Harvard CS 182: Artificial Intelligence September 15, 2021

Lecture 4: Motion Planning

Lecturer: Ariel Procaccia *Author: Shuvom Sadhuka*

In this lecture we cover *motion planning*, which will combine ideas from the first two lectures on *uninformed search* and *informed search*. In motion planning problems, we wish to navigate from a source (s) to a destination (t) while avoiding obstacles. Unlike previous lectures, however, where discrete graphs defined the search problem, we now have a continuous space. Let M be a continuous region in which s and t live, and $\mathcal{O}$ be a set of obstacles in this map. A key challenge in motion planning is to discretize M such that we can apply the algorithms from previous lectures.

1 Cell Decomposition

One natural way to discretize M is to decompose it into cells. For instance, we can subdivide $M \setminus \mathcal{O}$ into squares and choose the center of each square as a node in a graph. We delete nodes in squares that intersect with some obstacle in $\mathcal{O}$ and the result is a discrete set of points D in $M \setminus \mathcal{O}$ ("free space"). We can then draw a graph on this set of points (edges drawn between vertices in adjacent cells) and run a search algorithm such as A^*

The issue with such an approach is that there may no be path from s to t through the nodes in D even when a valid path from s to t exists — i.e. t may not be connected to s through the points in D. In particular, an obstacle which only partially intersects with some cells will render these cells null, when in fact some of these cells contain part of the valid path(s).

One solution is to further subdivide the cells which only partially intersect with the cell and re-run A^* until a valid path is found. In pseudocode (visual in Figure 1):

Algorithm 1 Cell Decomposition

```
cell ← M
while path not found do
  subdivide cells partially intersecting with 𝒪, append to set 𝒯
  mark center of each new cell t ∈ 𝒯
  delete all centers in cells t where t intersects with some obstacle
  run A* to find path
end while
```

Definition: a motion planning algorithm A is resolution complete when:

1. if a path exists, it finds it in finite time

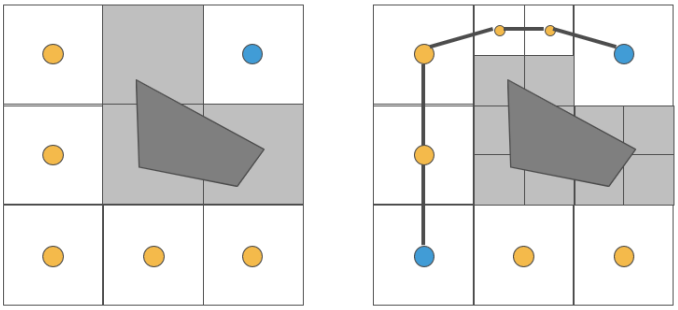

Figure 1: If the current mesh doesn't work, cell decomposition subdivides cells partially intersecting with the obstacles and reruns the search.

2. if a path does not exist, it returns in finite time

Algorithm 1 is not resolution complete, as it may not return in finite time when a path does not exist, as we can construct an obstacle set $\mathcal{O}$ and map M such that Algorithm 1 keeps subdividing the grid further.

Another issue with Algorithm 1 is that it may not find the shortest path due to the exact subdivision procedure. To solve this issue, we can apply A^* smoothing. In particular, suppose Algorithm 1 has found a valid path through some set of nodes $x_1, \ldots, x_n$ where x_k is visible from x_j and $k \neq j+1$. Then, we can shorten the path to $x_1, \ldots, x_j, x_k, \ldots, x_n$. However, it is not necessarily true that x_j, x_k exist even if $x_1, \ldots, x_n$ is suboptimal. In other words, it is not necessarily true that a visible shortcut exists even when the path we chose was suboptimal (Figure 2).

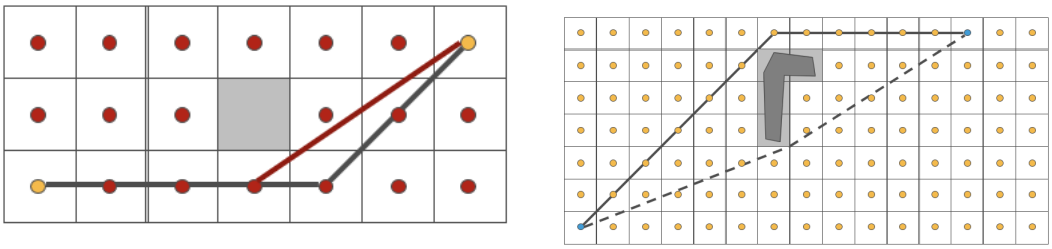

Figure 2: (Left) smoothing tries to find shortcuts in the path that A^* found. (Right) However, smoothing may not always be able to find a better path even when a suboptimal path was chosen.

A second solution is the θ^* algorithm. The key idea to this approach is to allow parents that are non-neighbors in the grid to be used in the search. Recall that in standard A^* when we insert a new node y, we insert it with a cost

$$g(y) = g(x) + c(x, y)$$

The estimate is then

$$f(y) = g(x) + c(x, y) + h(y)$$

We modify this in θ^* to be:

$$f(y) = g(parent(x)) + c(parent(x), y) + h(y)$$

In other words, we directly insert y from the parent of x instead of from x. This solution does better in many real-world settings.

2 Polygonal Paths

In the previous section, we saw a few algorithms for finding a valid path through $M \setminus \mathcal{O}$, but these were not necessarily optimal paths. In this section, we consider an abstract setting and describe how to find the optimal path.

Definition: a polygonal path is a sequence of straight lines through M

Definition: an inner vertex of a polygonal path is a vertex that is neither the beginning nor the end of the path.

Theorem 1 *Assuming $\mathcal{O}$ is a set of open polygonal obstacles, the shortest path from s to t through M is a polygonal path whose inner vertices are vertices of the obstacles.*

Sketch of Proof:

Assume towards a contradiction that the shortest path through such a map is not polygonal with inner vertices as vertices of the obstacles.

Lemma 1 If the path is not polygonal, there exists a point p in the interior of free space $(M \setminus \mathcal{O})$ such that the path through p is curved.

Proof of Lemma First note that there must exist a point p through which the path is curved as by assumption there is at least one section of the path that is curved. To show that such a p must exist in the interior of free space, assume that the only such p is on the edge of an obstacle. However, because the path through p is curved, we must be able to go slightly (i.e. some arbitrarily small ε) off p and into free space (since the curve must not follow the straight line of the obstacle edge by definition) which is the interior free space. Therefore, such point p exists in interior free space.

By the same logic, there exists an ε-ball of free space around p through which the curve passes. The path through this ball can be shortened, a contradiction to the minimality of the path we chose. **End Lemma 1**

Therefore, there are no curves in the path, indicating the path is polygonal. Now we prove that the inner vertices must also be vertices of the obstacles.

These vertices cannot lie in the interior of free space, as we'd otherwise be able to use the same trick as in Lemma 1.

Similarly, a vertex cannot lie on the interior of an obstacle edge, as we'd be able to apply the same ε-ball trick as in Lemma 1. ■

From theorem 1 we know that if all the obstacles are polygonal and the player moving through the map is a point (i.e. no volume), then the optimal path must be a polygonal path through some subset of the vertices of $\mathcal{O}$. Thus, we define the **visibility graph** (Figure 3) to be a graph with vertices as the vertices of the polygons, s, and t, and the edges to be all pairs of vertices (x, y) such that x is visible from y (i.e. a straight line path exists with no obstacles in the way).

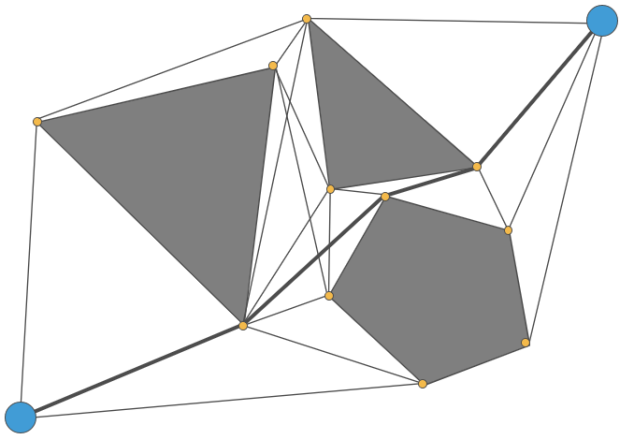

Figure 3: We can construct a visibility graph by connecting s, t, and vertices of the obstacles.

We can then run a search procedure on the visibility graph to find the optimal path. Through theorem 1, we've reduced a path planning problem over a continuous space to a discrete graph (given some assumptions about the obstacles)!

However, it is possible to construct a map M and obstacle set $\mathcal{O}$ such that the optimal path in the visibility graph will visit $\Theta(n)$ nodes, where n is the total number of nodes in the visibility graph.

3 Configuration Space and Probabilistic Roadmap

Up to now, we've considered the player moving through M to have no mass. However, this is often not the case, as the player may be a robot which takes up space. Thus, we transform the physical space (i.e. $M, \mathcal{O}$) into a configuration space. For instance, in the configuration space we might have that $\mathcal{O}' = \mathcal{O} \cup \mathcal{O}_p$ where $\mathcal{O}_p$ is the region of physical space such that the robot is partially intersecting the obstacle. Configuration spaces for other tasks, such as representing a robotic arm (parameterized by two angles), are also possible.

Another strategy for path finding and motion planning is a **probabilistic roadmap**. In this setting, we randomly choose a set of points in the map M, discarding the ones that are blocked. The remaining m points are **milestones**, and we draw edges between any milestone m_i and its k nearest neighbors.

If t is reachable from s in this graph, then we've found a path.

Another variation of the same idea is **rapidly-expanding random trees**, in which we incrementally build two trees rooted at s and t respectively. We then randomly sample a milestone, and connect it to the closest visible point in each tree (if such a point exists). If a milestone is visible to both trees, then we are done!