values have not decreased after their last expansion. *INCONS* contains all the states that have been expanded within the current search iteration and whose *g*-values *did* decrease after the expansion (lines 22, 26 and 31)

The main difference between ARA* and weighted A* is the use of the *INCONS* list. ARA* postpones the expansion of any inconsistent states until the next iteration of *ImprovePath*. By doing this ARA* prevents states from expanding more than once during each iteration, which usually produces a faster initial solution. In ARAE* we generalize this idea to equivalence classes by using an $\varepsilon$-inflated *f* value for states *s'* with lower *g* value than $s_{nearest}$ if $s_{nearest}$ has already been expanded in the current iteration (line 25). The termination condition of *ImprovePath()* is also different from that of weighted A*. Since the goal node may not become inconsistent, ARAE* terminates when the *f* value of $s_{goal}$ is equal to the minimum *f* value of the states in the *OPEN* list.

The main loop calls *ImprovePath* repeatedly, decreasing $\varepsilon$ between calls. Before each call to *ImprovePath*, the states in *INCONS* are added to the *OPEN* list. Then the *OPEN* list is re-ordered using the new $\varepsilon$ (line 10). Unlike ARA*, ARAE* differentiates dominated and non-dominated states by only inflating the heuristics of the dominated states in ***fvalue***(*s*). Also notice that states that had a lower *g* value than $s_{nearest}$ when $s_{nearest}$ had already been expanded will change their priority to that of non-dominated states, therefore no longer discouraging their expansion.

```
Procedure fvalue(s)

01  if s ∈ NOTDOM
02    return g(s) + ε·h(s)
03  else
04    return g(s) + h(s)

Procedure: ImprovePath()

01  CLOSED = ∅ ; INCONS = ∅
02  while fvalue(s_goal) > min_{s∈OPEN} (fvalue(s))
03   remove s with the smallest fvalue(s) from OPEN
04   CLOSED =CLOSED ⋃ {s}
05   for each succesor s' of s
06    s_nearest = Nearest(NOTDOM, s')
07    if ( s' hasn't been generated ) OR (g(s) + c(s,s') < g(s'))
08      g(s') = g(s) + c(s,s')
09      if (s_nearest = { ∅ })
10        NOTDOM = NOTDOM ⋃ {s'}
11        insert s' into OPEN with f(s') = g(s') + h(s')
12      else if (g(s') < g(s_nearest))
13        remove s_nearest from NOTDOM
14        NOTDOM = NOTDOM ⋃ {s'}
15        if s_nearest ∉ CLOSED
16          if (s_nearest in OPEN)
17            re-insert s_nearest into OPEN with
18                f(s_nearest) = g(s_nearest) + ε·h(s_nearest)
19          if (s' ∉ CLOSED)
20            insert s' into OPEN with
21                f(s') = g(s') + h(s')
22          else INCONS =INCONS ⋃ {s'}
23        else //s_nearest was CLOSED
24          if (s' ∉ CLOSED)
25            insert s' into OPEN with f(s') = g(s') + ε·h(s')
26          else INCONS =INCONS ⋃ {s'}
27      else // g(s') >= g(s_nearest)
28        if (s' ∉ CLOSED)
29          insert s' into OPEN with f(s') = fvalue(s')
30        else //s' was CLOSED
31          INCONS =INCONS ⋃ {s'}

Procedure: Main()

01  g(s_start) = 0; OPEN = ∅
02  NOTDOM = ∅
03  insert s_start into OPEN with f(s_start) = h(s_start)
04  improvePath()
05  ε' = min(ε, g(s_goal) / min_{s∈OPEN∪INCONS} (g(s) + h(s)))
06  publish current ε'-suboptimal solution
07  while ε' > 1
08    decrease ε
09    OPEN= OPEN ⋃ INCONS
10    update priorities for all s ∈ OPEN according to fvalue(s)
11    ImprovePath()
12    ε' = min(ε, g(s_goal) / min_{s∈OPEN∪INCONS} (g(s) + h(s)))
13    publish current ε' suboptimal solution
```

Figure 6. Algorithm 3: ARA* with equivalence classes

ARAE* then updates the suboptimality bound $\varepsilon$ after each iteration of *ImprovePath* according to

$$\varepsilon' = min\left(\varepsilon, \frac{g(s_{goal})}{min_{s \in OPEN \cup INCONS}(g(s)+h(s))}\right) \quad (4)$$

which is the minimum between $\varepsilon$ and the ratio between the best solution found so far and the best uninflated $f$ value in the nodes that have yet to be expanded ($OPEN \cup INCONS$) (Hansen and Zhou 2007; Zhou and Hansen 2002).

For non-holonomic motion planning in $(x,y,\theta)$ we have found that in most scenarios the initial solution found is within 10% of the optimal solution. However, there are cases where the difference can be arbitrarily large. Figure 7 shows one scenario in which the solution found by ARAE* is significantly better than the one found by AE*. Because ARAE* is not limited to the equivalence classes it is able to find a solution that could only be found using a much smaller equivalence class. Figure 8 shows the difference in cost between both solutions.

## Case Study: Non-holonomic Global Planning in Large Outdoor Environments

One of the most relevant applications of search-based planning using equivalence classes is long range non-holonomic global planning. Typically, global planning for large, outdoor environments is performed using a 2D planner that is unable to model the kinematic constraints of the vehicle. This planner is usually coupled with a local planner that does model the kinematic constraints of the vehicle and that ensures that the paths the vehicle drives are safe. While this approach performs well in many scenarios, it often fails in complex terrain due to the large disparity between the local and global planners. In order to improve autonomous navigation in outdoor environments, it is therefore important to have more complex global planners that are able to model more of the kinematic constraints of the vehicle while still being fast enough to be updated regularly.

Lattice-based planners are some of the few existing approaches that are able to plan long feasible global routes in a timely manner. Their main drawback is that they need to convert the desired action set into an action set that always starts and ends on a node in the grid. This requirement increases the design complexity of the planners and artificially restricts the action set in ways that introduce artifacts in the resulting paths.

ARAE* is a promising alternative since it is not limited by a grid. The action set selection becomes much simpler, and no artifacts are introduced by making the actions terminate in grid nodes.

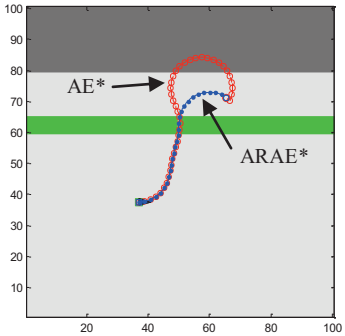


Figure 7. Left: comparison between A* with equivalence classes (circles) and ARA* with equivalence classes (dots) for a more extreme scenario.

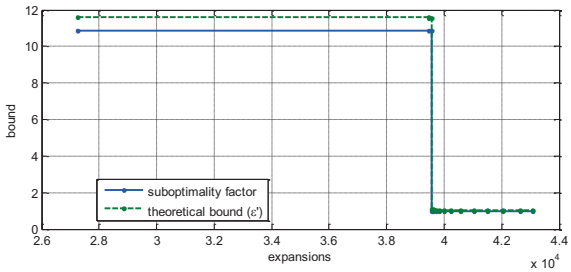


Figure 8. ARAE* showing the error bound $\varepsilon'$ (dashed line) and the actual suboptimality factor (solid line). The first solution found by ARAE* is the same solution found by AE*. The final solution is better by a large margin.

## Experimental Setup

In order to evaluate the suitability of this approach for long-range planning in outdoor environments we performed 400 simulations in simulated environments, assuming a forward moving robot with minimum turning radius of 10 m. The following describes the experimental setup and its results.

### Action Set

The action set used was the one described by equation (3), with $\rho_{\min}$ =10m, $v$ =1 and $\Delta t$ = 3.14, plus a straight segment with $v$ =1 and $\Delta t$ = 1. The size of the equivalence class was set to the maximum allowed by this action set, $\delta_{xy}$ = 1 m and $\delta_{\theta}$ = 18 degrees. The action set is the same for all headings, as can be seen in Figure 9.

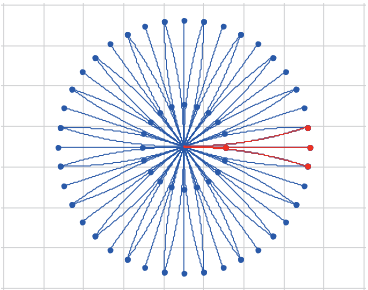

Figure 9. Action set used for experiment (red) and rotated versions for all possible angles given the action set (blue)

### Simulated Worlds

We created fractal worlds that resemble outdoor environments with sizes from 100x100 to 1000x1000 meters, at 1 m resolution. Ten different random seeds were used for each world size. For each world, we ran four experiments, each with a different initial heading for the robot. Figure 10 shows one of the simulated worlds with the paths found at different initial headings plus the 2D path found by A* on the grid (for reference). The non-holonomic paths for the selected headings (-90 and 180 degrees) don't match the 2D path for more than 300 meters, and would have caused a significant mismatch between a 2D global planner and the local planner.

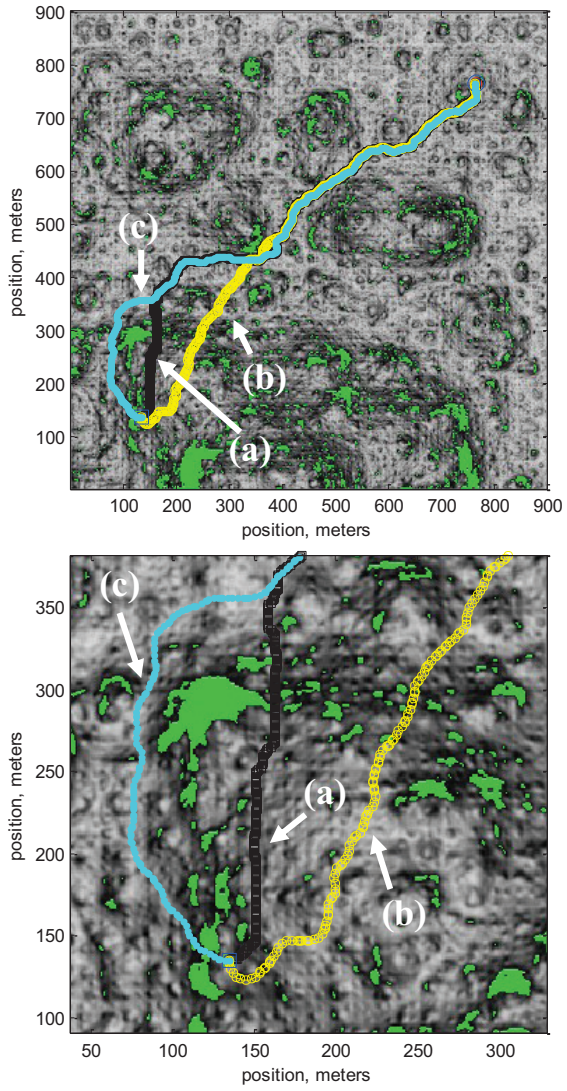


Figure 10. Top: Fractal world used to test long-range planning. Black squares (a) are 2D path (holonomic), yellow circles (b) and cyan dots (c) are the non-holonomic 3-D paths found for initial headings of -90 and 180 degrees respectively. Bottom: detail of previous figure near start location. Notice how different the holonomic and non-holonomic paths are for the selected headings.

### Heuristic

We used the cost-to-goal calculated by a 2-D grid search as the heuristic for the 3-D planner. This heuristic provides an improvement in performance of at least two orders of magnitude compared to the Euclidean distance, independently of which 3-D planner is used.

For our planner, the choice of heuristic also influences the space requirements and the bounds reported. States in the planner are allocated when they are generated, and as such the space requirements of the planner vary greatly with the number of states generated. At least two orders of magnitude fewer states are generated when using the 2-D cost heuristic compared to the Euclidean distance.

The bounds reported by our planner are affected by the heuristic in two ways. The bound from equation (4) is the minimum between $\varepsilon$ and the ratio between the best solution found so far and the best uninflated $f$ value in the nodes that have yet to be expanded. Since a more informed heuristic has a higher $f$ value for these nodes, the bound found is a tighter one. Furthermore, since a more informed heuristic expands fewer nodes, it is possible to decrease further the value of $\varepsilon$ for a given time or space allowance.

### Results

The following figures summarize the results obtained after running the simulations for the different worlds and initial headings. The computing hardware used was an Intel Core2 Duo CPU 2.5GHz, with 4GB of memory, without parallelization or hyperthreading for the planner.

The planner was allowed up to 2.5 seconds to refine a solution, allowing extra time if a solution was not found by that time. Figure 11 (top) shows the planning time until the *first* solution was found, and the error bounds for that solution. Planning times were on average much smaller than one second for worlds up to 600x600, and about one second for worlds up to 1000x1000 (less than 2 seconds 95% of the time). The error bounds for smaller worlds are on average less than 10% (1.10), but can be as high as 40% (1.40). For larger worlds the initial error bound is on average less than 2% (1.02), and it is less than 10% (1.10) 95% of the time.

Figure 11 (bottom) shows the planning time until the *last* solution allowed within the allocated time. Planning times are on average about 1 second, and less than 2.5 seconds 95% of the time. The error bounds have been significantly reduced for the smaller worlds, with the average errors bound at about 2% (1.02) and the 95% confidence interval around 5% (1.05). Planning times do not include the time required to calculate the heuristic for the first time, as this time is a one-time cost at the beginning of a mission and the heuristic can be quickly repaired during the mission using D*Lite (Koenig and Likhachev 2002).

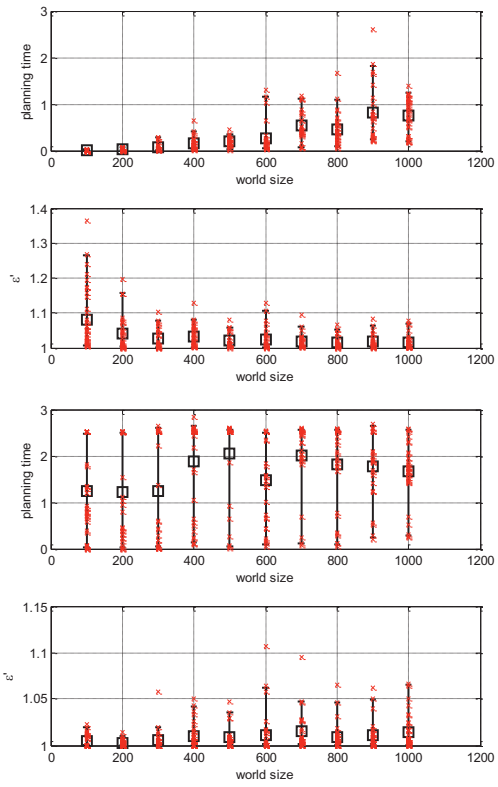


Figure 11. Planning time and error bound for the first solution found(top two images) and last solution found (bottom two images). Squares indicate mean value, with bars for 95% confidence intervals. "x" indicate the actual results for each run.

## Conclusions and Future Work

We have presented a novel approach to search-based planning that improves upon grid-based planners by using equivalence classes to find an initial solution, but instead of pruning states that fall within an equivalence class and have higher cost, it uses an inflated heuristic to lower the priority of these states in the search. In further iterations, the algorithm uses ARA* to reduce the inflated heuristic in a principled way. This allows us to provide fast solutions with provable suboptimality bounds that can be improved as time allows. The proposed approach produces smooth paths with the resolution dictated by the action set. Finer action sets produce higher resolution paths that are more computationally intensive to calculate and coarser action sets produce lower resolution paths that are faster to compute. To the best of our knowledge, this is the first algorithm that is able to plan in continuous state-spaces with provable guarantees on suboptimality for a given action set

The experimental results show that at least for long range non-holonomic path planning this approach is promising. It produces high quality feasible paths for worlds up to 1000x1000 meters in less than 2 seconds. The error bound on these paths is well under 5%, and it is often as low as 1 or 2%. We still have to perform field experiments to evaluate how planning times and error bounds are affected by position errors and noisy sensor data, and to evaluate the impact of the non-holonomic global planner on mission performance.

We would like to explore the applicability of this approach to other domains as well. Since the state space is not explicitly instantiated, this approach may be useful for higher dimensional planning when only a few of the dimensions are relevant

## Acknowledgements

This work was supported by the U.S. Army Research Laboratory under the Robotics Collaborative Technology Alliance program, Cooperative Agreement W911NF-10-2-0016. The views and conclusions contained in this document do not represent the official policies or endorsements of the U.S. Government.

## References

Barraquand, J. & Latombe, J. 1993. Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles. *Algorithmica, 10* (2-4) , 121-155.

Gonzalez, J. & Stentz, A. 2009. Using linear landmarks for path planning with uncertainty in outdoor environments. *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on,* , 1203 -1210.

Hansen, E. A. & Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR), 28* , 267-297.

Karaman, S. & Frazzoli, E. 2010. Incremental Sampling-based Algorithms for Optimal Motion Planning. *Proceedings of Robotics: Science and Systems.*

Kavraki, L. E.; Svestka, P.; Kavraki, L. E.; Latombe, J. & Overmars, M. H. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation, 12* , 566-580.

Koenig, S. & Likhachev, M. 2002. D*lite. *Eighteenth national conference on Artificial intelligence, American Association for Artificial Intelligence*, 476-483.

LaValle, S. 1998. Rapidly-exploring random trees: A new tool for path planning. *TR 98-11, Computer Science Dept., Iowa State University.*

Likhachev, M. & Ferguson, D. 2009. Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *Int. J. Rob. Res., Sage Publications, Inc., 28* , 933-945.

Likhachev, M.; Gordon, G. & Thrun, S. 2003. ARA*: Formal Analysis. *Tech. Rep. CMU-. CS-03-148, Carnegie Mellon University, Pittsburgh, PA.*

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

Pivtoraiko, M. & Kelly, A. 2005. Generating Near-Minimal Spanning Control Sets for Constrained Motion Planning in Discrete State Spaces. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems,* 3231-3237.

Zhou, R. & Hansen, E. A. 2002. Multiple sequence alignment using anytime A*. *Eighteenth national conference on Artificial intelligence, American Association for Artificial Intelligence,* 975-976.