A Review of Path Planning for Autonomous Robotics

Joseph M. Wetterling
jwetterling@acm.org

Introduction

The problem of path planning is a common one in computer science, particular in the area of artificial intelligence. It involves finding the optimal path from some starting point to an end point or goal state. This optimal path is defined, in this case, as the shortest or lowest-cost of those paths that span from start to goal. The area being searched can be defined, for instance, as a state-space (network) of nodes and the available paths as the arcs between those nodes.

Imagine we are trying to get to the Trenton Computer Festival in Edison, NJ from New Brunswick (Figure 1). There are several different routes we can take, and we must consider both the length of each road and how close it will take us to our goal in making a decision. Traveling to Milltown may be a short trip, but it takes us in the wrong direction. While Highland Park may be a slightly longer step, it gets us significantly closer. While these decisions seem like common sense to us, a computer needs to be taught such planning rules. (Its hard to consider "moving toward your destination gets your there faster" to be a "rule" only because we know it so implicitly.)

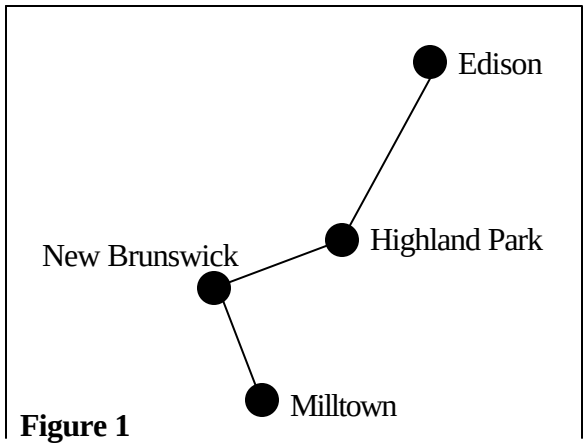

Figure 1

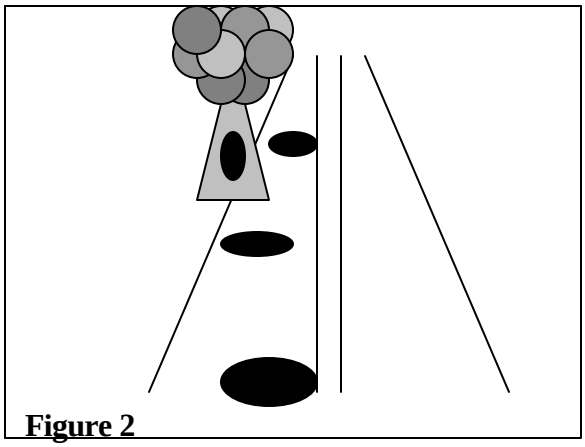

Figure 2

At other times, two paths may both be heading in the same direction and seem the same length. Another factor, broadly called the "path cost", helps determine the better choice. Consider the two roads in figure 2. While Robert Frost might disagree, there is an obvious choice when efficiency is the key factor. One path is clearly easier (and, therefore, less "costly") to travel than the other.

A good path-planning algorithm, then, needs to possess some "common sense". It also must be able to work efficiently, as most meaningful problems (navigating a room, traveling from one city to another, etc.) involve a great number of possible choices.

Brute Force Methods

"Brute force" solutions to path planning solve the problem and some could even be considered elegant solutions – expressed in very few, understandable lines of code. These methods force their way to the solution by checking every possible answer until either the correct one is found or the list of possibilities is exhausted. For any significant problem, they violate the requirement of efficiency.

Consider the state-space in figure 3. The shortest path from start (marked 'O') to finish (marked 'X') can easily be found using a breadth-first search (pseudo-code shown in figure 4).

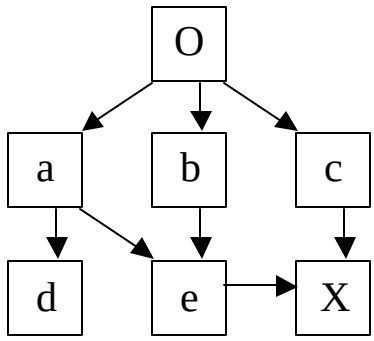

Figure 3

```
Put State on the OPEN list
While OPEN list is not empty
    Remove leftmost state from OPEN
    If it is the goal, success!
    Else
        Generate children of the state
        Put the state on CLOSED
        Ignore children on OPEN/CLOSED
        Put others on right end of OPEN
```

Figure 4

Breadth-first search is one example of a brute-force search. It doesn't use a really intelligent strategy to find the goal. Breadth-first, however, does happen to always find the best solution. Not all brute-force methods have this guarantee (depth-first search, for example).

Breadth-first search uses two lists, called OPEN and CLOSED here. These list which states, out of O, a, b, c, d, e, X, are currently under consideration (OPEN) or completely eliminated (CLOSED). First, the start state (O) is put on the OPEN list. The rest of the search takes place in a loop, running until it is out of states on OPEN (failure) or finds the goal (success). During each iteration of the loop, the search looks at the leftmost item on the OPEN list; we'll call this X. The children of X – that is, those nodes pointed to by arrows coming from X – are examined. All those not already duplicated on the OPEN list or already CLOSED are put on the list for consideration.

Using the example in Figure 3, the search would progress like this:

	OPEN	**CLOSED**
1.	O	none
2.	a b c	O
3.	b c d e	O a
4.	c d e	O a b
5.	d e X	O a b c
6.	e X	O a b c d
7.	X	O a b c d e
8.	Success – X is the goal.	

Note that in step 4, because e was already on the list (a -> e), it was not added again. The search stops after X is found; it is coincidence that X happens to also be the last node in the search. If the goal were at node c, the search would have stopped at step 5, when c was examined. This procedure found the answer, of course, but it needed to go through every possibility. If there were hundreds of nodes, this could have been a very long exercise!

Admissible Heuristics

There are considerably better options available. Much literature has been published about path-planning methods that are more efficient than brute-force algorithms. Most of these methods use heuristics – "guesses" about which paths are likely to be good answers.

There can be many heuristics able to solve a single problem. For example, game playing – a cousin of the path-planning problem – makes use of many heuristics. In the game of chess, there are countless ways to gauge your performance versus your opponent. You could look at who has more pieces. This is a heuristic. You could give each piece a value (pawn = 1, queen = 9, etc.) and consider who has more points. This is another possible heuristic. For yet another, you could judge who has better control of the center or makes better use of his pawns. All of these can help solve your problem, but which do you use? We need a way to judge which heuristics will solve our problem better. One way to do this for path-planning problems is to use the idea of "admissibility."

An admissible algorithm is one that never overestimates the cost of traveling from one node to another. If there is a cost, X, of traveling from node a to node b, then an admissible heuristic would return a value <= X for this path. The closer it gets to X, the more "well-informed" it is considered, but the fact that it is at or under X is what matters more.

The common name for these algorithms needs to be mentioned, as it will come into play later. An algorithm that works perfectly (which is impossible for any significant problem) is called A. This is our ideal. The algorithm that approximates it, using an admissible heuristics, is called A* (read "ay-star").

Why be concerned with admissibility? Simple. Avoiding a detailed analysis – which many good introductory artificial intelligence textbooks can give you – an algorithm using an admissible heuristic is guaranteed to always return the optimal path! Algorithms with admissible heuristics have solved many path-planning problems. By describing an environment as a series of nodes – a state-space – and applying an admissible heuristic, we can find the optimal path to our goal.

Still, there are more problems that cannot be solved by this procedure. Algorithms using admissible heuristics rely on perfect information and an unchanging state-space. Perfect information means that we know, in advance, exactly what the environment is like – number and position of nodes, cost of paths, etc. And an unchanging state-space means that that perfect information remains perfect. This presents a sizeable problem for us. Almost every significant problem breaks one or both of these requirements. Even walking across a room lacks unchanging information, which anyone who has ever stubbed their toe in a chair or end table can tell you.

The Dynamic Planning Problem

These changing, imperfectly-known environments – the "real world" problems – present a need for *dynamic* planning. First, there is the case of not having perfect information. A robot, especially, needs to react to discoveries about its environment. A simple example of this comes from the bump-and-go cars I used to play with as a child. Granted, they had a fairly simple - and rather chaotic - algorithm for handling contact with walls, but they *did*, in fact, have a response to the problem. A more sophisticated robot could record the new information and move around it – either traveling along the wall, hoping to follow it closer to the goal, or perhaps assuming it is a large obstacle, and turning away from it. The key is recording the information for future use. A state-space map is an excellent method for such recording, though there are others. Without this recorded information, the robot is forever stumbling about "in the dark".

Second, there is the case of changing information. Even in an environment which is currently perfectly-known, things can change in the future. If a robot I sent into a perfectly mapped room, and someone has moved a chair by one inch, what happens to the robot? Would it drive repeatedly into it, or learn the new location and plan around it? Planning around such new obstacles is difficult, however, as the size of the object is unknown. We only know it is there; it could be the size of the robot or a hundred times the size of it.

There are several research projects being done along this vein. Carnegie Mellon University and Rensselear Polytechnic Institute, for two examples, are doing research in the area of robot path planning. Their work is highly recommended reading. One research project from a faculty member at Carnegie Mellon is of particular interest, and it's described briefly here.

The D* Algorithm

An algorithm called D* (read "dee-star"), which stands for "dynamic A*", provides a solution for changing environments. Calculating a new optimal path each time the environment changes is inefficient, and it could result in hundreds or thousands of replannings.[1]

Many obstacles to be encountered in a changing environment will be relatively small and therefore affect only the local environment (the "nearest" few states, by depth and/or cost)[2]. The D* algorithm only replans this local path, assuming that the rest of the planned path is still viable. (There is no reason not to assume this, as the more distant path may be the same or vastly different; the algorithm will replan when it reaches those later obstructions anyway.)

---

[1] Stentz, Anthony. *Optimal and Efficient Path Planning for Partially-Known Environments*. Proceedings IEEE International Conference on Robotics and Automation, May 1994. Online. Carnegie Mellon University. Internet. 7 Feb. 2001.

[2] Stentz, Anthony. *Real-Time Replanning in Dynamic and Unknown Environments*. Online. Carnegie Mellon University. Internet. 7 Feb. 2001.

The developer of D*, Anthony Stentz of Carnegie Mellon University, proves the admissibility of D* in one of his early technical reports, which makes it a viable solution to the dynamic planning problem.[3]

3-D Environments

There are plenty of other, real-world concerns in planning for autonomous robotics. I'll address only one here, briefly, to give the reader an idea of them. A robot traveling in the real world needs to handle not only information about the length and width of objects and paths around it, but it must understand height as well. Why is this necessary when a robot may be low-to-the-ground and only move along two dimensions?

Robotic vision is not nearly as accurate as our own, at least as far as distinguishing objects and paths. If the robot is looking at the image in figure 5, it may represent this internally as shown in figure 6. The distinction between a safe path and a potentially dangerous obstacle is lost because in two dimensions, they look alike.

---

[3] Stentz, Anthony. *Optimal and Efficient Path Planning for Unknown and Dynamic Environments*. Carnegie Mellon Robotics Institute Technical Report CMU-RI-TR-93-20, Aug. 1993. Online. Carnegie Mellon University. Internet. 7 Feb. 2001.

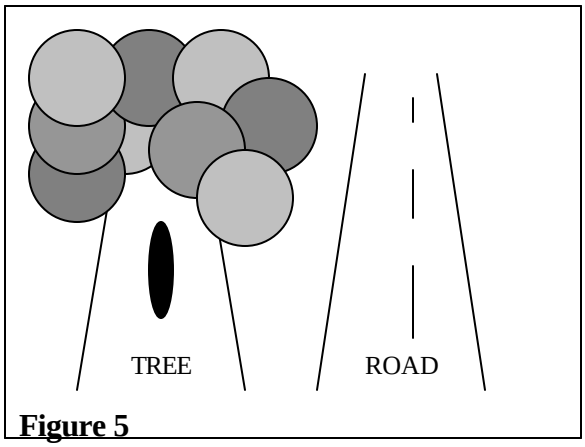

Figure 5

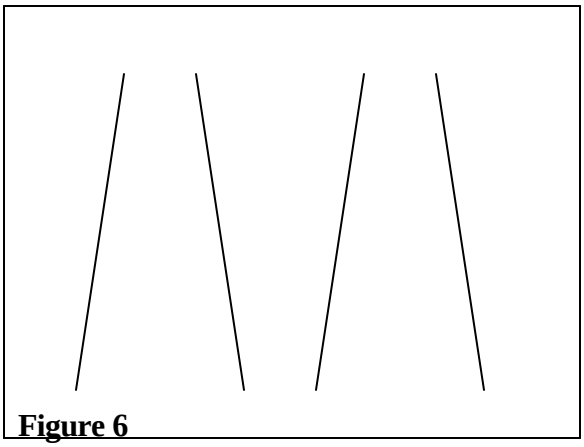

Figure 6

There are several specific concepts – admissibility, handling imperfect information, dynamic versus static planning – which comprise the basics of robot path planning. There are many different directions to go from there, however. Path planning can concern itself with navigating a room or a moon, with learning the best way around an unchanging office or dynamically reacting to falling debris in a dangerous rescue operation. For further reading, I suggest several starting points:

- MIT AI Lab: Mobot Group
  http://www.ai.mit.edu/projects/mobile-robots/
- NASA, Advanced Autonomy for Rovers
  http://ic-www.arc.nasa.gov/projects/ai-rovers/index.html
- Rensselear Polytechnic Institute, Robotics Lab
  http://www.robotics.cs.rpi.edu/
- The Robotics Institute, Carnegie Mellon University
  http://www.ri.cmu.edu/
- The Tech Museum: Robotics
  http://www.thetech.org/robotics/