Tree Cache

Ken Anderson
March Networks
kenneth.a.anderson@gmail.com

Abstract

This program generates a single search tree, compacts it to reduce memory/disk space, then utilizes it to very quickly generate valid paths in an 8-connected gridworld. This technique is similar in principle to RRTs used in robotics.

RRTs

RRTs (rapidly-exploring random trees) are a classic path-planning technique used in the robotics community. RRTs first create a graph within a continuous search space. Beginning at the start state, random states are generated and the closest state in the tree is expanded toward that node. A state is added to the graph if a straight line between the state and the closest other state in the graph is collision-free.

The RRT can be used for path-planning very simply: create an RRT rooted at either the start or goal node. Periodically try adding the other state to the tree. Once the other state is connected to the tree, reconstruct the path by simply traversing up the tree from the goal state to the root (start) state. This process is quite fast, both for generating the graph and creating the solution.

Tree Cache

For this competition entry, paths are generated very quickly on the 8-connected gridworld using an approach similar to that of using an RRT on a continuous domain. First, a valid state is selected near the center of the graph to root the tree structure. Then the tree is generated, but instead of creating an RRT, a search tree is created using Djikstra's search. The tree is stored in two arrays in memory. One array stores the operator used to get to the predecessor state (one byte). The other array stores the cost to get to the tree root (one byte). The memory required is 2 times the width times the height. In this competition the maximum height and width are 2048, so the maximum memory is 8 MB.

Once the tree is created, it is used to generate paths. No search is involved. The start state and goal state are pushed onto separate stacks. The top state on each stack is examined. The parent of the state that is further from the root is added to its respective stack. When the states on the top of each stack are the same, the two paths are combined into one final path.

Conclusions and Future Work

This entry is designed exclusively for speed. Relying solely on this technique will actually generate fairly poor solutions in most cases. When RRTs are used in practice, the robotics community typically uses a second step, called shortcutting, to improve solution quality after the initial path has been generated. These shortcutting techniques attempt to connect local portions of the path by shortcutting between states. To improve solution quality, shortcutting is a good option, which can be performed incrementally, enabling an anytime algorithm implementation.

Another improvement is to more selectively choose the tree root. Choosing a state at the center of the map is not necessarily the best approach; the state could be in a dead end. It may be worthwhile investigating further the criteria that make for a good tree root location.

References

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Technical Report, TR 98-11, Computer Science Deptartment, Iowa State University.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*: 269–271.