Homework Assignment 1: Search and CSPs

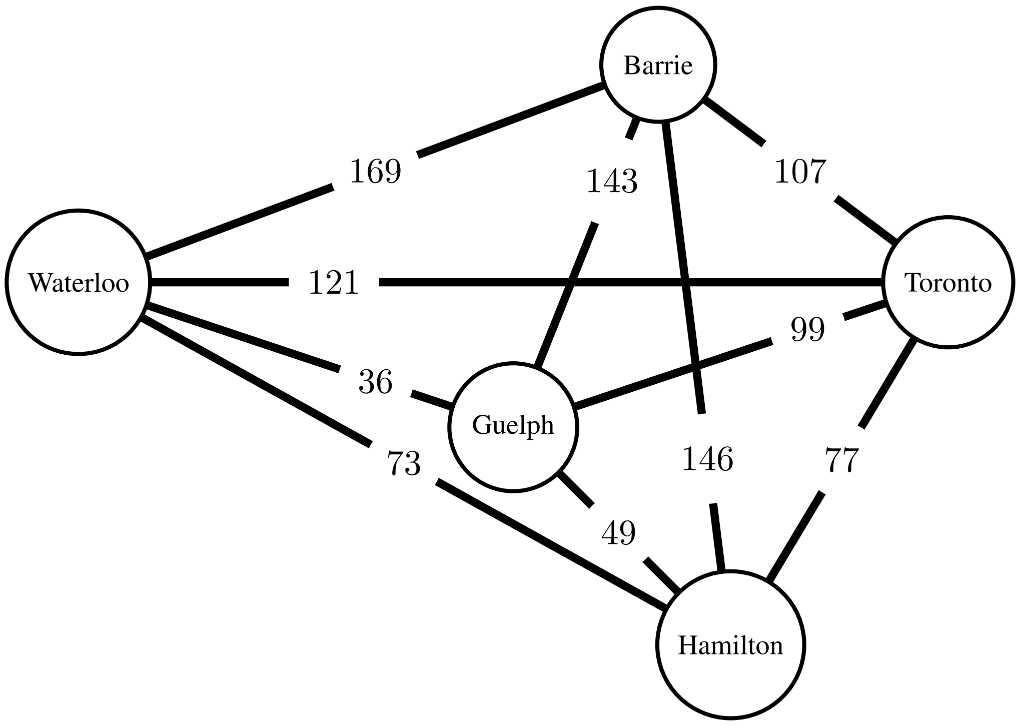
CS486/686 – Spring 2022
Instructor: Jesse Hoey

Out: May 10, 2022
Due: May 25, 2022 at 5pm

Note: The following exercises are to be done individually. No late assignments will be accepted. Please hand in your assignment on the LEARN site for CS486/686 in the dropbox for Assignment 1
`https://learn.uwaterloo.ca`

Informed Search

1. The Travelling Salesperson Problem (TSP) is a famous problem with many applications. As a small example, a salesperson must visit each of the 5 cities shown in the map below. Starting at Waterloo, the problem is to find a route of minimal distance that visits each of the cities only once and returns to Waterloo. You can assume all cities are connected to all other cities (the graph is complete).



Suppose we wish to use algorithm A* to solve an arbitrary instance of the travelling salesperson problem (NOT just the one above, but any TSP).

(a) **[15 pts]** Give a representation of the general TSP problem stated above suitable for A* (that is, describe states and their representation, the initial state, the goal state or goal condition, and the neighbour rules - how to generate neighbours of a node - and their cost). Use the small example above to illustrate your representation, but your representation must be applicable to any (complete) TSP. You can use the letters W,B,G,H,T to refer to the cities by their first letter when referring to the small example above.

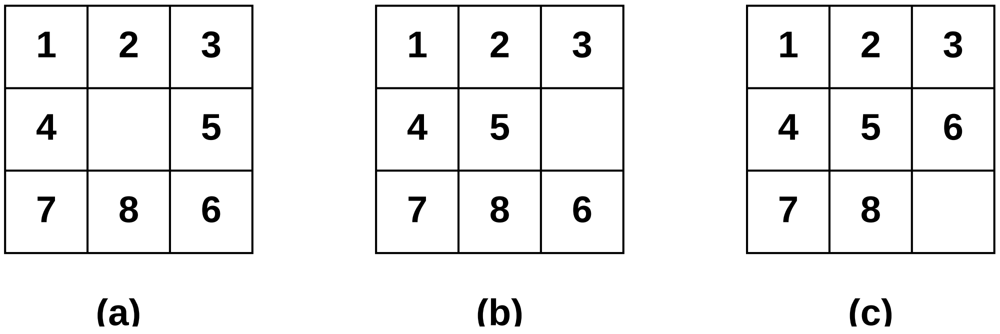
(b) **[5 pts]** Specify the *cost* function. Be precise. Give an example using the small problem above so it is clear what you are proposing.

(c) **[10 pts]** Propose an admissible $h(n)$ (heuristic) function for node n. Be precise. Give an example of computing $h(n)$ for one node using the small problem above.

(d) **[8 pts]** Prove that your heuristic from part (c) is admissible *in general*.

(e) **[10 pts]** Using your heuristic and cost function, show part of the search tree produced by algorithm A* when applied to the small example shown above. Specifically, continue until you have expanded (generated the successors of) **eight** nodes. Break ties by expanding the node with the smallest value of the cost to get to that node. Label each node with its cost value (total path cost to get to that node) and its heuristic value. Also, number the nodes to indicate the order in which they are expanded.

2. Consider the 8-puzzle, which is a simple (one-person) game that we discussed briefly in class. In this game, tiles numbered 1 through 8 are moved on a 3-by-3 grid. Any tile adjacent to the blank position can be moved into the blank position. By moving tiles in sequence we attempt to reach the goal configuration. For example, in the figure below, we see three game configurations: the configuration (b) can be reached from configuration (a) by sliding tile 5 to the left; configuration (c) can be reached from configuration (b) by sliding tile 6 up. Configuration (c) is the *goal configuration*. The objective of the game is to reach the goal configuration from some starting configuration with the fewest moves. Note that not all starting configurations can reach the goal.



Consider the following two heuristic functions that we discussed in class:

- *Misplaced tile heuristic:* number of tiles (exluding the blank) that are misplaced (with respect to the goal)
- *Manhattan distance heuristic:* total Manhattan distance of all the tiles (excluding the blank). That is, for each tile, the Manhattan distance is the sum of the horizontal and vertical distances between its current position and the desired position in the goal configuration.

Suppose that you use the above heuristics in A* search to solve the 8-puzzle game.

(a) **[10 pts]** Which heuristic do you expect to perform best? You do not need to implement the heuristics with A*. Based on the properties of each heuristic, just explain which one you expect to perform best.

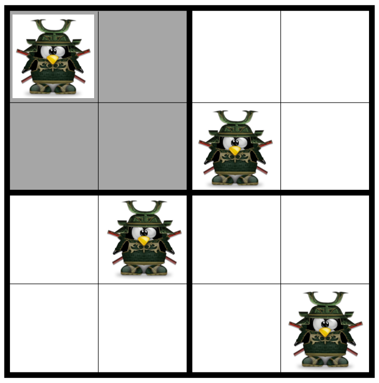
(b) **[10 pts]** We have seen in class that those heuristics are *admissible*. Are they also *monotone*? Give a proof or a counter example for each heuristic.

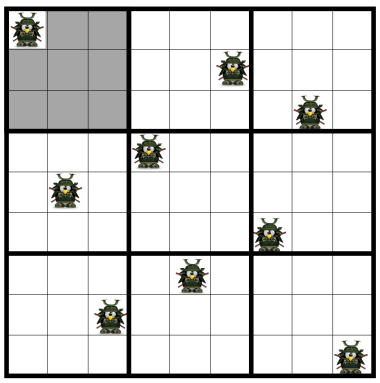
Constraint Satisfaction

3. The N-Queens problem is to place N Queens on an $N \times N$ chessboard (grid) so that no Queen can attack any other Queen. Queens can attack one another if they are on the same row, same column, or same diagonal. Thus, a solution to the N-Queens problem has N Queens on an $N \times N$ chessboard with no two Queens on the same row, same column, or same diagonal.

   (a) **[8 pts]** Formulate the N-Queens problem as a constraint satisfaction problem (CSP) by giving the variables, their domains, and the constraints.

   (b) **[8 pts]** Like the N-Queens problem, the N-Samurai problem is to place N Samurai on an $N \times N$ chessboard so that no Samurai can attack any other Samurai. However, Samurai attack in a slightly different way than Queens. Samurai *cannot* attack on a diagonal, but can attack if they are on the same row or column. Furthermore, Samurai can attack each other if they are within one of the $M \times M$ *blocks* that make up the $N \times N$ grid, where M is integer, $M \geq 2$, $N = M^2$, and the M^2 blocks completely fill the $N \times N$ grid (so they are all aligned and adjacent). This is shown below for $M = 2$ (a 4×4 grid) and $M = 3$ (a 9×9 grid): the darker lines delineate the *blocks*, the top left block is shaded, and Samurai are shown in positions from which they cannot attack each other. If two Samurai were in the same block, or on the same row or on the same column, they could attack one another. Formulate the N-Samurai problem as a constraint satisfaction problem (CSP) by giving the variables, their domains, and the constraints.


$M = 2, N = 4$


$M = 3, N = 9$

   (c) **[12 pts]** In a Sudoku puzzle, the goal is to fill a $N \times N$ grid (where $N = M^2$) so that each column, each row, and each of the N $M \times M$ blocks (as defined above for the N-Samurai problem) contains the digits from 1 to N, only one time each. Some of the squares are initially filled with digits, and none of the initial digits are in conflict with one another. Formulate the Sudoku problem as a single CSP by giving the variables, their domains, and the constraints. Can you formulate a CSP for Sudoku using only unary and binary constraints?

   (d) **[4 pts]** Give a **brief** explanation for how you would build a Sudoku solver using the CSP solver for N-Samurai **combined with** a search strategy. Justify the search strategy you propose. A few sentences suffice. You may need to make a small modification to the CSP solver.