Artificial Machine Intelligence Search Assignment

Nathan Glover (15164755)

April 28, 2017

Contents

1 Introduction

For this assignment, I have implemented two Informed search strategies.

The first strategy was a **Beam Search** with that utilized heuristic distance values as its method of cutoff allowing it to narrow down its search range to a number defined by the K value passed into it.

The second strategy was a **Memory Limited A* Search** that worked similarly to a typical A* search, but had the added requirement of not allowing any more than 15 nodes in memory at one time.

The classes that make up the foundations of both my search implementations are Vertex, Edge and Graph. These classes offered a large assortment of functionality to my searching methods. The decision to use this structure was one made early during the initial stages of my designs and has slowly been abstracted out with other classes to make it more suitable to each use case. It is important to understand that I designed the classes this way to make them interchangeable with other searching methods. While the structure might not seem so beneficial for the implementations assessed for this assignment, it has a lot of potential for use in other searching landscapes.

2 Solution Approach

2.1 Beam Search

The requirements set on the beam search I needed to implement were that I could expect a K-value in the range of 1-3. The K-value with respect to the Beam search algorithm is the number of paths to be explored and assessed at each depth iteration. The information used in a beam search was found to be a lot more informative and easy to work with as a tree, so I set out to implement Tree and TreeNode classes to compliment my Vertex, Edge and Graph classes.

At each level of the search; a priority queue was filled with potential candidate nodes that were discovered through the previous iterations selected nodes children. The use of a priority queue was beneficial to reducing algorithm complexity as I was able to alter the metric used for priority by simply overriding the **compare()** method used in the queue system.

```java
// Overrides the default compare method
new Comparator<TreeNode>() {
    @Override
    public int compare(TreeNode o1, TreeNode o2) {
        if(o1.getValue().hHeuristic > o2.getValue().hHeuristic)
            return 1;
        else if(o1.getValue().hHeuristic < o2.getValue().hHeuristic)
            return -1;
        else
            return 0;
    }
}
```

Listing 1: PriorityQueue comparator override

This allowed me to use the nodes defined distance heuristic as the value to order the queue in and made things much easier when I needed to remove nodes in the correct order later on in the algorithm.

2.2 Memory Limited A*

The primary requirement for the memory limited A* search was to implement an A* search that can store at most 15 nodes at a time. This meant that all nodes; including the ones waiting idle that are read in from the data lists, count as nodes in memory.

The approach I took to solving this task was to utilize file IO and java class serialization to save and load my node from disk on the fly. Whilst this might not be the most efficient way to handle the problem normally, it does fulfill the specified requirements.

Each Node (Vertex in my case) is initially loaded in one at a time and saved to disk using the **Node-Name.vertex** naming convention. The only references to nodes (vertex objects) stored in memory during runtime is a String value representing the a node's ID, which is added to a queue of items waiting to be explored at any given time. Nodes that are not being processed do not having their string values stored, as it can be extracted from the serialized Node objects when a leaf is explored. This means that I also do not exhaust memory with the strings that are used to access the node objects, as everything can be loaded in from disk on demand.

3 Bugs/Limitations/Problems

3.1 Beam Search

There weren't many if any notable bugs or problems associated with this search. The only thing that caused me troubles while implementing this search was related to finding the alternate paths. I had to create a separate queue of vertex queues that would store any and all of the alternate paths I come across after discovering the best path.

3.2 Memory Limited A*

The first limitation is that the supporting Edge class that represents a link between two nodes (Vertex), is stored within my serialized Vertex node objects. Upon loading a instance from disk of a Vertex, it brings along an Edge reference that is used to extract adjacent node ID's. Whilst I do still load a copy of the correctly serialized class object from disk, I am technically doing extra work to give off the perception that I am not storing these two extra node items as I populate the next level of the search.

While in my testing I did not run into a case where the 15 nodes limit was exceeded, I could see the limit being reached if a graph node had a sizable number of adjacent nodes.

The way I could resolve this problem is by changing the Vertex references within the Edge class to be Strings. Then when edge adjacencies are needed, the vertices's are simply loaded in from disk the same way as normal.

A major problem I had with my A* implementation was when trying to set up a way to print out alternative paths. Once the best path had been computed, I was left with entries in my **stringQueue** of alternative paths. Unfortunately I also used a **stringExplored** list as a way of keeping track of where the node came from and this list was blocking the progress of my alternative paths.

The way I got around this was to simply re-instantiate my **stringExplored** list and add the current working node ID and its parent ID to the list. I found that this, coupled with the A* was enough to ensure that the alternate paths did not loop in circles as the heuristics always pointed it in the correct direction.

4 Example Execution

4.1 Beam Search

```
./beam-search

Sample Input:
Graph File [e.g. graph-info-0.txt]: graph-info-0.txt
Heuristic File [e.g. heuristic-distance-0.txt]: heuristic-distance-0.txt
Source node [e.g. 1]: 1
Destination node [e.g. 15]: 15
K-value [e.g. 3]: 3

Sample Output:
Informed Beam Search Search BEST path:
[1][2][4][8][13][15]

Informed Beam Search Search UNFINISHED paths:
[1][2][4][8][9]
[1][2][4][5][7]
[1][2][4][8][13][14][11]

Informed Beam Search Search ALTERNATE paths:
[1][3][6][10][11][14][15]
```

4.2 Memory Limited A*

```
./alim-search

Sample Input:
Graph File [e.g. graph-info-0.txt]: graph-info-0.txt
Heuristic File [e.g. heuristic-distance-0.txt]: heuristic-distance-0.txt
Source node [e.g. 1]: 1
Destination node [e.g. 15]: 15

Sample Output:
Limited A* Search BEST path:
[1][3][6][10][11][14][15] Cost: 27.0

Limited A* Search UNFINISHED paths:
[1][2][4][8]
[1][2][4][5]

Limited A* Search ALTERNATE paths:
[1][2][4][8][13][15] Cost: 28.0
[1][2][4][8][13][14][15] Cost: 30.0
```