AI Programming: Assignment 1

Eirik Vågeskar

October 6, 2015

1 Introduction

This report describes the supplied A* implementation. It is implemented as a function in Python 3, based on the pseudo-codes from the course website and the A* Wikipedia article as of August 2015.

2 Additions

1. *UnsolvableError*: error raised when a problem is found to be unsolvable.
2. *solvability_test*: optional parameter of the A* function. Some problems can be determined to be unsolvable before search. An *UnsolvableError* is raised if the problem is found to be unsolvable.

3 A* Implementation Details and Variables

3.1 Essential Variables and Data Structures of the Implementation

- The implementation operates on objects of some node class suited for the problem, which must contain a *get_successors()* function. Nodes must be hashable and implement functions for equality and sorting to work with the open and closed sets.
- *a_star*: function with parameters *start* (start node), *goal_test* (takes current node as argument), *move_cost* (function for cost of moving from one node to another), and *heuristic_cost_estimate*. Optional arguments: *mode* ("best_first", "depth_first", or "breadth_first"), *solvability_test* (see Additions), and *gui_function* (sends data to GUI).
- *closed_set*: Python set. Makes it possible to check for membership in constant time. Nodes must be hashable.
- *open_set*: Python list-heap for the open set (or agenda). Nodes pushed and popped by the *heappush* and *heappop* functions of Python's *heapq* module. Elements are of form *(priority, g-score, node)*.

- *g_score*: Python defaultdict. defaultdict is an extension of dictionary which returns a default value if a key has no attached value. In this case, the default value is $+\infty$.
- *came_from*: Python dictionary (i.e. hash table). Keeps track of each node's lowest cost parent. See Termination for details.
- *depth_or_breadth_first_priority*: Integer. Used when breadth- or depth-first mode is requested (see Inner loop).
- *nodes_passed_over*: Number of nodes popped from open set and found to have been expanded with an equal or lower g-score.

3.2 Main Loop

The main loop runs as long as there are elements in *open_set*. Otherwise, an *UnsolvableError* is raised. Starts with popping the node with the lowest f-score, called *N*, from the open set. The g-score calculated at *N*'s creation time has been stored along with it, and is used in the next step.

The function checks if *N* is in the closed set, and if so, whether the g-score from this element's creation time is lower than *N*'s currently registered g-score. As explained in Fig. 5 of *Essentials of the A* Algorithm*, new paths with a lower g-score may be discovered after *N* is added to the closed set. In that case, path improvements must be propagated by a re-expansion of *N*. If *N* is not eligible for re-expansion, *nodes_passed_over* is incremented and the loop skips to the next iteration.

If a skip to the next iteration has not been triggered, *N* can be expanded. At this point, the GUI-function is given a dictionary of current data. *N* is tested for being a goal state, in which case the function goes on to return results (see Termination for details). If *N* is not a goal, it is added to the closed set. *N*'s successors are found through the node's *get_successors*-function. Each successor is handled in an iteration of the inner loop.

3.3 Inner Loop

D is a generated successor of *N*. A tentative g-score for *D* is calculated, which is the g-score of N plus the *move_cost* from *N* to *D*. If this score is not lower than the lowest previously calculated g-score for the node, the function skips to the next generated successor.

If the g-score is an improvement, *D* can be added to the open set. *N* is assigned as the value of the key *D* in *came_from* and the tentative g-score is assigned as the value of key *D* in *g_score*.

The priority of *D* is then found. In the case that the mode is best-first search, *D*'s priority is the f-score: a sum of *D*'s g-score and the heuristic cost estimate for *D*. In the case that depth- or breadth-first–search is requested, the priority is determined by the node's order of discovery, which is assigned from the variable *depth_or_breadth_first_priority*. Breadth-first mode requires that the firstly discovered nodes have priority, and so the variable is incremented for each assignment. In the depth-first case, it is decremented in order to give the lastly discovered nodes priority. A *(priority, g-score, node)* tuple is pushed onto the *open_set* heap.

3.4 Termination

When a node qualifying as a goal state is popped from the open set, the solution path and some additional data is returned. This is handled by the function *return_current_data* (which is also used to send data to the GUI). This returns a dictionary with values for the following keys: 'solution' (a reconstructed solution path from the start to goal, explained in next paragraph), 'solution_cost' (g-score of the node just popped from the open set), 'closed_set', 'open_set' (stripped of priorities), 'came_from', 'nodes_passed_over', and 'current' (the current node).

A path to the goal node is reconstructed by the function *reconstruct_path*, which takes the *came_from* dictionary and current node as arguments. It adds the current node to a list, *total_path*. It loops, appending the parent of the last examined node to *total_path*, going on to examine the parent's parent. This goes on as long as there is a parent element for the current node. The start state has no parent, so when no parent can be found, it returns a reversed *total_path*.

4 Generality of the A* Implementation and Domain-Specific Choices

4.1 Arguments for Generality

This implementation is made general by having all domain specifics implemented either as methods of the node objects or as functions passed as arguments to the algorithm.

4.2 Specifics in the 2D navigation task

A class Board has been made for the 2D navigation task. It is possible to move diagonally. The constructor takes the board specification as an argument. Search nodes are instances of the BoardNode class, which simply contains coordinates for a position on the board.

Neighbour generation and the heuristic function are implemented as methods of the Board class. A BoardNode requests the *getSuccessors* method of Board to generate its successors. In order to find the successors, an array containing the the relative coordinates of possible neighbours is then looped through. Any neighbour that is found to be an occupied space or off the board is then eliminated, and a set of actual neighbours is returned.

The heuristic function, which is also used for move costs, is the Euclidean distance between two points in 2D space, $D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.