Matt Forbes
January 20, 2011
CS 400 - AI Indep. Study
Ch 2 & 4 Discussion

Intelligent Agents

An agent perceives its environment and is capable of action. That is the simple definition, but for an agent to be useful, it must have both goals and the 'intelligence' to act in a way that its goals are met. In its most basic form an agent should be able to sense its environment, perform actions, and detect if a goal has been met.

In the book, there is a simple example of an automatic vacuum cleaner that should move around and clean up its environment. One property of these cleaners we varied was whether or not it kept track of its cleaning history. The vacuum that doesn't track state will endlessly roam around its environment even if there is nothing to clean.

When the agent's only goal is a clean room this is fine; if we try to minimize the battery spent moving and cleaning it would definitely not be a good model.

Agent Applications

A generic model of agents such as this are applicable to a huge range of problems. Basically anything that can observe its environment and makes decisions from that can be categorized as an agent. Automated robots are a general case, while web crawling bots are more specific.

One of my math professors has told me about his current research project, where he is trying to find a chemical dye that when used in a solar cell produces the effects he is looking for. The solution to this problem is highly experimental, so he generates data by running simulations. When he has a set of data, he can analyze which changes in variables got him closer to the solution. After making a few changes, he runs the simulations again.

I envisioned this process similarly to an intelligent agent. After running the simulation (performing an action), he then re-evaluated his environment (perception), finally making changes and re-running the simulation (informed action). A* search might come in to this equation when searching for a new simulation state after receiving the results.

A* and the Traveling Salesman Problem

A* search tries to direct the would-be headless walk through a search space. Rather than exhaustively check every possibility, we can just expand nodes that we expect to be good bets. To accomplish this we need some notion of "good" when describing a node in the tree; this is the heuristic. A heuristic function estimates the path length to the destination

from it. When combined with the path cost to the node in question, we have a good way rank nodes.

A good definition of the TSP is: "Find a path through a weighted graph which starts and ends at the same vertex, includes every other vertex exactly once, and minimizes the total cost of the edges." Brute forcing this problem would be a disaster, so taking a bit more directed approach is more appropriate (but still inefficient.)

Minimum spanning trees connect all the vertices of a graph with a minimum total edge weight. This total edge weight will always be less than or equal to the final path of a TSP tour. If we take the TS problem and drop the constraint that you can only visit each node once, it would be solved with a MST. Because of this property of always giving an optimistic estimation, the MST would be a good heuristic for the TSP.

So that's how I implemented my solution to the TSP. Using A* with MST heuristic to direct an informed search through the large search space, my non-optimized implementation can solve a graph with about 10 nodes and between 2 and 5 edges per node in a couple seconds.