D*-Lite
Path Finding Algorithm and its Variations

Can Eroğul
erogul@ceng.metu.edu.tr
Outline

- Motivation
- Environment Properties
- D* Lite (2002 & 2005)
- Field D* (2005)
- Multi-resolution Field D* (2006)
Problem Definition: Path Finding
Motivation

- Only A* in lectures
- D* is more useful for robotic domain.
- Used in various robots including Mars rovers "Spirit" and "Opportunity"
Environment Properties

- Static vs Dynamic
- Complete vs Incomplete (Accessible vs Inaccessible)
- Discrete vs Continuous
- Deterministic vs Non-deterministic
- Stationary Target vs Moving Target

We assume **discrete** & **deterministic** environment with **stationary** target.

Note: All **continuous** domains can be **discretized**.
Robotic domain is **continuous**, why discretize? Discretization is a mathematical method.

- Easier calculation: Making data more **suitable** for
  - numerical computation
  - implementation on digital computers
Problems of Grid Based Path Planning

- Path Quality (Limited rotation values \((0, \pi/4, \pi/2)\) 4 or 8 neighborhood)
- Memory & computational power requirements
Lack of Smooth Paths

Figure 10. Paths produced by $D^*$ Lite (top) and Field $D^*$ (bottom) in a $150 \times 60$ uniform resolution grid. Again, darker cells have larger traversal costs.
Other Environment Properties

- Stationary Target vs Moving Target (MTS)
- One agent vs Multi-agents
- Fuel constraint? (PHA*)
- Time constraint? (Anytime algorithms)
- Real-Time? (RTA* & LRTA*)
- Shortest path needed?
- Agility or Fatigue?
A* Environment Assumptions

- **Static** world
- **Complete** knowledge of the map
- Freespace Assumption: The robot assumes that the terrain is **clear** unless it knows otherwise.

& also **deterministic** and **discrete** with **stationary** target.
A*

- Covered in 585 lectures.
- Brief reminder: Breadth first search using a **heuristic** function.
- Forward A*: Search starts from the start to goal
- Backward A*: Search starts from the goal to starting point.
A* Details

- \( f(x) = g(x) + h(x) \)
- \( g(x) = \) Path cost from start to node \( x \)
- \( h(x) = \) ”Heuristic estimate” of the distance to the goal from node \( x \).
- \( h(x) \) should be admissible (kabul edilebilir). It must **never overestimate** the distance to the goal, so that A* guarantees to find the **shortest path**.
- Implemented with a **priority queue**.
Priority Queue

- **Priority Queue** is an abstract data type
- **Heap** is a data structure.
- Priority Queue can be implemented with heap structure.
Dynamic A*

Capable of planning paths in unknown, partially known and changing environments efficiently.

When the map changes or an unknown obstacle cuts the way, the algorithm replans another path efficiently.
How D* Lite works?

- First run is the same as A*. D* finds the same path with A*.
- When a node changes, D* just recomputes the values of the **inconsistent** nodes, which are necessary to compute, while A* recomputes all of the path.
- D* makes **backward** search. (Starts searching from the goal node)
Inconsistent Nodes

![Knowledge in Start Situation](image1)

![Knowledge after the Discovery of an Additional Obstacle](image2)

Fig. 2. Simple example (part 1). Gray cells are cells with changed goal distances.
Inconsistent Nodes

- Consistency = (g(x) == rhs(x))
- If a node is inconsistent update all of it's neighbors and itself again. (Updating nodes will try to make them consistent)
- Continue updating while the robots node is inconsistent or there are inconsistent nodes that are closer to the target, which may open a shorter path to the robot.
How D* Lite works?

Example run of D*
How D* Lite works?

First run of D*

Fig. 6. Operation of the second version of D* Lite (part 1).
How D* Lite works?

Second run of D*

Fig. 7. Operation of the second version of D* Lite (part 2).
## Comparison

### TABLE 1
**Experimental Results—Terrain With Random Obstacles**

<table>
<thead>
<tr>
<th>Search Algorithm</th>
<th>Planning Time</th>
<th>Cell Expansions</th>
<th>Heap Percolates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-First Search</td>
<td>302.30 msecs</td>
<td>845,433</td>
<td>4,116,516</td>
</tr>
<tr>
<td>Backward A*</td>
<td>10.55 msecs</td>
<td>17,096</td>
<td>276,287</td>
</tr>
<tr>
<td>Forward A*</td>
<td>7.29 msecs</td>
<td>8,722</td>
<td>177,476</td>
</tr>
<tr>
<td>DynamicSWSF-FP</td>
<td>6.41 msecs</td>
<td>13,962</td>
<td>75,738</td>
</tr>
<tr>
<td>(Focussed) D*</td>
<td>4.28 msecs</td>
<td>2,138</td>
<td>79,214</td>
</tr>
<tr>
<td>D* Lite</td>
<td>2.82 msecs</td>
<td>2,856</td>
<td>32,988</td>
</tr>
</tbody>
</table>
### TABLE II
**Experimental Results—Fractal Terrain**

<table>
<thead>
<tr>
<th>Search Algorithm</th>
<th>Planning Time</th>
<th>Cell Expansions</th>
<th>Heap Percolates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-First Search</td>
<td>194.13 msecs</td>
<td>543,408</td>
<td>2,643,916</td>
</tr>
<tr>
<td>Backward A*</td>
<td>5.49 msecs</td>
<td>8,680</td>
<td>156,801</td>
</tr>
<tr>
<td>Forward A*</td>
<td>4.78 msecs</td>
<td>5,459</td>
<td>124,814</td>
</tr>
<tr>
<td>DynamicSWSF-FP</td>
<td>6.26 msecs</td>
<td>13,931</td>
<td>76,703</td>
</tr>
<tr>
<td>(Focussed) D*</td>
<td>1.18 msecs</td>
<td>596</td>
<td>19,066</td>
</tr>
<tr>
<td>D* Lite</td>
<td>0.97 msecs</td>
<td>393</td>
<td>5,316</td>
</tr>
</tbody>
</table>
Fig. 8. Simple example (part 2). Gray cells are cells that were expanded.
D* Lite has 2 versions, which just have different implementations, and D* is also another algorithm. They all find the same path. Second D* Lite algorithm is the fastest.
Problems of Grid Based Path Planning

- Path Quality (Limited rotation values \((0, \pi/4, \pi/2)\)) Solved by Field D*
- Memory & computational power requirements
Field D*

- Operates on continuous domain. After D* computes the path, a post processing function, shortens the path based on interpolation.
Figure 10. Paths produced by D* Lite (top) and Field D* (bottom) in a 150×60 uniform resolution grid. Again, darker cells have larger traversal costs.
Problems of Grid Based Path Planning

- Path Quality (Limited rotation values \(0, \pi/4, \pi/2\)) Solved by Field D*
- Memory & computational power requirements Solved by Multi-resolution Field D*
Multi-resolution Field D*

- Computes nearly the same path with Field D* in **shorter time** with **less memory** usage.

**Figure 1.** (left, center) Multi-resolution Field D* produces direct, low-cost paths (in blue/dark gray) through both high-resolution and low-resolution areas. (right) The GDRS XUV robot used for autonomous navigation of outdoor terrain. Multi-resolution Field D* was initially developed in order to extend the effective range of rugged outdoor vehicles such as this by one to two orders of magnitude.
Related Papers

- Optimal and Efficient Path Planning for Partially-Known Environments, Anthony Stenz, ICRA, 94
- Multi-resolution Field D* IAS 2006 Dave Ferguson, Anthony Stenz
Questions?

Thanks for listening