Planning in Domains with Cost Function Dependent Actions

Mike Phillips and **Maxim Likhachev**
mlphilli@andrew.cmu.edu, maxim@cs.cmu.edu
Carnegie Mellon University

Content Area
Constraints, Satisfiability, and Search::Heuristic Search
Robotics::Motion and Path Planning
Reasoning about Plans, Processes and Actions::Planning Algorithms

Abstract

In a number of graph search-based planning problems, the value of the cost function that is being minimized also affects the set of possible actions at some or all the states in the graph. For example, in path planning for a robot with a limited battery power, a common cost function is energy consumption, whereas the level of remaining energy affects the navigational capabilities of the robot. Similarly, in path planning for a robot navigating dynamic environments, a total traversal time is a common cost function whereas the timestep affects whether a particular transition is valid. In such planning problems, the cost function typically becomes one of the state variables thereby increasing the dimensionality of the planning problem, and consequently the size of the graph that represents the problem. In this paper, we show how to avoid this increase in the dimensionality for the planning problems whenever the availability of the actions is monotonically non-increasing with the increase in the cost function. We present three variants of A* search for dealing with such planning problems: a provably optimal version, a suboptimal version that scales to larger problems while maintaining a bound on suboptimality, and finally a version that relaxes our assumption on the relationship between the cost function and action space. Our experimental analysis on several domains shows that the presented algorithms achieve up to several orders of magnitude speed up over the alternative approaches to planning.

Introduction

In certain planning problems, the resource whose usage is being minimized also affects the availability of actions. Moreover, it affects it in such a way that the smaller resource level leads to the same or smaller set of available actions. For example, a robot with a limited battery is able to perform fewer and fewer actions, especially the actions that involve moving uphill, as its battery level approaches zero. Consequently, minimizing energy consumption is a common cost function in planning for mobile robots. Another example is planning for a downhill racecar that runs on its initial potential energy but loses it to friction as it moves. Yet another example is planning for unpowered gliders. Minimizing energy loss allows the glider to stay in the air longer and go farther distances.

All of these planning problems possess an important property that with every action, the resource, and consequently the set of available actions, is monotonically non-increasing. In planning problems represented as a graph search, this property enables us to remove the value of the remaining resource from the variables defining each state in the search-space without affecting the completeness and optimality of the planner (Tompkins 2005). This can be done because the cost function that is being minimized is equal to the value of the variable that is being removed, and when planning with optimal graph search such as A* search, we are already keeping all and only states with the smallest cost function. There is no need to also compute the states whose cost function, and consequently the remaining level of the resource, is suboptimal. This however, is *not* true for planning with suboptimal graph searches such as weighted A*, which are important tools for achieving real-time performance and scalability to large problems. Unfortunately, dropping the state variable that represents the objective function leads to the loss of guarantees on completeness and sub-optimality.

In this paper, we develop an optimal search called Cost Function Dependent Actions A* (CFDA-A*) and then its suboptimal version, called Weighted CFDA-A*, that does allow us to drop the state variable that represents the cost function without sacrificing completeness or sub-optimality bounds. Furthermore, we extend the algorithm to the problems in which our assumption on the relationship between the cost function and action space may sometimes be violated. The theoretical analysis shows that all three variants of CFDA-A* possess all the properties normally expected of A* and weighted A*. Our experimental analysis on two different domains shows that CFDA-A* and its variants can be over 1000 times faster than searching the original state-space with A* or its weighted variant.

Related Work

Somewhat related to our work is the existing work on planning in domains with state dominance relations (Fujino and Fujiwara 1992; Bhattacharya, Roy, and Bhattacharya 1998; Yu and Wah 1988). A state s dominates another s' if knowing that s is reachable guarantees that s' cannot be in the

optimal solution. In other words, the cost of an optimal solution that passes through s must be guaranteed to be less than an optimal solution that passes through s'. This concept allows dominated states to be pruned and reduces the number of states to be examined. Most of the work on planning with state dominance relations however was done *not* in the context of planning with heuristic search-based planning.

Within the area of planning with heuristic searches, there were also several graph searches developed that exploited the dominance relations to gain speedups (Tompkins 2005; Korsah, Stentz, and Dias 2007; Gonzalez and Stentz 2005; Xu and Yue 2009). For instance, similarly to our algorithm, the ISE framework (Tompkins 2005) reduces the dimensionality of a graph search problem by removing the cost function from the set of state variables under certain conditions. DD* Lite (Korsah, Stentz, and Dias 2007) on the other hand does not remove any dimensions but prunes dominated states as it discovers them during the search. The other algorithms (Gonzalez and Stentz 2005; Xu and Yue 2009) exploit the state dominance relations to speedup a heuristic search in the context of specific planning domains. To the best of our knowledge however, none of these algorithms are addressing the use of suboptimal heuristic searches such as weighted A*, which are highly beneficial in scaling to large planning problems and/or achieving real-time performance (Gaschnig 1979; Zhou and Hansen 2002; Likhachev, Gordon, and Thrun 2003; Bonet and Geffner 2001). Filling this gap is the main contribution of our paper.

Notations and Assumptions

We assume the planning problem is represented as searching a directed graph $G = (S, E)$ where S is the state space, or set of vertices, and E is the set of directed edges in the graph. We use $A(s)\ \forall s \in S$ to denote the set of actions available at state s. The function $succ(s, a)$ takes a state s and action a and returns the resulting state. In other words, $(s, succ(s, a))$ corresponds to an edge in E. We assume strictly positive costs $c(s, s') > 0$, for every $(s, s') \in E$. The objective of the search is to find a least-cost path from state s_{start} to state s_{goal} (or set of goal states). For any $s, s' \in S$, we define $c^*(s, s')$ to be cost of a least-cost path from state s to state s'.

We will denote x_d as the state variable that corresponds to the cost function, and x_i as the set of all other state variables. To model the fact that one of the state variables is the cost function, we define the set of state variables x as $x = \{x_i, x_d\}$, where $x_d = c^*(s_{start}, s)$. The main assumption we are making is that for any given x_i, the set of actions is monotonically non-increasing as the cost increases. Formally, for any $s, s' \in S$ such that $x_i(s) = x_i(s')$, if $x_d(s) < x_d(s')$, then $A(s') \subseteq A(s)$ (**assumption 1**).

An example of when this assumption holds would be planning for a robot with a limited battery. The cost function is the amount of energy consumed and as the battery is used up there are fewer actions the robot can perform. For instance, there may be steeper parts of the terrain that can only be climbed if the robot has enough charge left.

```
1  g(ŝ_start) = 0; OPEN = ∅;
2  insert ŝ_start into OPEN with f(ŝ_start) = h(ŝ_start);
3  while(ŝ_goal is not expanded)
4    remove ŝ with the smallest f-value from OPEN;
5    s = {x_i(ŝ), g(ŝ)};
6    for each a in A(s)
7      s' = succ(s, a)
8      ŝ' = {x_i(s')}
9      if ŝ' was not visited before then
10       f(ŝ') = g(ŝ') = ∞;
11     if g(ŝ') > g(ŝ) + c(s, s')
12       g(ŝ') = g(ŝ) + c(s, s');
13       f(ŝ') = g(ŝ') + h(ŝ');
14       insert ŝ' into OPEN with f(ŝ');
```

Figure 1: Optimal CFDA-A*

Optimal CFDA-A*

In A*, discovered states s are put into a priority queue called the *OPEN* list and are sorted by the function $f(s) = g(s) + h(s)$, where $g(s)$ is the cost function representing the best path found so far from the start state to s, and $h(s)$ is a heuristic estimate of the cost from s to the goal. The heuristic must not overestimate the actual cost to the goal and must satisfy the triangle inequality. Initially, the start state is put into *OPEN*. On each iteration, the state s with lowest $f(s)$ is removed from *OPEN* and gets *expanded*. During expansion, A* iterates over the successors of s. If any of these states have not been discovered previously or if a cheaper path to them was found, they are put/re-inserted into *OPEN* with their updated f-values. Every state expanded under A* has the optimal (minimum) g-value. Therefore, when the goal is expanded, A* finds the optimal solution. In optimal A* a state cannot be re-expanded since a shorter path to a state cannot be found after the state has been expanded. Therefore, it is not necessary to maintain a list of states that have already been expanded (often called the *CLOSED* list).

An optimal version of CFDA-A*, shown in Figure 1, is basically an A* search except that it is executed on a new graph $\hat{G}$ which has a reduced state space $\hat{S}$. In $\hat{S}$, the dimension x_d is removed, and therefore each state $\hat{s} \in \hat{S}$ is only defined by x_i. In other words, the search maps each state $s = \{x_i, x_d\}$ in the original state space S onto a state $\hat{s} = \{x_i\}$ in the new state space $\hat{S}$ (line 8) and maps each state $\hat{s} \in \hat{S}$ onto $s \in S$ with the smallest g-value that is reachable during the graph search (line 5). The transitions between $\hat{s}$ and $\hat{s}'$ are defined as the transitions between the corresponding states s and s' in the original state space (line 7).

Theorem 1 *The algorithm terminates, and when it does, the found path from $\hat{s}_{start}$ to $\hat{s}_{goal}$ corresponds to the least-cost path from s_{start} to s_{goal} in S*

Proof Sketch: The path $\hat{p} \in \hat{G}$ from $\hat{s}_{start}$ to $\hat{s}_{goal}$ corresponds to the path $p \in G$ from s_{start} to s_{goal} by taking each state $\hat{s}_k = \{x_i\} \in \hat{p}$ and making it $s_k = \{x_i, g(\hat{s})\} \in p$. Assume $\hat{p}$ does not correspond to a least-cost path p^*. Then p^* must contain a state s that CFDA-A* did not consider or else it would have found the same path. However, the only states from S that CFDA-A* ignores are ones that have larger (less optimal) x_d. In other words, there is a cheaper path to $x_i(s)$,

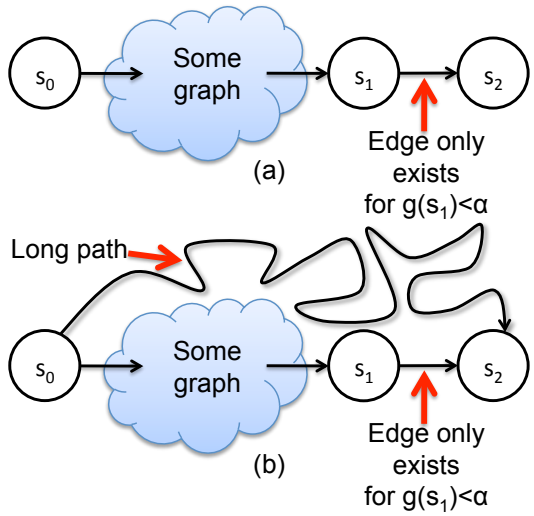

Figure 2: (a) Running normal weighted A* on this graph may return no solution if s_2 is expanded sub-optimally such that its g-value exceeds α. (b) For a similar reason, the sub-optimality bound can be violated on this graph if the cost of the long path is greater than ϵ times the minimum cost of the lower path.

then the one in p^*. More detailed proofs can be found in our tech report (Phillips and Likhachev 2011).

Weighted CFDA-A* with bounded sub-optimality

Weighted A* is a common generalization of the A* algorithm which puts states s into the *OPEN* list according to $f(s) = g(s) + \epsilon * h(s)$ (Pohl 1970). The scalar $\epsilon > 1$ inflates the heuristic. As ϵ increases, the f function becomes more heavily driven by the heuristic component and will often find the solution by expanding far fewer states. While the solution is no longer guaranteed to be optimal, its cost is guaranteed to be no more than ϵ times the cost of an optimal solution (Pohl 1970). While normal weighted A* allows a state to be expanded more than once (whenever a shorter path to the state is found), it turns out that the same guarantees can be provided even if we allow each state to be expanded only once (Likhachev, Gordon, and Thrun 2003). The limit helps in assuring the run-time complexity, since without it, the number of state expansions can be exponential in the number of states.

Running CFDA-A*, as described in the previous section, but with the inflated heuristics, destroys its guarantees on completeness and solution quality. Figure 2(a) demonstrates this problem. The issue occurs when a state s_1 is expanded sub-optimally (due to the inflated heuristic) and its sub-optimal g-value leads to the absence of a critical action between s_1 and s_2. Figure 2(b) shows how the ϵ bound on the solution quality may also get ignored. The example shows that if there is more than one path to the goal, and the lower-cost path is not found due to sub-optimal expansions, then the algorithm will find a longer path to the goal and terminate, independently of how much costlier it is.

Weighted CFDA-A* shown in Figure 3 solves this problem by introducing two versions of each state $\hat{s}$, an optimal version and a sub-optimal version. To achieve this, we redefine $\hat{s}$ as $\hat{s} = \{x_i, o\}$, where o is a boolean variable representing the optimality of $\hat{s}$. The invariant that Weighted CFDA-A* maintains is that when expanding any state $\hat{s}$ whose $o(\hat{s}) = true$, the algorithm guarantees that its $g(\hat{s})$ is optimal. Similarly, when expanding any state $\hat{s}$ whose $o(\hat{s}) = false$, the algorithm guarantees that its $g(\hat{s})$ is no more than ϵ times the optimal g-value.

When expanding, optimal states generate both optimal and sub-optimal successors, but sub-optimal states generate only sub-optimal successors. Sub-optimal states go into the *OPEN* list with the usual $f = g + \epsilon * h$, but optimal states go in with an inflated $f = \epsilon * (g + h)$. The following can be said about this ordering. First, sub-optimal states tend to come out earlier than optimal ones because optimal states have both terms of the f-value inflated instead of inflating just the heuristics for sub-optimal states. This is beneficial for runtime because a sub-optimal search usually finds a path to goal much faster due to the drive of the inflated heuristic. Second, the optimal states, with respect to each other, are expanded in the exact same order as they would be in the optimal A* search. The only difference is that their f-values are just scaled by a positive constant, but this has no effect on the relative ordering of the optimal states. This property restores the completeness guarantee to our algorithm: even if the sub-optimal states get blocked from the goal such as in Figure 2(a), the optimal states will expand in exactly the order they would in the optimal algorithm and will therefore get past this trap. As soon as an optimal state is expanded at the blocking point, it will generate optimal and sub-optimal successors past the block, and the sub-optimal states can again move quickly towards the goal. For the same reasons, this approach also restores the bound on sub-optimality, as stated in Theorem 2.

Figure 3 shows the weighted CFDA-A* variant. There are a few differences from the pseudocode of optimal CFDA-A* explained previously. The addition of a *CLOSED* list on lines 1,4,13 ensures that a state can only be expanded at most once. On lines 6, 9, and 10 we see that if an optimal state is being expanded, we generate both optimal and suboptimal versions of each successor. But if a suboptimal state is being expanded, then only suboptimal successors are generated. Also, lines 15-18 now reflect how the f-value can be set in one of two ways depending on whether the successor is optimal or not.

Theorem 2 *The algorithm terminates, and when it does, the found path from $\hat{s}_{start}$ to $\hat{s}_{goal}$ corresponds to a path from s_{start} to s_{goal} that has a cost no greater than $\epsilon * c^*(s_{start}, s_{goal})$.*

Proof: If we can show that the path from $\hat{s}_{start}$ to $\hat{s}_{goal}$ has a cost no greater than $\epsilon * c^*(\hat{s}_{start}, \hat{s}_{goal})$, then we can apply Theorem 1 to prove the claim. Let $c^*(\hat{s}_{start}, \hat{s}) = g^*(s)$. Assume for sake of contradiction that upon expansion of $\hat{s}_{goal}$, $g(\hat{s}_{goal})$ is greater than ϵ times the optimal g-value ($g(\hat{s}_{goal}) > \epsilon * g^*(\hat{s}_{goal})$). Then $\frac{g(\hat{s}_{goal})}{\epsilon} > g^*(\hat{s}_{goal})$ be-

```
1  g(ŝ_start) = 0; O(ŝ_start) = true; OPEN = ∅; CLOSED = ∅;
2  insert ŝ_start into OPEN with f(ŝ_start) = ϵ * h(ŝ_start);
3  while(ŝ_goal is not expanded)
4    remove ŝ with the smallest f-value from OPEN and insert ŝ in CLOSED;
5    s = {x_i(ŝ), g(ŝ)};
6    If O(ŝ) = true then Opt = {true, false}, else Opt = {false}
7    for each a in A(s)
8      s' = succ(s, a)
9      for o in Opt
10       ŝ' = {x_i(s'), o}
11       if ŝ' was not visited before then
12         f(ŝ') = g(ŝ') = ∞;
13       if g(ŝ') > g(ŝ) + c(s, s') and s' ∉ CLOSED
14         g(ŝ') = g(ŝ) + c(s, s');
15         if O(ŝ')
16           f(ŝ') = ϵ * (g(ŝ') + h(ŝ'));
17         else
18           f(ŝ') = g(ŝ') + ϵ * h(ŝ');
19         insert ŝ' into OPEN with f(ŝ');
```

Figure 3: Weighted CFDA-A*

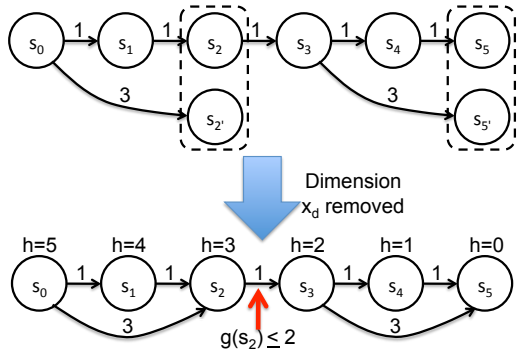

Figure 4: The original graph (top) is reduced (bottom) by removing dimension x_d. In this example, running the normal weighted A* on the reduced graph will return no solution because of the edge that depends on the g-value. Weighted CFDA-A* resolves this problem.

cause ϵ is positive.

Now any state $\hat{s}$ on the optimal path satisfies $g^*(\hat{s}) + h(\hat{s}) \leq g^*(\hat{s}_{goal})$. By substitution, $g^*(\hat{s}) + h(\hat{s}) \leq \frac{g(\hat{s}_{goal})}{\epsilon}$. Therefore, $\epsilon * (g^*(\hat{s}) + h(\hat{s})) \leq g(\hat{s}_{goal})$. This $\epsilon * (g + h)$ is how we sort optimal states in the *OPEN* list. Since $g^*(\hat{s}_{start})$ is the first state in the *OPEN* list, by induction, $g^*(\hat{s}_{goal})$ would be discovered and expanded before $g(\hat{s}_{goal})$. Therefore, when the goal state is expanded, $g(\hat{s}_{goal})$ would be optimal, which is a contradiction. This means that the optimal goal state would have to be expanded before a suboptimal state that is not bounded by ϵ. ■

The following theorem relates the complexity of Weighted CFDA-A*, in particular, that it reduces the dimension x_d to a binary value, since for every x_i, we now have only two states: an optimal state and a suboptimal state.

Theorem 3 *For every collection of states with the same x_i in S, Weighted CFDA-A* performs at most two expansions.*

Example

Using the example in Figure 4, we demonstrate how our algorithm avoids the pitfalls shown in Figure 2 using optimal states. We assume that $\epsilon = 3.0$. The heuristic function is just the number of edges to the goal. The top graph represents the original graph. The states grouped in dotted boxes have the same x_i but different x_d. In the bottom graph, this dimension has been removed. Thus, s_2 and $s_{2'}$ have become the same state, as have s_5 and $s_{5'}$. However, this has caused a cost dependent edge between s_2 and s_3.

The start state is s_0 and the goal is s_5. Weighted CFDA-A* first expands s_0^{opt}, s_2^{sub}, and s_1^{sub}, where the superscript ^{opt} denotes an optimal state and ^{sub} denotes a suboptimal state. Since, the g-value of s_2^{sub} at time of expansion is 3, it does not generate any successors. The state s_1^{sub} also generates no successors because s_2^{sub} has already been expanded. If we were running regular weighted A*, the search would have ended here and declared that there was no solution. However, in weighted CFDA-A*, s_1^{opt}, and s_2^{opt} are still in the *OPEN* list. s_1^{opt} would be expanded first, updating the g-value of s_2^{opt} to 2. Then, when s_2^{opt} is expanded, it generates both optimal and sub-optimal versions of state s_3. Now, the lower-priority sub-optimal states will again be expanded starting with s_3^{sub}, followed by the goal state s_5^{sub}.

Weighted CFDA-A* with Proper Intervals

The final extension to our algorithm allows it to work on the problems where assumption 1 may sometimes be violated. We assumed that for any pair of states $s = \{x_i, x_d\}$, $s' = \{x_i', x_d'\}$, if $x_i = x_i'$ and $x_d < x_d'$, then $A(s') \subseteq A(s)$ in G, and since s' has a worse g-value, we can replace both of these states with $\hat{s}$ in $\hat{G}$. This is not true if for some x_i the valid values of x_d are not one contiguous interval, because they are separated by invalid states in G. In these cases, the broken assumption leads to incompleteness. In Figure 5(b), s_1 represents the state with the smallest g-value in x_i that is reachable. When expanding this state, the action that moves to x_i' only generates s_2 but not s_3. In some scenarios the later interval may be the only way to the goal. Not generating this successor causes incompleteness. However, if the planning problem includes a *spend* action that increments x_d without changing x_i, then CFDA-A* regains its guarantees. In Figure 5(b), this allows s_1 to reach a higher (downward in the figure) g-value before executing the action so that it generates s_2 and s_3.

We now define a proper interval as a maximal contiguous series of valid values of x_d for a particular x_i. In Figure 5, x_i' has two proper intervals. With the addition of a spend operation that allows positive movement along x_d, the only state in a proper interval that matters is the one with the smallest g-value. We can therefore augment states for $\hat{G}$ with a variable that identifies the proper interval for the x_i. More formally, a state is now defined as $\hat{s} = \{x_i, p, o\}$, where p is the proper interval index. In Figure 5(c), the intervals have been converted to a graph. There are two states for x_i' with interval index p being 0 and 1. State s_1 has an action that goes to s_2 if $g(s_1) < 4$ and the same action always goes to s_3, but sometimes requires a spend operation to get it past the collision period ($g(s_1) < 6$). With the intervals and spend operation we once again have the guarantee that the set of successors is monotonically non-increasing as the

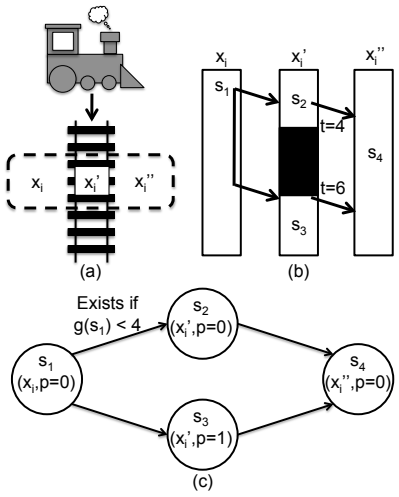

Figure 5: (a) A scenario with three locations (x_i, x_i', x_i'') and a train which arrives at x_i' at time 4 and leaves at time 6. (b) The intervals shown for the three locations. Since x_i and x_i'' are free for all time, they each have one interval (therefore one state). Location x_i' has a safe interval before the train passes ($t < 4$) and after ($t > 6$) resulting in two states for this location (s_2, s_3). (c) The intervals converted into a graph with a cost function dependent action. The edge (s_1, s_2) only exists if $g(s_1) < 4$. The edge going to s_3 always exists due to the spend operation.

cost increases.

Figure 6 shows the algorithm for CFDA-A* with proper intervals. On line 9 we see that after getting the successor state $s' \in S$ we iterate over the proper intervals for that x_i. After forming $\hat{s}'$ for this interval, on line 14 we compute c_{spend} which is how much the g-value needs to be incremented before executing action a in order to arrive at interval p at the smallest g-value. This value may not exist indicated on lines 15-16 either because the interval we are trying to get to ends before $g(\hat{s}) + c(s, s')$ (we already missed it) or the smallest g-value we could arrive at interval p would require incrementing $g(\hat{s})$ past the end of its proper interval before executing the action. If such a c_{spend} does exist, then on lines 17-18, it is used as an additional cost in the g-value update step.

Theorem 4 *The algorithm terminates, and when it does, the found path from $\hat{s}_{start}$ to $\hat{s}_{goal}$ corresponds to a path from s_{start} to s_{goal} that has a cost no greater than $\epsilon * c^*(s_{start}, s_{goal})$.*

Proof Sketch: We can apply Theorem 2 if **assumption 1** applies. For some $s, s' \in S$, where $x_i = x_i'$ and $x_d < x_d'$, s does not have a superset of the actions of s' because an action a could lead to an invalid state for s but a valid one for s' due to invalid intervals along minimized dimension. This is remedied by adding a proper interval index as part of the state and a spend operation so that the state s can have all the successors of s' by spending until s' before executing a.

```
1  g(ŝ_start) = 0; O(ŝ_start) = true; OPEN = ∅; CLOSED = ∅;
2  insert ŝ_start into OPEN with f(ŝ_start) = ε * h(ŝ_start);
3  while(ŝ_goal is not expanded)
4    remove ŝ with the smallest f-value from OPEN and insert ŝ in CLOSED;
5    s = {x_i(ŝ), g(ŝ)};
6    If O(ŝ) = true then Opt = {true, false}, else Opt = {false}
7    for each a in A(s)
8      s' = succ(s, a)
9      for each proper interval p in x_i(s')
10       for o in Opt
11         ŝ' = {x_i(s'), p, o}
12         if ŝ' was not visited before then
13           f(ŝ') = g(ŝ') = ∞;
14         c_spend = extra increment before a to arrive in p with smallest g-value;
15         if c_spend does not exist
16           continue;
17         if g(ŝ') > g(ŝ) + c(s, s') + c_spend and s' ∉ CLOSED
18           g(ŝ') = g(ŝ) + c(s, s') + c_spend;
19           if O(ŝ')
20             f(ŝ') = ε * (g(ŝ') + h(ŝ'));
21           else
22             f(ŝ') = g(ŝ') + ε * h(ŝ');
23           insert ŝ' into OPEN with f(ŝ');
```

Figure 6: Weighted CFDA-A* with Proper Intervals

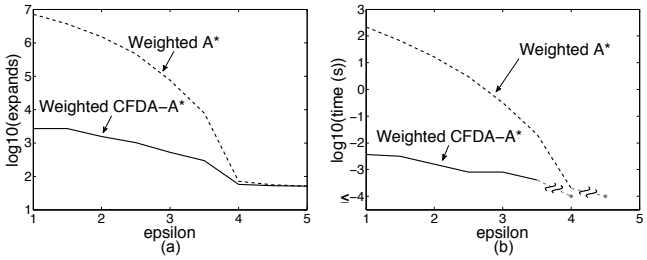

Figure 7: Planning for a limited battery comparing the full state space and our reduced state space over various ϵ. (a) Average expansions (on the log_{10} scale). (b) Average seconds (on the log_{10} scale). When the plot drops below the breaks, the planning time was less than or equal to 0.0001 (≤ -4 on the log_{10} scale) seconds and was too small to measure.

Experimental Analysis

We performed simulation experiments in two domains to show the benefits of using our algorithms.

Robot with a Limited Battery

The domain of the robot with a limited battery exhibits the benefits of weighted CFDA-A* (without the need of proper intervals). The original state space of this domain is x, y, energy consumed. There is an energy consumption limit (when the battery is exhausted). The environments are randomly generated fractal costmaps commonly used to model outdoor environments (Yahja et al. 1998). The robot had to move from the upper left corner to the lower right. The cost of any cell ranges from 1 to 6 . The cost of traversing a cell is the distance traveled (in cells) times the cost of the cell. This cost is assumed to be the energy consumed by that action. The heuristic used was Euclidean distance. We ran two sets of experiments.

The first experiments compare the full state space

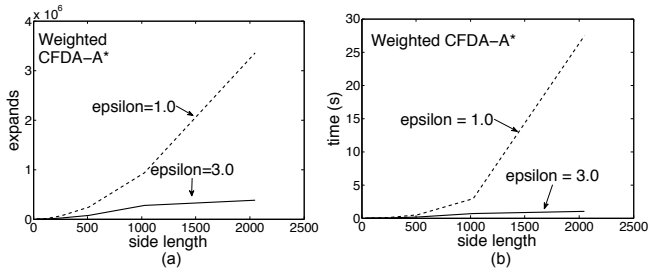

Figure 8: Planning for a limited battery with a reduced state space comparing $\epsilon = 1.0$ and $\epsilon = 3.0$ over various map size. (a) Average expansions (b) Average seconds

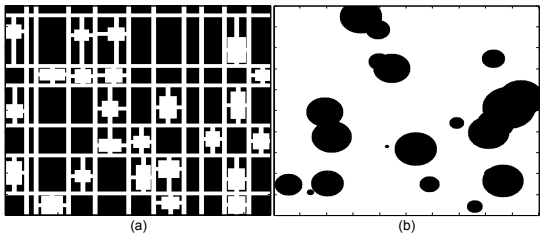

Figure 9: Examples of random (a) indoor, (b) outdoor maps used for planning in dynamic environments.

weighted A* against our weighted CFDA-A*. We generated 50, 51-by-51 random fractal environments. For much larger environments, the full state space search didn't find solutions most of the time (especially for low ϵ). The graph shown in Figure 7(a) shows the average number of expansions for various values of ϵ on a log_{10} scale. CFDA-A* shows up to 1000 times less expansions for smaller ϵ. After $\epsilon = 4.0$, the two are similar, because the cost function becomes so dominated by the heuristic (particularly because the max cell cost is only 6) that both methods end up expanding a straight line from the start to the goal. Figure 7(b) shows that the actual planning times have roughly the same speed up as the expansions. After $\epsilon = 4.0$ the planninng times become too small to measure.

Our second experiment shows the advantage of using weighted CFDA-A* over optimal CFDA-A*. To test this we compared the weighted CFDA-A* (with $\epsilon = 3.0$) against the optimal on maps of varying size ranging from 33-by-33 to 2049-by-2049. For each map size we had 5 trials (to average out noise). We tried 7 different map sizes so there were 35 trials in total. The graph in Figure 8(a) shows the average number of expansions for various map sizes. This plot shows that having an inflated heuristic produces a significant speed up over optimal planning in spite of the fact that the state space is two times larger (optimal and sub-optimal states) in order to maintain the guarantees. Figure 8(b) shows similar results for the corresponding planning times.

Planning in Dynamic Environments

Planning in dynamic environments is the problem of finding a collision free path from the start location and time to a goal location. There are both static and moving obstacles in the environment and in order to find an optimal path, planning

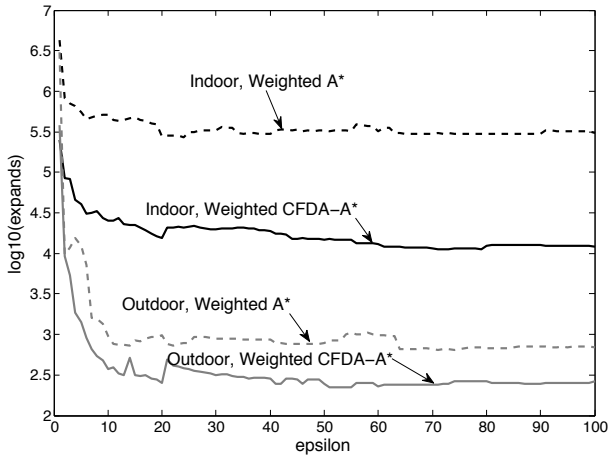

Figure 10: Average expansions (log_{10} scale) for indoor (black) and outdoor (gray) dynamic environments comparing full state space (dashed line) to CFDA-A* (solid line) for varying ϵ.

must be done in the space-time state space. Also, for many robots, the direction they are facing affects where their motions take them so we've incorporated θ into the search space as well. The planning in dynamic environments domain requires proper intervals in order for our weighted A* variant to maintain guarantees. This experiment shows the benefits of using our weighted CFDA-A* with proper intervals. The original state space is x, y, θ, time. The maps we ran on were 500-by-500 with θ discretized into 16 angles and time discretized into 0.1 seconds. We assumed we knew the trajectories of the dynamic obstacles in advance. For each map, there were 200 dynamic obstacles with randomly generated time parameterized paths to follow. We ran two types of experiments, on indoor maps and outdoor maps. The indoor maps were comprised of randomly generated narrow hallways and rooms. The halls were narrow enough that some dynamic obstacles filled the entire width and could not be passed by the robot. An example of this type of environment is shown in Figure 9(a). The outdoor maps were more open with about 20 percent of the map covered with circles of varying size. An example of an outdoor map is shown in Figure 9(b). Backward 2D Dijkstra search that accounted for static obstacles (but no dynamic obstacles) was used as the heuristic.

Our experiments compared a planner on the original full state space against Weighted CFDA-A* with proper intervals. The black curves in Figure 10 show the average expands for the indoor trials across various values for ϵ. The plot shows 10 times fewer expansions not only for optimal but also for the weighted CFDA-A*. It is also important to note that weighted CFDA-A* runs significantly faster than optimal CFDA-A* search. The gray curves in Figure 10 show the average expands for the outdoor trials across various values for ϵ. There is also an improvement here, however it is not as dramatic. This is because in the indoor case the robot must wait for obstacles to pass (because of the tight hallways) very often. For our algorithm, a wait (i.e., spend

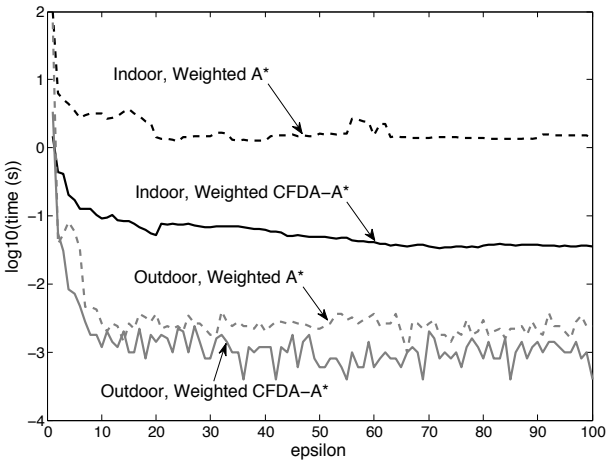

Figure 11: Average seconds (log_{10} scale) for indoor (black) and outdoor (gray) dynamic environments comparing full state space (dashed line) to CFDA-A* (solid line) for varying ϵ.

operation) of arbitrary length is handled intelligently with a single successor, but for the full state space, waiting in place requires many expansions along the time dimension (one for each timestep). In the outdoor case this matters less because the environment is so open it is almost always faster to dodge around dynamic obstacles instead of waiting them out. In Figure 11, the actual planning times show roughly the same speed up as the expansions.

Acknowledgements

This research was partially sponsored by the ONR grant N00014-09-1-1052 and DARPA grant N10AP20011. We also thank Willow Garage for their partial support of this work.

Conclusions

Many domains minimize a cost function that is also a dimension in the search space, because it affects the possible actions. Our algorithm allows for the removal of this dimension from the search space resulting in a graph with Cost Function Dependent Actions. CFDA-A* can solve these types of graphs in the optimal case. Weighted CFDA-A* uses optimal and suboptimal states with a clever expansion order in order to continue to provide this reduction in the size of the state space while maintaining the theoretical properties of completeness and bounded sub-optimality. Finally, our algorithm can be extended to handle more complicated problems that violate our key assumption overall, but still maintain it within proper intervals. This replaces the dimension in question with a more compact representation that works with both optimal and weighted CFDA-A*. We've shown strong theoretical guarantees on all of our algorithms as well as significant experimental results from two domains: planning with a limited battery and planning in dynamic environments. While not addressed directly in our paper, the guarantee of bounded sub-optimality easily allows our method to be extended to anytime planning methods like ARA* (Likhachev, Gordon, and Thrun 2003).

References

Bhattacharya, S.; Roy, R.; and Bhattacharya, S. 1998. An exact depth-first algorithm for the pallet loading problem. *European Journal of Operational Research* 110(3):610–625.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Fujino, T., and Fujiwara, H. 1992. An efficient test generation algorithm based on search state dominance. In *FTCS*, 246–253.

Gaschnig, J. 1979. Performance measurement and analysis of certain search algorithms. Tech. Rep. CMU-CS-79-124, Carnegie Mellon University.

Gonzalez, J. P., and Stentz, A. 2005. Planning with uncertainty in position: An optimal and efficient planner. In *in Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS 05)*, 2435–2442.

Korsah, G. A.; Stentz, A. T.; and Dias, M. B. 2007. Dd* lite: Efficient incremental search with state dominance. Technical Report CMU-RI-TR-07-12, Robotics Institute, Pittsburgh, PA.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press.

Phillips, M., and Likhachev, M. 2011. Planning in domains with cost function dependent actions: Formal analysis.

Pohl, I. 1970. First results on the effect of error in heuristic search. *Machine Intelligence* 5:219–236.

Tompkins, P. 2005. *Mission-Directed Path Planning for Planetary Rover Exploration*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Xu, Y., and Yue, W. 2009. A generalized framework for bdd-based replanning a* search. In *SNPD*, 133–139.

Yahja, A.; Stentz, A.; Singh, S.; and Brumitt, B. L. 1998. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 650–655.

Yu, C. F., and Wah, B. W. 1988. Learning dominance relations in combinatorial search problems. *IEEE Trans. Software Eng.* 14(8):1155–1175.

Zhou, R., and Hansen, E. A. 2002. Multiple sequence alignment using A*. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Student abstract.