Artificial Intelligence with Logic Programming (COMS30106)

Dhiraj Narwani (1431555), Mario Viti (10110)
Department of Computer Science, University of Bristol
BS8 1UB, UK

Implemented strategy.

Please run this command to test code: **repeat, complete, ailp_reset, start_solving(A)**.

The aim of this assignment was to implement an intelligent strategy where an agent is moving around in a maze to deduce his identity. In a nutshell, we tried to make our agent a bit dumb and presumptuous by using simple strategy thumb rules instead of pondering the costs of all possible paths (always finds the best solution). We modeled our strategy to consist of two states or behaviours: normal and critical. Our agent will switch between these two states when its current energy level hits a dynamic threshold. This is represented by a bound predicate. If we cannot afford asking an oracle, there is a possibility of running out of energy, so we increase the threshold forcing us to search for the nearest charging station. In order to do this, we added a third state to switch between the two previous states.

Below in Figure 1 and 2, the states are represented as two branches of the find predicate. Figure 3 shows the switch state representing the energy switch predicate. The strategy consists of switching between the different states which are expressed procedurally as a recursive structure. This is similar to a finite state automa.

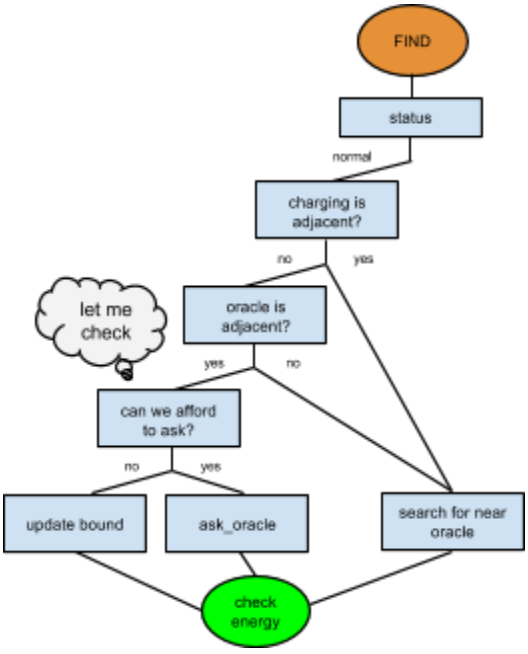

Fig 1. Normal state branch

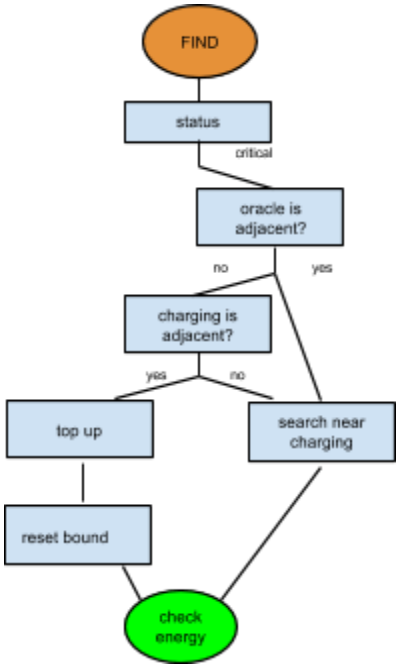

Fig 2. Critical state branch

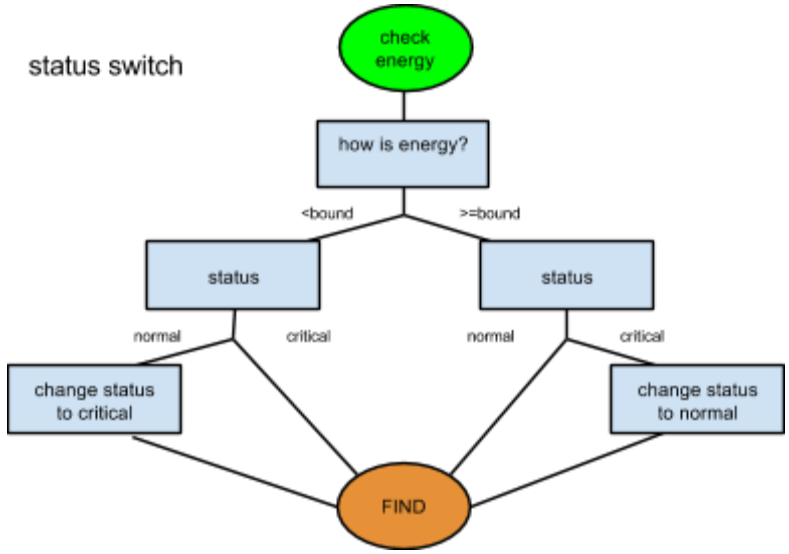

Fig. 3. Energy switch case

Optional.

Please run this command to test code: **repeat, complete, reset_game, start_game, start_solving(A)**.

Our strategy also applies to the dynamic world, the only difference is that we need to replan if a path is obstructed by an obstacle aka “thing” moving. In this way the agent is able to dodge “things” by moving around them. Since the positions of oracles and charging stations are being dynamically changed, once the agent finds its target, it may have already moved. Therefore, it appropriately re-plans and locates the nearest oracle/charging station. Moreover, this consists of a switch case firing a recursive call:

query_world(agent_do_moves, [Agent,R])->true
; otherwise->solve_task_A_star(Goal,_NCost)

The corner case we did not deal with is the situation in which the agent is trapped. Traditionally we fail, but since we are living in a dynamic world, being trapped may be temporary, therefore one possible solution may be to set a fixed number of trials before failing.

CAVEAT: When testing, this strategy doesn’t perform too well when facing extreme conditions in the dynamic world (all objects moving in all 4 directions every 10 seconds, meaning a change every 2.5 seconds). This is also due to the speed of the agent in the maze.