COMP-424: Artificial intelligence, Winter 2022

Homework 1

Due on *myCourses* Tuesday Feb 1, 9:00pm.

General instructions.

- This is an individual assignment. You can discuss solutions with your classmates, but should only exchange information orally, or else if in writing through the discussion board on *myCourses*. All other forms of written exchange are prohibited.
- Unless otherwise mentioned, the only sources you should need to answer these questions are your course notes, the textbook, and the links provided. Any other source used should be acknowledged with proper referencing style in your submitted solution.
- For each problem, you can solve manually, or write a program to help you. You can use a programming language of your choice. You can modify code from other sources if you provide adequate citation; this cannot be code from other students in the class.
- Submit a single pdf of your responses through *myCourses*. You can scan-in hand-written pages. If necessary, learn how to combine many pdf files into one.
- In addition, submit any code developed to answer questions (e.g., Question 3) as a separate file through *myCourses*.

Question 1: Language Generation [20]

We would like to design a simple language generation system which can generate sensible and grammatically correct sentences of English of up to 6 words long. The system is able to generate these five words: *the*, *cat*, *sat*, *on*, *mat*.

Thus, the sentences that we would want the system to generate are (we ignore capitalization and punctuation issues):

the cat sat
the cat sat on the mat

The system also incurs a cost for generating each word, equal to the number of letters the word contains.

a) Formulate the sentence generation process as a search problem, stating each of the parts of the search problem as shown in class.
b) Describe the state space graph of this problem. What does it look like? How many nodes are there in the search graph?
c) Draw the search tree down to a depth of 2 (i.e., the tree has 3 levels in total, including the root.)
d) Trace the run of the search process using the following algorithms for up to 10 search steps. Given multiple states to explore that are otherwise equivalent in priority, the algorithm should prefer to generate the word that comes first alphabetically.
   i. Breadth first search
   ii. Uniform cost search
   iii. Depth first search
   iv. Iterative deepening
e) Reformulate this problem as a local search problem, stating each of the parts of a local search problem as shown in class.

Question 2: Search algorithms [30]

(Adapted from Russell & Norvig)

a) Describe a state space in which iterative deepening search performs much worse than depth-first search (for example, $O(n^2)$ vs $O(n)$).

Prove each of the following statements, or give a counterexample:

b) Breadth-first search is a special case of uniform-cost search.
c) Best-first search is optimal in the case where we have a perfect heuristic (i.e., $h(n) = h^*(n)$, the true cost to the closest goal state).
d) Suppose there is a unique optimal solution. Then, A* search with a perfect heuristic will never expand nodes that are not in the path of the optimal solution.
e) A* search with a heuristic which is admissible but not consistent is complete.

Question 3: Travelling Salesman Problem [50]

Write code in a language of your choice to solve this problem. Hand in the source code with your submission, making clear that the file is/are for Question 3 (e.g., a1q3.py). If you think it will be easier, you are welcome to use external code and libraries with citation, as long as they do not come from current or past students in COMP 424, but please implement your own greedy local search method.

Consider the travelling salesman problem, as described in class. For parts a) – c), write code to generate 100 random TSP instances involving **7 cities**, where cities lie at points in the 2D plane ([0.0,1.0], [0.0,1.0]), sampled uniformly at random. Your code should be correct but will not be marked for style. Use Euclidean distance as your distance measure between cities.

a) Solve each TSP exactly by brute-force search over all possible tours. Record and save these costs. In your report, simply state where in your code this is implemented, and state the mean, min, max, and standard deviation of the optimal tour lengths across the 100 instances.
b) As a baseline, generate a random tour for each of the 100 instances. What is the mean, min, max, and standard deviation of the tour lengths found? Report the number of instances for which the random tour happens to be the optimal solution (may be zero).
c) Implement and apply hill climbing/greedy local search using the 2-change neighbourhood function described in class on the 100 instances, using the randomly sampled start state from part b). What is the mean, min, max, and standard deviation of the tour lengths found? Also report the number of instances for which the algorithm found the optimal solution.
d) Scale up your experiments! Repeat parts b) and c) using 100 random TSP instances involving **100 cities**. Since the number of possible tours is factorial with respect to the number of cities, it may no longer be feasible to compute the costs of the optimal solutions in a reasonable amount of time, so simply omit that part of the table.

(If you have extra time, you can find libraries to help you plot and visualize the solutions found by your algorithms, but this is not necessary for the submission.)

Implementation tips:

- Represent a tour as an array of cities, which are (x,y) coordinates
- Some useful methods you should write should implement:
    - Computing the cost of a tour
    - Computing the cost between two cities
    - Generating the neighbours of a state
    - Generating a TSP instance
- If you're using Python, the following modules may be useful: random, itertools (in particular itertools.permutations), statistics