Learning Real-Time A* Search via a Multi-agent System

Jason Natale

University of Auckland, Auckland, New Zealand
jnat826@aucklanduni.ac.nz

Abstract. The purpose of this assignment was to develop a multi-agent solution to a given problem. I chose the problem of multi-agent path finding due to its explicit visualization of agents interacting as they algorithmically find an optimal path between two points. The core algorithm used in this implementation was Learning Real-Time A*, with some functionality from the Boris platform being utilized.

1 Path Finding Problem

Over the years, computer science researchers have developed an immense collection of algorithms to discover optimal paths within a graph. This is largely due to the vast variety of real world problems which can be formatted as a graphical space for these solutions to be applied to. Here, I've focused on finding the shortest path within an undirected graph (or mesh) of equidistant vertices. It should be noted though that my solution to this problem would also be valid for directed graphs with variant edge weights. There have been obstacles added within the virtual environment used to replicate barriers of any kind which could be found in more realistic scenarios. Given a start vertex and an end vertex, I aim to find and highlight this shortest path comprised of edges along a uniform mesh.

2 Solution

In CSC767, we've outlined a couple asynchronous approaches to solve the shortest path problem via a multi-agent system (MAS). I've utilized a form of the Learning Real-Time A* algorithm which has been extended to a MAS implementation. This variant, LRTA*(n), works with any "n" number of independent software agents. The Boris multi-agent system Java package was used to initialize agents, although this was later found to be an unnecessary addition as functionality stemming off of this interface could have been replicated with a small amount of independent work. This is mainly due to the fact that agent-to-agent messaging is not necessary for the LRTA*(n) algorithm to work properly.

2.1 Learning Real-Time A* Algorithm

Like any algorithm grounded in dynamic programming principles, LRTA* deconstructs the large problem of finding an entire shortest path between two potentially distanced vertices to many smaller problems of finding the closest adjacent vertex to another. Setting a focus vertex “u” to the start, we iteratively continue to find new distance functions (f(v1), f(v2), etc.) until this focus vertex is our end point. Each iteration of this loop first consists of a thorough sweeping of all vertices in the given graph. If a vertex is found to reside on an edge with “u”, a distance function is produced for that vertex. This function is: f(v) = w(u,v) + h(v), where w(u,v) is the weight of the edge between vertex “u” and our chosen vertex “v”, and h(v) is the current heuristic of “v”. Once all adjacent vertices have had their distance function’s computed for an iteration, we set a vertex v’ as the vertex with the minimal distance function. Then, we update the heuristic for our focus vertex “u” to equal the max value between its current heuristic and the distance function to the v’ which was just discovered. Lastly in a single iteration, our focus vertex “u” is set to equal v’. This procedure as described represents the synchronous version of LRTA which only a single agent (or thread) would be able to process.

Converting this algorithm to work with a MAS doesn’t take too much tweaking though. First, each agent must be provided this algorithm to process over. Second, proper locking mechanisms must be developed to protect shared data from race conditions. Lastly, for the sake of clear visual representation a synchronization mechanism has been put into place to ensure that each independent agent completes a single trial before a generation or set of paths computed for a given trial is output. Solving the first issue, I developed a Worker class extending from Thread and wrote the main functionality of the LRTA algorithm in a run method which would be executed for every search started. An infinite loop contained the algorithm for a single trial, as a solution is only known when two consecutive trials result in the same path. Since the only shared data was the heuristic values of each vertex, it was also a relatively simple fix to eliminate any race conditions. I simply added synchronized setter and getter methods for the heuristic of my Vertex objects. Finally, a CyclicBarrier was utilized to ensure that all agents completed a single trial before that generation’s results were output. Some additional thread scheduling was needed to finalize a search when a single agent discovered a solution to the search.

2.2 Agent Roles, Perception, Coordination, and Communication

The interaction between agents in LRTA*(n) is more subtle than one might find in other MAS solutions. Each is provided with an identical copy of the entire search algorithm, but in this process relay information to one another via heuristic adjustments. All agents are also omniscient within this given environment. Each can perceive any vertex or edge in the graph. Only very minute coordination is required to avoid race conditions although a more visible representation of LRTA*(n), as attempted with this project, necessitates synchronized completion of trials and inter-agent communication upon completion.

3 Demo

A video demonstrating this project can be found at:
https://youtu.be/oa4MZhSslG4