**COMP 3710 Applied Artificial Intelligence**

**Seminar/Lab 2.**

**A\* Algorithm, and *n*-Puzzle Game**

1. **Objectives**

* Understand how A\* algorithm works
* Use of A\* algorithm for solving 8-puzzle game

1. **Exercise 1**

* Use “Generic Search Algorithm” with BFS (Breadth First Search) to find a solution on the graph in Slide 13.
* At each step, you need to show ExpandedQ and VisitedQ, and selected node, as shown in Slide 13 for DFS.

1. **Exercise 2**

8-puzzle game:

Initial node: Goal node:

1 2 3 1 2 3

4 8 5 4 5 6

7 0 6 7 8 **0**

Note. 0 means empty tile.

* Find all the next nodes from the initial node, with their g-values, h-values with “the number missed tiles” heuristic, and f-values.
* Select the next node to visit, and visit.
* Find all the next nodes with their g-values, h-values with “the number missed tiles” heuristic, and f-values. (Note that you should not include any visited node.)
* Select the next node to visit, and visit.
* Find all the next nodes with their g-values, h-values with “the number missed tiles” heuristic, and f-values. (Note that you should not include any visited node.)
* Select the next node to visit, and visit.

1. **A\* algorithm for *n*-puzzle game**

current\_node = initial\_node;

Mark current\_node as a visited node; // How?

while (current\_node is not the goal\_node)

{

// Expand nodes

for (all nodes, e\_node, that can be expanded (i.e., moved) from current\_node) {

if (e\_node was visited)

continue;

else if (e\_node was expanded before) {

if (the g-value of e\_node is > the g-value of current\_node + cost(current\_node, e\_node)) {

The g-value of e\_node = the g-value of current\_node + cost(current\_node, e\_node);

The parent of e\_node = current\_node;

}

}

else { // e\_node was not expanded before.

Mark e\_node as an expanded node; // How?

The g-value of e\_node = the g-value of current\_node + cost(current\_node, e\_node);

The parent of e\_node = current\_node;

}

}

// Select next node to visit, and visit

current\_node = one of expanded nodes, that has the smallest f-value;

Mark current\_node as a visited node; // How?

}

General example:

ExpandedQ: [ A:{g=8, h=10, p=K}, B:{g=7, h=12, p=C}, D:{g=9, h=11, p=C}, E:{g=10, h=8, p=K} ]

VisitedQ: [ D, F, H, K, C ]

Current node: G:{gvalue=3, parent=J} that is connected to B, F, I with costs 2, 3, 2.

Let’s assume that G is not a goal node.

At the expansion:

For F? Already visited

For I? Not visited, not expanded, and push I:{g=3+**2**, h= , p=**G**} into the queue.

For B? Not visited, but expanded before, and hence 7 (the current g-value of B) > 3 + 2 (the

g-value of G + the cost from G to B) => B is updated with {g=**5**, h=12, p=**G**}

At the visitation of next node:

The one in ExpandedQ, which has the smallest f-value: ???

VisitedQ: {D, F, H, K, C, **G**}

1. **How to implement A\* algorithm for 8-puzzle game**
   * An *n* x *n* board is a 1-dimensional array of *n* x *n* elements.
   * The goal board is [1, 2, 3, 4, 5, 6, 7, 8, 0].

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 0 |

0 means empty tile.

* The following functions are included in this library – <http://cs.tru.ca/~mlee/comp3710/Software/board_game.min.js>
  + make\_initial\_board(n) Return an 1 x *n* x *n* array. *n* could be 3, not 8.

E.g., [1, 3, 7, 4, 2, 8, 0, 5, 6].

* + index\_of\_empty\_cell(board) Return the index of the empty cell, i.e., 0, in board.

E.g., the return value for [1, 3, 7, 4, 2, 8, 0, 5, 6] is 6.

* + Queue MIN priority queue

E.g., var expandedQ = new Queue();

* + - push(id, priority, obj) Push an object into the queue.
    - pop(id) Return the object of id.

The object will be removed from the queue.

* + - get(id) Return the object of id.

The object will NOT be removed from the queue.

* + - popTheHighestPriorityOne() Return the object of the highest priority.

The object will be removed from the queue.

* + - getTheHighestPriorityOne() Return the object of the highest priority.

The object will NOT be removed from the queue.

* + - isIn(id) Return true if an object of id is in the queue, otherwise false.
    - update(id, priority, obj) Update the object of id with different priority and data.
* Example
  + w2\_board\_game\_student.html
  + You can use the f-value as the priority in the queue.
  + The id of a board (e.g., 1 x 9 array) can be the string converted from the array. E.g., [1, 3, 7, 4, 2, 8, 0, 5, 6] -> ‘137428056’ can be the id of the board.
  + var visited\_q = new Queue(); visited\_q.push(get\_id\_of\_node(node), get\_fvalue\_of\_node(node), node);
* For h-values, you can use the hamming distance heuristic (i.e., the number of missed tiles) or the Manhattan distance heuristic. But it is recommended to use the Manhattan distance heuristic.

1. **Assignment**
   1. You will be given roughly 1 assignment or 2 assignments every week to help you understand all the topics in the lectures.
   2. Submission

* Submit a document file for 2) and 3) by email.
  + - Due:
      * 6:00 pm, September 25, 2017
      * Any late submission will NOT be accepted.
    - Total marks: 5
* Submit the application in 3) by email, with the screen shot that shows how your application works. **Please do not share the code.** Any violation could cause a very serious problem.
  + - Due:
      * 6:00 pm, September 27, 2017
      * Any late submission will NOT be accepted.
    - Total marks: 15
      * No completion: 0 mark
      * Correct completion with the hamming distance heuristic (i.e., the number of missed tiles): 13 marks
      * Correct completion with the Manhattan distance heuristic: 15 marks
      * **No partial marks for any code that cannot solve the puzzle game using A\*.** You really need to complete this assignment.