Computer Department
Faculty of Engineering
Cairo University

CMP402A
Machine Intelligence
Fall 2020

Programming Assignment Document

This document contains 10 problems. Only 8 students are allowed to pick the same problem. This is an individual assignment.

Deadlines: Saturday December 26th 2020 23:59.

Deliverables:

1. **Code**:
    a. Include any code you wrote to solve the problem.
    b. Make sure that your code is well-organized and well-documented.
    c. Only use **"Python"** as your coding language.
2. **Datasets**:
    a. Include any datasets you created or used. (This is only relevant to some problems).
3. **Report (PDF Only)**:
    a. Include your name, section, bench number and problem number.
    b. Briefly describe each attached code (and dataset if any) files and mention the relationship between these files.
    c. Explain how to run the code and mention any required dependencies.
    d. Write any textual answers to the problem.
    e. Briefly explain how you wrote the code and how it helps you solve the problem.
    f. Include your results and conclusions. Any conclusion should be supported by some data or examples presented in the report.
    g. Mention any assumptions you made and the reasoning behind your assumptions.

Note: Some problems have some variables that are unspecified. This usually means that your code should be able to handle different values of these variables. Or you may have to assume a reasonable value or range of values. For example, "Othello(n)" is a generic version of Othello where the board size is "nxn". So your Othello code should handle any board size within reasonable limits. For reporting results, assume a range that is reasonable for your tests. I recommend that you start from the lowest possible value and go up until your resources can no longer handle the problem (e.g. memory limit exceeded or run-time is too long).

Problem 1

Description:

Text segmentation is a common task for natural language processing. It is concerned with splitting text into tokens. For example, “This is a whole sentence.” can be segmented into [“This”, “is”, “a”, “whole”, “sentence”, “.”]. While most of english is “almost” already segmented by white spaces, some languages such as chinese and some text sources such URLs may not have such a helpful property. In this exercise, we will be concerned with segmenting text with no spaces.

Requirements:

1. Write a program to do segmentation of words without spaces. Given a string, such as the URL “thelongestlistofthelongeststuffatthelongestdomainnameatlonglast.com,” return a list of component words: [“the,” “longest,” “list,” . . .].
2. Analyze the performance of your program.

Hint:
It can be solved with a unigram or bigram word model and a dynamic programming algorithm similar to the Viterbi algorithm.

Problem 2

Description:

The travelling salesman problem (TSP) is concerned with finding the shortest closed path through all cities where each city is visited only once. Using informed search, the traveling salesperson problem (TSP) can be solved with the minimum-spanning tree (MST) heuristic, which estimates the cost of completing a tour, given that a partial tour has already been constructed. The MST cost of a set of cities is the smallest sum of the link costs of any tree that connects all the cities.

However, In this exercise, we explore the use of local search methods to solve TSPs.

Requirements:

1. Write a problem generator for instances of the TSP where cities are represented by random points in the unit square.
2. Find an efficient algorithm in the literature for constructing the MST, and use it with A* graph search to solve instances of the TSP.
3. Implement and test a hill-climbing method to solve TSPs. Compare the results with optimal solutions obtained from the A∗ algorithm with the MST heuristic.
4. Repeat using a genetic algorithm instead of hill climbing.

Problem 3

Description:

Othello (also known as Reversi) is a board game where each player seeks to dominate the board by flipping his opponent's pieces. The goal is to have more pieces with your color on the board when the game ends. Initially, the board has only 2 pieces of each color placed diagonally to each other on the center of the board. While Othello is commonly played on an 8x8 board as shown in Figure 1, it can be generalized to an nxn board, which we will call Othello(n).

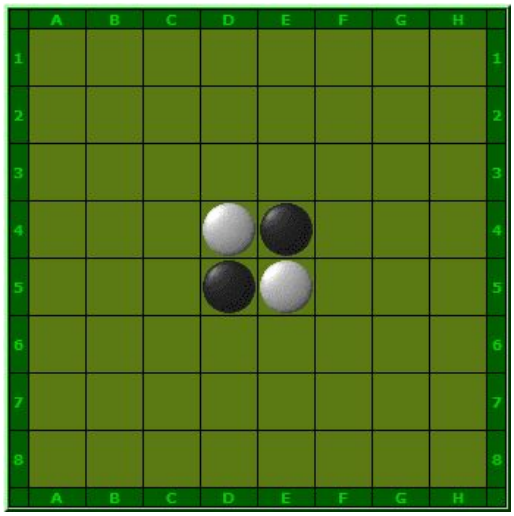

Fig. 1: Initial State of Othello(8) (http://www.iggamecenter.com/info/en/reversi.html)

More details on the rules can be found at: http://www.iggamecenter.com/info/en/reversi.html

Requirements:

1. Construct a general alpha–beta game-playing agent.
2. Implement move generators and evaluation functions for Othello(n).
3. Compare the effect of increasing search depth, improving move ordering, and improving the evaluation function. How close does your effective branching factor come to the ideal case of perfect move ordering?

Problem 4

Description:

Checkers is a board game where each player seeks to capture all of their opponent's pieces or to block their moves. The game ends when one player has no legal moves. Initially, all the black positions on the board are covered (with exception to the 2 middle rows) by pieces where white pieces cover one side and black pieces covering the other side. English Checkers is usually played on an 8x8 board as shown in Figure 1 but it can be generalized to an nxn board, which we will call Checkers(n).

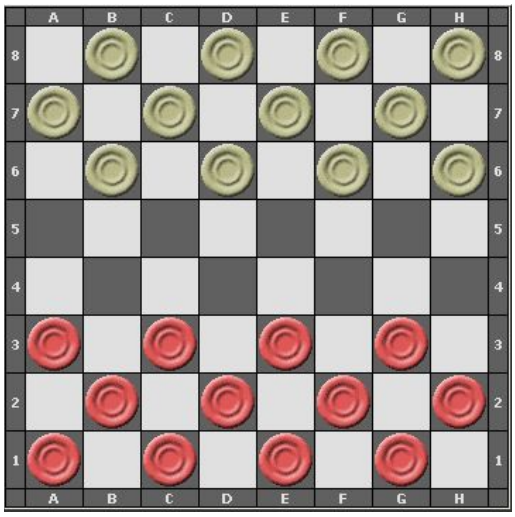

Fig. 1: Initial State of Checkers(8) (http://www.iggamecenter.com/info/en/checkers_eng.html)

More details on the rules can be found at: http://www.iggamecenter.com/info/en/checkers_eng.html

Requirements:

1. Construct a general alpha–beta game-playing agent.
2. Implement move generators and evaluation functions for Checkers(n).
3. Compare the effect of increasing search depth, improving move ordering, and improving the evaluation function. How close does your effective branching factor come to the ideal case of perfect move ordering?

Problem 5

Description:

Backgammon is a popular 2-player board game. Since the available actions are determined by dice rolls, the game relies on both strategy and luck. Figure 1 shows the initial state of a backgammon game.

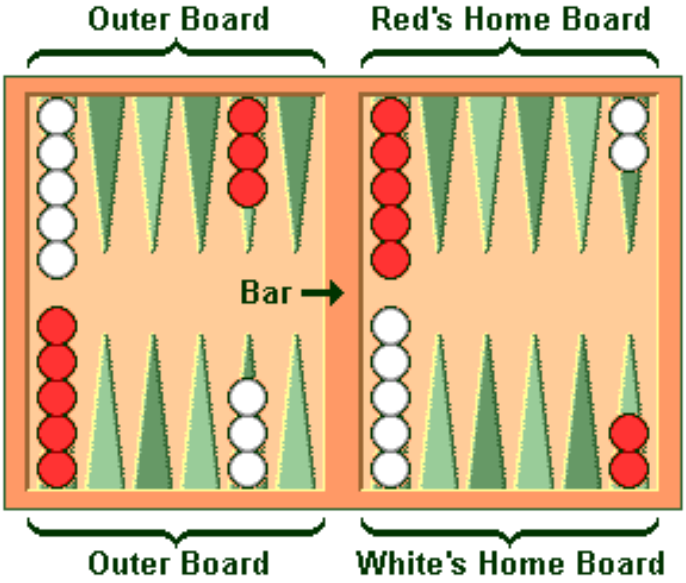

Fig. 1: Initial State of Backgammon (https://www.bkgm.com/rules.html)

More details on the rules can be found at: https://www.bkgm.com/rules.html

Requirements:

1. Construct a general expectiminimax game-playing agent.
2. Implement move generators and evaluation functions for Backgammon.
3. Compare the effect of increasing search depth, and improving the evaluation function.

Problem 6

Description:

Map coloring is a problem concerned with assigning each node a color from a set of colors such that no pair of neighbors have the same color. A k-coloring of a map is a solution of the map-coloring problem using only k colors.

Requirements:

1. Generate random instances of map-coloring problems as follows: scatter n points on the unit square; select a point X at random, connect X by a straight line to the nearest point Y such that X is not already connected to Y and the line crosses no other line; repeat the previous step until no more connections are possible. The points represent regions on the map and the lines connect neighbors.
2. Write solvers to find a k-coloring for each map (for k=3 and k=4) using:
    a. Min-conflicts.
    b. Backtracking.
    c. Backtracking with forward checking.
    d. Backtracking with MAC.
3. Construct a table of average run times for each algorithm for values of n up to the largest you can manage. Comment on your results.

Problem 7

Description:

This exercise explores the quality of the n-gram model of language.

Requirements:

1. Find or create a monolingual corpus of 100,000 words or more. Segment it into words, and compute the frequency of each word. How many distinct words are there?
2. Count frequencies of bigrams (two consecutive words) and trigrams (three consecutive words).
3. Use those frequencies to generate language: from the unigram, bigram, and trigram models, in turn, generate a 100-word text by making random choices according to the frequency counts.
4. Compare the three generated texts with actual language. Finally, calculate the perplexity of each model.

Problem 8

Description:

"n" vehicles occupy squares (1,1) through (**n**,1) (i.e., the bottom row) of an **n**×**n** grid. The vehicles must be moved to the top row but in reverse order; so the vehicle **"i"** that starts in (**i**,1) must end up in (**n**−**i**+1,n). On each time step, every one of the **"n"** vehicles can move one square up, down, left, or right, or stay put; but if a vehicle stays put, one other adjacent vehicle (but not more than one) can hop over it. Two vehicles cannot occupy the same square.

Requirements:

1. Suppose that vehicle **"i"** is at ($\mathbf{x}_i$,$\mathbf{y}_i$); write a nontrivial admissible heuristic hi for the number of moves it will require to get to its goal location (**n**−**i**+1,n), assuming no other vehicles are on the grid.
2. Which of the following heuristics are admissible for the problem of moving all n vehicles to their destinations? Explain.
    a. Sum {h1, …, hn}.
    b. Max {h1, …, hn}.
    c. Min {h1, …, hn}.
3. Build an A* agent for the problem of moving all **"n"** vehicles. Compare its results and performance with Uniform cost search and Best-first search.
4. Try to design a better heuristic for the problem of moving all **"n"** vehicles and compare it with the heuristics in point 2.

Problem 9

Description:

Stylometry is a subfield of linguistics which is concerned with identifying the author behind a piece of text. its successes include the identification of the author of the disputed Federalist Papers (Mosteller and Wallace, 1964) and some disputed works of Shakespeare (Hope, 1994). Khmelev and Tweedie (2001) produce good results with a simple letter bigram model.

In this exercise you will develop a classifier for authorship: given a text, the classifier predicts which of two candidate authors wrote the text.

Requirements:

1. Obtain samples of text from two different authors. Separate them into training and test sets.
2. Train a language model on the training set. You can choose what features to use; n-grams of words or letters are the easiest, but you can add additional features that you think may help.
3. Compute the probability of the text under each language model and choose the most probable model.
4. Assess the accuracy of this technique. How does accuracy change as you alter the set of features?

Problem 10

Description:

This exercise is concerned about the classification of spam email.

Requirements:

1. Find or create a corpus of spam email and one of non-spam mail. Separate them into training and test sets.
2. Examine each corpus and decide what features appear to be useful for classification: unigram words? bigrams? message length, sender, time of arrival?
3. Train a classification algorithm (decision tree, naive Bayes, SVM, logistic regression, or some other algorithm of your choosing) on a training set and report its accuracy on a test set.
4. Assess the accuracy of this technique. How does accuracy change as you alter the set of features?