Q–Learning Proposal

By Sam Shinn *on* November 20, 2017

Description

The goal of this project is to design a program which will allow a virtual robot to traverse a virtual representation of the 4th floor of WWU's Communications Facility (CF) building. In particular, the routes of traversal will not be programmed into the robot, but instead the robot will be simply issued a goal destination and the robot will use its own prior experience to determine the best path to the goal destination. This condition of using prior experience means that without prior knowledge of the environment, the robot will have no choice but to select a route at random. However, the robot will then remember whether the selected route was useful in reaching the goal and then proceed to use this knowledge to improve pathfinding ability in subsequent attempts.

Auxillary Files

In the zip file containing this document are three image files:

- CFblueprint.png
- StateDiagram.png
- NestedStateDiagram.png

The CFblueprint file is a modified version of the original blueprint of the CF 4th floor converting all the room numbers to contiguous values starting from 0, thereby making the representation of the 4th floor friendlier to index representation. A note on the CFblueprint file, blue boxes can be observed separating rooms which were not previously separated by doors according to the original blueprint; their presence is simply to make explicit the delineation between rooms and they are not obstacles in any way. The StateDiagram file shows how all the rooms connect to one another. Because the state diagram is not very well interconnected, but is instead sort of hierarchical, another rendition called NestedStateDiagram is also included. The reasoning for this representation is explained later in the 'Discussion' section.

Approach

Q-Learning is generally used when the goal state is fixed, and the Q-Table (which directs the robot's movements) is populated with calculated values which will allow the robot to find the goal state in the shortest route from any starting position. In such a case it is possible to use one single Q-Table for this purpose because the pathways converge to a single goal state. But this application is different in that the robot will be required to travel from a fixed position to any arbitrary position, meaning that the pathways are now divergent. This means that it's not possible for a single Q-Table to show pathways to all arbitrary states. What must be done instead is have multiple Q-Tables which can be selected based on the goal destination. Then, of course, in this application the robot returns to a "home" state, which is more like the conventional situation where the robot must reach one particular goal state from any arbitrary start state, which can be achieved with one Q-Table.

Discussion

As of yet, this method of having a different Q-Table for each state (possible goal) seems to be the best option. However, this approach makes some concessions which seem worth discussing. Two significant problems of creating unique Q-tables for each potential goal state are wasteful redundancies in memory usage and training time. Take for example two rooms which are right next to each other; most of the path to these rooms are the same path, only branching slightly at the end. With entirely separate Q-Tables for each destination, the pathways have to be stored and trained independently, even though a large portion of the pathway are identical. Surely a better model (which would incidentally better model human intelligence) would create, record, and use one single pathway to reach whatever state is just before the two rooms, and then use more detailed pathway information to find the specific room which is the robot's destination.

One possible approach is to consider a nesting system of states (rooms). As can be seen in the NestedStateDiagram, which is fundamentally the same diagram, we can begin to think of states as being containers which provide access to other states. This would require a system such that for any goal state within a nested state, the robot creates a subgoal of reaching the nest, which is a generalized path applicable for all interior states. To accomplish this the robot will have to temporarily interpret multiple arbitrary ultimate goal states as a single state for the purposes of reaching a nest state. To make this generalization, the robot would have to, for each nest state, maintain a list of interior states (since the interior states are arbitrary and therefore have no discernable pattern).

If we take this approach, however, the algorithm will become too dependant on the conditions, and won't easily be adaptable to other pathfinding scenarios. This is for two reasons: (1) the length of interior states for any given nest state is arbitrary, and (2) the nesting topology of the state space has to be explicitly provided to the robot in advance. Humans have an advantage here: States (locations) are inherently separated and nested by physical space. To properly model this, we would need to give the robot the same awareness. By laying a grid overtop of the blueprint map, and defining where various rooms are, the robot is given the same faculties humans enjoy for the purposes of pathfinding.

At this point I feel that this program may be deviating too far from the exercise of pathfinding via Q-Learning. Rather than using a greedy algorithm combined with past experience to reach a destination, this discussion is now verging on mapping out the environment via exploration and then calculating routes to various destinations. Fundamentally this is a different process and lacks the agency to still be considered "AI." This is not to mention the difficulty in executing this approach. It is for these reasons that I will implement the standard Q-Learning approach with the variation that I will use multiple Q and R tables, one for each potential goal state.

Pseudocode

Top Level

```
getParameters(args);                    // get filename, learning rate, discount, etc.
n = number of unique states;
stateSpace = getStateSpace(filename, n);    // create stateSpace from input file
QTables[n][n][n];                       // n Q tables, all elements initialized to 0
train(stateSpace, QTables, n);          // train all Q tables
test(Q);                                // run tests
```

getStateSpace(filename, n)

```
file = openFile(filename);
// let each line in the file be formatted as:
// stateNumber, edge1, edge2, ...
stateSpace[n][n]; // initialize all entries to -1, represents closed pathway
for each line in file (denoted by 'line') {
    stateNumber = firstValueInLine(line);
    for each remaining value in line (denoted by 'edge') {
        stateSpace[stateNumber][edge] = 0; // opens pathway
    }
}
```

train(stateSpace, Q, n)

```
for each possible goal state (denoted by 'g') {
    R = buildRewardTable(stateSpace, g, n);
    for each possible start location (denoted by 's') {
        do {
            QLearning(Q[g], R, s, g);
        } while (Q[g] has not converged);
    }
}
```

buildRewardTable(stateSpace, g, n)

```
R[n][n] = copy of stateSpace;
for each state in R (denoted by index 's')
    If (open pathway from s to g)
        R[s][g] = 100; // some arbitrary big reward
```

QLearning(QTable, R, start, goal)[1]

```
currentState = start;
do {
    action = nextState = random available action from currentState;
    maxQofNextState = maximum Q value of all possible actions from nextState
    QTable[currentState][action] = R[currentState][action] + gamma*maxQofNextState;
    currentState = nextState;
} while (goal state not reached)
```

[1] http://mnemstudio.org/path-finding-q-learning-tutorial.htm