Report A*

Anders Sildnes, Andrej Leitner *students*

***Abstract*—This text answers assignment 1**

***Index Terms*—Stuff**

A* is a local search algorithm. It is similar to DFS and BFS, except that when choosing a new node, it always chooses a node based on an index value.

I. Method

Since A* always chooses either the lowest or highest cost for a node, we desided to implement the queue with a heap. All heaps have the property that the root node has a consistenly higher or a lower value than all of its children. In addition, insertions are $\log n$, so maintaining such a queue is relatively fast. A regular list would cost at least $n \log n$ for sorting.

A* star is a method that should work equally on all *problems*. A problem needs to maintain a list of *states*. Since problem and states is the only variying input, we decided to model these using different classed and objects.

While python does not have the strict notion of abstract classes, we created a superclass "Problem" and an abstract superclass "State". These have general methods to be used in A*. The methods are listed in Figure 1. Explanation:

In the *Problem* class, the network

network is a reference to the (currently) cartesian plane of vertices and edges.

Destructor is invoked after the queue of states is exhausted (clean-up and animation). We included this since some animations will e.g. paint the goal node as a part of a path rather than a final state, etc.

genNeighbour() will genereate all possible successor states from the current state

triggerstart generates a starting state to be used in A*

updateStates paints and animates the next state. In the next state is a direct successor, we often dont need to paint every other state.

In Figure 2, you will find the following:

func,funcargs: when propagating and updating old states, I found it useful if all states had some knowledge of their own function and arguments. This is redundant (since often the function is the same in all states), but easier to read.

betterThanOther is the function equivalent of the <operator, used to compare states.

Anders Sildnes, 4.års student, master informatikk, NTNU.

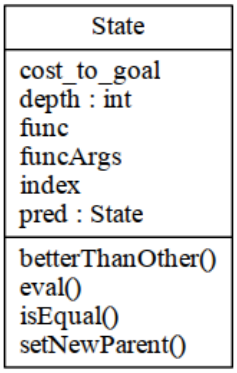

Fig. 1: UML Class diagram for a problem instance

Andrej Leitner, 4.års student, master informatikk, NTNU.

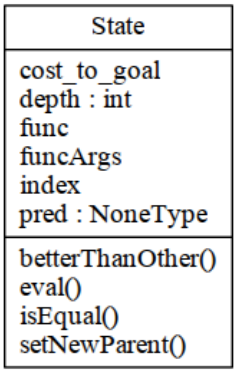

Fig. 2: UML Class diagram for a state instance