Chapter 10
Motion Planning

Part 3
10.3 Real Time Global Motion Planning
Outline

• 10.3 Real Time Global Motion Planning
  – 10.3.1 Introduction
  – 10.3.2 Depth Limited Approaches
  – 10.3.3 Anytime Approaches
  – 10.3.4 Plan Repair Approach: D* Algorithm
  – 10.3.5 Hierarchical Planning
  – Summary
Outline

- 10.3 Real Time Global Motion Planning
  - 10.3.1 Introduction
  - 10.3.2 Depth Limited Approaches
  - 10.3.3 Anytime Approaches
  - 10.3.4 Plan Repair Approach: D* Algorithm
  - 10.3.5 Hierarchical Planning
  - Summary
10.3.1 Introduction

• Unknown and dynamic environments can be treated similarly because a dynamic environment is partially unknown.

• Unknown Environments
  – Limited perception limits what you can know.
  – Often, the only way to learn more is to move.
  – You may eventually learn that the path you are on is wrong.

• Dynamic Environments
  – Limited prediction fidelity limits what you can know.
  – Often, the only way to learn more is to wait.
10.3.1 Introduction

(Thinking vs Doing)

• Often, it is possible to trade off the cost of execution and planning.
  – More planning time makes better use of available information.
  – More motion gathers more information.

• Sometimes its better to stop and think, other times not.
10.3.1.1 Unknown Environments

(Changing Strategy)

- It is not unusual for a robot to continue to change its mind as it learns new information.
10.3.1 Introduction
(Four Techniques)

• Four techniques are available to deal with the real time / limited computation issues:

  1. Limited Horizon
     • Don’t predict too far

  2. Anytime Approaches
     • Always have an answer available

  3. Plan Repair
     • Reuse elements of last plan.

  4. Hierarchical Planning
     • Ignore detail when possible
Outline

• 10.3 Real Time Global Motion Planning
  – 10.3.1 Introduction
  – 10.3.2 Depth Limited Approaches
  – 10.3.3 Anytime Approaches
  – 10.3.4 Plan Repair Approach: D* Algorithm
  – 10.3.5 Hierarchical Planning
  – Summary
10.3.2.1 Purely Reactive Planning

• Search is conducted physically with the robot.
  – Bias toward goal added

• However, the right answer (above) is to move away from the goal for a while.

• Cyclic behavior is a common failure mode.
10.3.2.2 Depth Limited Planning

• Same as receding horizon predictive control.
• Propagate best child up the tree ...
• Then, takes the first step toward the best leaf.
• and repeat.
10.3.2.3 Real Time A* (Korf)

- MiniMin lookahead search:
  - Search forward some fixed depth determined by the available computation.
  - Compute the “backed” up value of each potential first move as the minimum heuristic value of all of its children on the search frontier.
  - Employ the principle of least commitment by making a single move to the best child of the current node.

Equivalent to simply finding the best leaf node and the first move toward it.
Real Time A* (Korf)

- In RTA* we re-interpret $g(X)$ to mean the cost to get \textit{from the current state} to state $X$ rather than the cost from the original initial state - which is irrelevant once motion takes place.
- Net effect is to permit \textit{physical backtracking} to an earlier visited state if the benefit of doing so outweighs the cost.
- This planner and all unknown environment planners are subject to \textit{strategy waffling} (cycles).
Outline

• **10.3 Real Time Global Motion Planning**
  – 10.3.1 Introduction
  – 10.3.2 Depth Limited Approaches
  – 10.3.3 Anytime Approaches - Skip
  – 10.3.4 Plan Repair Approach: D* Algorithm
  – 10.3.5 Hierarchical Planning
  – Summary
Outline

• 10.3 Real Time Global Motion Planning
  – 10.3.1 Introduction
  – 10.3.2 Depth Limited Approaches
  – 10.3.3 Anytime Approaches
  – 10.3.4 Plan Repair Approach: D* Algorithm
  – 10.3.5 Hierarchical Planning
  – Summary
A* Replanning is Still Too Slow
Replanning Done Right D* (Lite)
10.3.4 Plan Repair Approach: D* Algorithm
(Basic Approach)

• Construct an initial solution using A* (or whatever).

• Continuously maintain this solution as....
  – 1: New information arrives
  – 2: The robot moves.
10.3.4 Plan Repair Approach: D* Algorithm (Basic Approach)

- 1: Compute initial path up front.
- 2: Follow path until something new is learned.
- 3: Propagate the changes through search tree.
- 4: Compute new path
- 5: Goto 2:
10.3.4 Plan Repair Approach: D* Algorithm
(Search Graph Vs Tree)

Search Graph

Search Tree

Downward arrows are graph elaboration
point parent → child

Upward arrows are backpointers
point child → parent
10.3.4 Plan Repair Approach: D* Algorithm
(Some Observations)

1. Only the path from here (not from start) to the goal is needed.

2. Discoveries are generally made close to the robot.
10.3.4 Plan Repair Approach: D* Algorithm (Propagating Cost Changes)

- Changes must propagate all the way to all pertinent affected leaves of the search tree
10.3.4 Plan Repair Approach: D* Algorithm

(Conclusions)

• Since changes to a search tree must propagate all the way to the leaves to fully understand their implications ......

• Search from the goal BACKWARD to the robot.
  – Root = Goal
  – Leaf = Robot
10.3.4 Plan Repair Approach: D* Algorithm
(Compute initial Path)

- Use A* from G (goal) to R (robot). Save f() values of every node opened.
- Dstar: Work in terms of \( h() \) and \( k() \) where:
  - \( h() \) is same as \( f() \) in A*
  - \( k() \) is the minimum value \( h() \) has ever had since it was placed on OPEN.
- DstarLite: Work in terms of \( g() \) and \( \text{rhs()} \) where:
  - \( g() \) is same as before
  - \( \text{rhs} \) is best possible \( g \) RIGHT NOW based on all possible neighbors

Now the GOAL is the root of the search tree but I call it S for cleaner code.

This is the search tree spanning tree encoded in backpointers.
10.3.4.1.2 Implications of Edge Changes
(Lowered Cost)

• Suppose an edge $E$ gets cheaper....

• Nodes $W$ and $Y$ may want to abandon their parents in favor of $N$.

This is the search tree $\rightarrow$ spanning tree encoded in backpointers.
10.3.4.1.2 Implications of Edge Changes (Raised Cost)

• Suppose an edge gets costlier....

• Node N may want a different parent.

This is the search tree spanning tree encoded in backpointers.
10.3.4.1.2 Implications of Edge Changes
(Efficient Propagation)

• (Almost) Brute force approach:
  – Go back in time....
  – Remove all nodes from OPEN or CLOSED for which \( f(\text{Node}) > f(P) \).
  – Mark remaining leaves as open
  – Rerun Astar.

• Efficient?
  – Touches every node between P and G in the solution tree.
  – Many end up unchanged from last time.

• Not efficient.

• BUT: Placing affected nodes on OPEN is a good idea.
  – See next figure to visualize.
10.3.4.1.2 Implications of Edge Changes
(D* Processing Wavefront)

Present Highest KEY Value

Optimal Nodes

Changed Nodes Not Known to be Optimal

Nodes Previously on OPEN

Next Node To Check
10.3.4.1.4 Processing Multiple Changes

- Propagate changes **downward in one pass committing** as you go
  - Hence sort changed nodes (perhaps on OPEN?)
- **Lowered** states may need to move up the tree
  - Their sort key is their **new cost** (move up before the slot closes)
- **Raised** states may need to move down the tree
  - Their sort key is their **old cost** (move down before you get stuck)
10.3.4.1.4 Propagating Cost Changes
(Will Rerunning Astar Work?)

• Place N on OPEN and propagate changes downward (reopening closed nodes)

• Does not work.
  – In Astar, nodes on OPEN compete to be the parents of neighboring nodes.
  – The resulting subtree must have N as its root (N is like start).
  – So, every changed node will have a path that goes through N.
  – No mechanism for M to route around N if Edge E increased in cost.
Idea: Propagate Inconsistency

- Node N is inconsistent if it does not point to its “best” parent.
- Remove this node and, if it is not optimal, reinsert in O in correct place.

```java
00: updateVertex(x) {
01:     if ( x != x_start ) getRhs(x);
02:     if ( x ∈ O ) O.remove(x);
03:     if ( g(x) ≠ rhs(x) ) O.insertSorted(x);
04: }
```
Dstar Lite Goodies

• “Right Hand Side”

\[
\text{rhs}(s) = \begin{cases} 
0 & \text{if } s = s_{\text{start}} \\
\min_{s' \in \text{Pred}(s)} (g(s') + c(s', s)) & \text{otherwise.}
\end{cases}
\]

– It’s the **cost a node would** have if one level of lookahead was resolved.

• “Key” (f value)

\[
\text{[min}(g(s), \text{rhs}(s)) + h(s, s_{\text{goal}}); \text{min}(g(s), \text{rhs}(s))] 
\]

– Cost a node will have as soon as its neighbors are told they need to change.

– **Break ties** with second key.
Idea: Propagate Inconsistency

• Check all neighbors of nodes on open.

```cpp
00:   computeShortestPath(x) {
01:     while (f[O.peek()] < f(x/goal) || g(x/goal) ≠ rhs(x/goal))
02:       x_next = O.pop(); Remove from O
03:       if (g(x_next) > rhs(x_next)) { g is too high
04:         g(x_next) = rhs(x_next); Correct it, don’t put it back on O
05:         for (each x_neigh ∈ pred(x_next)) Check all neighbors
06:           updateVertex(x_neigh);
07:       } else {
08:         g(x_next) = ∞ Force it on O with key based on rhs()
09:         for (each x_neigh ∈ pred(x_next) ∪ {x_next}) Check all neighbors
10:           updateVertex(x_neigh);
11:     }
```
Termination

• Terminate when:
  – lowest $f()$ on OPEN > $f(\text{robot})$
  – Robot node is then optimal.

• Often need to compensate for roundoff:
  – lowest $f()$ on OPEN > $f(\text{robot}) + e$
Entire Algorithm

For Sorting OPEN

Initialize

Perception Info

Plan Paths

Initial A* - Like call

Move, Percieve, Replan
Beware

• Code switches GOAL and START for Dstar only.
• That means they are switched relative to the DstarLite paper.
Random Observations

- Runtime is not constant

- The alternative is worse.
Outline

• **10.3 Real Time Global Motion Planning**
  – 10.3.1 Introduction
  – 10.3.2 Depth Limited Approaches
  – 10.3.3 Anytime Approaches
  – 10.3.4 Plan Repair Approach: D* Algorithm
  – 10.3.5 Hierarchical Planning - **Skip**
  – Summary
Outline

• 10.3 Real Time Global Motion Planning
  – 10.3.1 Introduction
  – 10.3.2 Depth Limited Approaches
  – 10.3.3 Anytime Approaches
  – 10.3.4 Plan Repair Approach: D* Algorithm
  – 10.3.5 Hierarchical Planning - Skip
  – Summary
Summary

• The real motion planning problem is that of planning in dynamic and uncertain environments.
  – Maps are never completely accurate.

• Computational techniques are mature in the abstract case of grids.

• The real motion planning problems therefore become:
  – Understanding mobility
  – Adequate perception

Mobile Robotics - Prof Alonzo Kelly, CMU RI