Lancaster University

School of Computing and Communications

SCC361 AI Course Work#2

Introduction:

Marking Scheme

20% of the mark for the SCC361 module is based on the Course Work#2.

Submission deadline

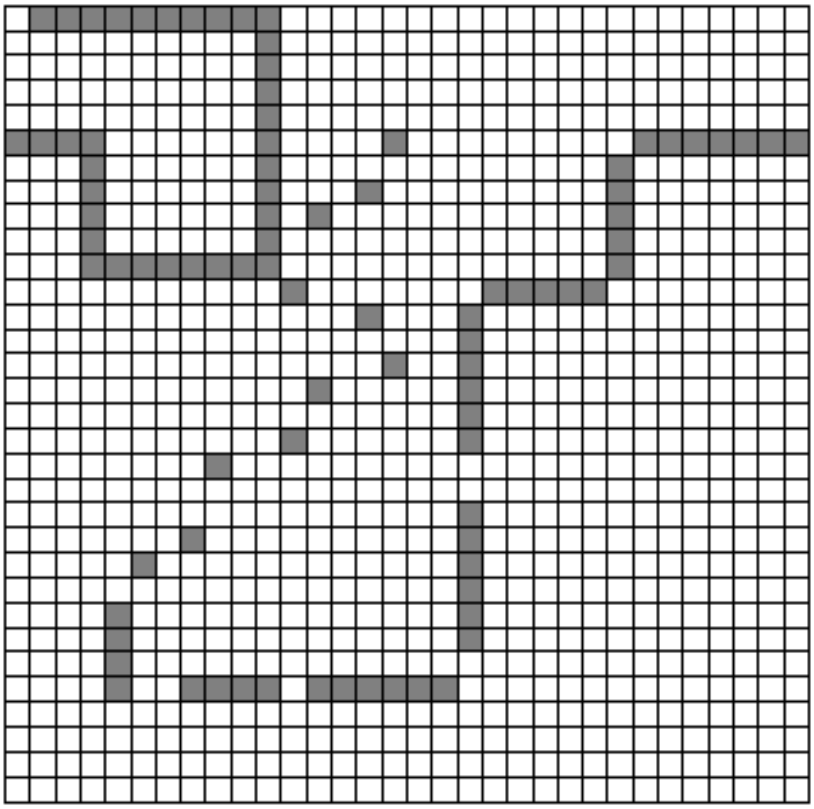
The deadline for submission of your CW is 4pm, Friday, 17 December, 2021 (end of week 10) and should be submitted electronically on Moodle. The cut-off deadline is 4pm, Monday, 21 December, 2021 (with late submission incurring one grade penalty). Submissions after this deadline are not acceptable according to the University regulations.

The implementation should be done in **MATLAB 2020a**. **The codes should be run by pressing MATLAB RUN bottom (without requiring any changes in the codes)**. If your codes consist of more than one .m file, the name of the main file should be "*main.m*". The output should be two plots (as explained below in the CW). Copy your code (including "*main.m*", "*simulate_ant.m*", "*muir_world.txt*") in a folder "CW2_lastname_firstname" (replace lastname and firstname with your names). Submit your file on Moodle.

In the CW#2, you will implement a Genetic Algorithm (GA) to solve a specific problem, and show and analyse your results. These are to be done in **MATLAB 2020a**. GA has been introduced and discussed as part of the week 6 lectures. There have also been two lab sessions about GA in week 6 & week 7 to give you an initial understanding of the GA approach, but this CW is applying GA to a different problem than the ones in the lab.

Problem Definition

In this CW, you will evolve a controller for a simulated ant. Each ant must survive on its own in a world represented by a 2D grid of cells by following trails of food. Each cell in the world either has a piece of food or is empty and the cells wrap-around (so, moving up when in the top row leaves the ant in the bottom row of the grid). Shown below is an environment (called the "John Muir" trail) that consists of a 32 by 32 grid containing 89 food cells (shown in grey).



The ant's position at any point in time can be specified by a cell location and a heading (north, south, east, or west). The ant always starts in the cell in the upper left corner, facing right (east). At the beginning of each time-step it gets one bit of sensory information: whether there is food in the cell in front of the cell it currently occupies (i.e., the cell it would move to if it moved forward). At each time-step it has one of four possible actions. It can move forward one cell; turn right ninety degrees without changing cells; turn left ninety degrees without changing cells; or do nothing. If an ant moves onto a food-cell, it consumes the food and the food disappears; when the ant leaves that cell, the cell is empty. The fitness of the ant is rated by counting how many food elements it

consumes in 200 time-steps. (An ant that consumes 10 cells worth of food receives a fitness score of 10.)

The controller for our ant will consist of a 10-state finite state machine (FSM). At each time step, the ant takes the following actions:

1- Read the sensor value.

2- The controller changes state based on the sensor value/

3- The ant takes an action indicated by the new state (which may result in a change in position).

4- If the ant is in a cell with food, the ant eats the food.

Each of the ten states in the FSM controller has a unique identifier (a number ranging from 0 to 9) and the content of that state can be represented by three digits:

Digit #	Range	Meaning
1	1-4	The action that the ant takes upon entering this state, where 1 = move forward one cell 2 = turn right ninety degrees without changing cells 3 = turn left ninety degrees without changing cells 4 = do nothing
2	0-9	If the ant is in this state and the sensor value is **false** (there is no food in the square ahead of it), then the ant will transition to the state with the unique identifier indicated by this digit.
3	0-9	If the ant is in this state and the sensor value is **true** (there is food in the square ahead of it), then the ant will transition to the state with the unique identifier indicated by this digit.

The ant begins its life with the controller in state 0. The entire genetic material for an ant thus consists of 10 states, each of which is represented by three digits, for a total of 30 digits.

Simulator

We provide you with a simulator function that takes as input the genetic encoding for an ant (i.e. `string_controller` containing 30 digits) and the file containing the environment, and will produce as output the trail that the ant takes and an overall fitness value as below:

```
[fitness, trail] = simulate_ant(dlmread('muir_world.txt',' '), string_controller)
```

The file called "*simulate_ant.m*" on Moodle does this. In the file, the variable "*ant_ori*" specifies the ant's orientation, and 1, 2, 3, 4 represents east, north, west, south respectively.

There is also a file called "*muir_world.txt*" containing the Muir world shown above. The numbers in this file describe the 32×32 grid world, where 0 represents no food and 1 represents food.

What you have to do here: Perform an example run of a test ant (with a genetic encoding of your own choosing or the genetic encoding given below) on the Muir world. Try to understand how the simulator works.

[1 1 3 2 7 6 3 1 5 1 8 0 1 7 6 3 4 6 1 1 0 1 5 8 4 4 9 2 9 3]

Implement Genetic Algorithm

Your main task is to implement with MATLAB a genetic algorithm that attempts to build a better ant through evolution. You cannot use MATLAB's "ga" function, so you have to implement something similar to what you did in the labs in week 6 & week 7. Note that there is no need to change the simulator file (i.e., "*simulate_ant.m*").

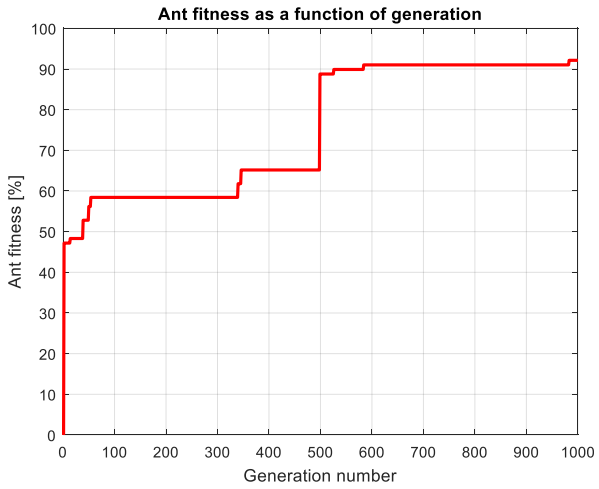
Your algorithm should make use of appropriate crossover and mutation operators. You will need to make many design decisions on how to implement the algorithm and what parameter values to use.

Your code should output a plot showing the fitness score of the most-fit ant in each generation using the following code:

```
hf = figure(1); set(hf,'Color',[1 1 1]);
hp = plot(1:Ngen,100*fitness_data/89,'r');
set(hp,'LineWidth',2);

axis([0 Ngen 0 100]); grid on;
xlabel('Generation number');
ylabel('Ant fitness [%]');
title('Ant fitness as a function of generation');
```

`Ngen` is the number of generations and `fitness_data` is a `1×Ngen` vector of fitness values of the most-fit ant in `Ngen` generation. One possible plot has shown below.



Your code should also output a plot showing the trail of the most-fit ant in the final generation using the following codes:

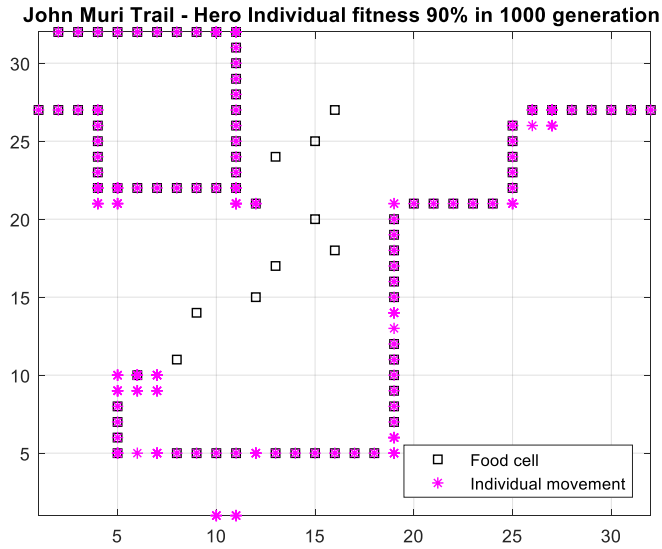
```
% read the John Moir Trail (world)
filename_world = 'muir_world.txt';
world_grid = dlmread(filename_world,' ');

% display the John Moir Trail (world)
world_grid = rot90(rot90(rot90(world_grid)));
xmax = size(world_grid,2);
ymax = size(world_grid,1);

hf = figure(2); set(hf,'Color',[1 1 1]);
for y=1:ymax
    for x=1:xmax
        if(world_grid(x,y) == 1)
            h1 = plot(x,y,'sk');
            hold on
        end
    end
end
grid on

% display the fittest Individual trail
for k=1:size(trail,1)
    h2 = plot(trail(k,2),33-trail(k,1),'*m');
    hold on
end
axis([1 32 1 32])
title_str = sprintf('John Muri Trail - Hero Ant fitness %d%% in %d
generation ',uint8(100*best_fitness/89), Ngen);
title(title_str)
lh = legend([h1 h2],'Food cell','Ant movement');
set(lh,'Location','SouthEast');
```

`Ngen` is the number of generations and `best_fitness` is the fitness value of the most-fit ant in in the final generation. `trail` is a 200×2 matrix, which is the second output of the *simulate_ant.m* function for the most-fit ant in in the final generation. One possible plot has shown below.



Important Notes:

1- You need to implement 3 different selection, 3 appropriate cross-over operators and 3 appropriate mutation operators. Then, try to find the best selection, cross-over and mutation operators by tuning all GA parameters.

2- At the beginning, the code should ask the user for using a default setting (i.e., the setting which you get the best results based on your experiments, for example, Tournament selection, uniform cross-over, swap mutation, etc.) or an optional setting. If the user chooses the default setting, then the code should output 1) the parameters used in the default setting (e.g., selection type, cross-over type, mutation type, number of generations etc.) and 2) the two plots as mentioned above. If the user chooses the optional setting, then the code should ask the user to select what types of selection, cross-over and mutation operators should be used to run the code. The user should also be asked to enter the number of generations. Based on the selected settings, the code should output the two plots as mentioned above.

3- I expect the code runs for around 10 seconds and produces the required outputs.

Marks allocation for this course work:

Structure of the code **20**

The code should be well-structured and easy to understand. You could write your code in a single .m file or have multiple separate function files as well as a single 'main.m' file in which you call the functions.

Correctness of results **30**

Using all settings (mainly changing selection, cross-over and mutation to any combinations (e.g., RWS+Uniform Crossover+Flip mutation OR Tournament+singlepoint crossover+swap mutation) should generate close to optimal results.

Enough comments to understand the code **20**

As you are not asked to submit a report for this CW, the code should be well-commented and provides enough details on methods used/implemented.

Time complexity of the algorithm (in all settings) **30**

The code should generate outputs in around 10 seconds for any settings.

Total: **100**