University of Wollongong

School of Computer Science and Software Engineering

CSCI464/964 **Computational Intelligence** **Autumn 2020**

Assignment 2 **(Due: 6PM, Monday 20 April)** **10 marks**

--- Part 1 (Support Vector Machines, 5 marks) ---

Aim:

This assignment is intended to provide basic experience in solving classification problems with Support Vector Machines (SVMs). After having completed this assignment you should know how to train and test a classifier by using SVM tool package LIBSVM (https://www.csie.ntu.edu.tw/~cjlin/libsvm/)

Assignment Specification:

1. Download "The practical guide" from https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf and read it carefully. You are strongly encouraged to try the examples given in this guide to obtain better understanding of SVM classification. You need to download the LIBSVM library from https://www.csie.ntu.edu.tw/~cjlin/libsvm/ and read README to know how to use it;
2. Three data sets, 1-SpiralData1.txt, data2.txt and data3.txt, are provided in this assignment. They are just the datasets that you have become familiar with in assignment one. For 1-SpiralData1.txt, randomly partition all the 192 data into 60% as training data and 40% as test data. For data2.txt and data3.txt, just use the training and test data predefined in the head of the files.
3. Using the knowledge you learn from the practical guide, for each of the three datasets, train an SVM classifier with the training data and test this classifier on the test data. Try your best to achieve the highest classification accuracy on the test data by carefully tuning the parameters in SVMs.
4. Write a report on this part. It shall include
   a. An introduction of this part of assignment, datasets, and how the training and test data are generated;
   b. Indicate which kind of SVM classifier (for example, linear or nonlinear) is chosen for each dataset and justify your choice;
   c. How you tune the parameters in SVMs (for example, the number of parameters, the range you choose for each parameter, and the number of grids you use, etc.);
   d. The achieved classification accuracy on the training and test data for each of the three datasets;
   e. Detailed discussion and analysis of what you have observed and experienced. Compare the training of SVM classifier with the training of neural networks in assignment one.
   f. Attach a printout of the commands that you use to train and test an SVM classifier with LIBSVM. Group these commands for each of the three data sets clearly.

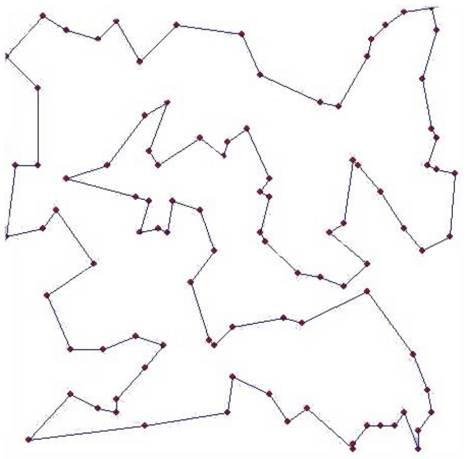
--- Part 2 (Genetic Programming, 5 marks) ---

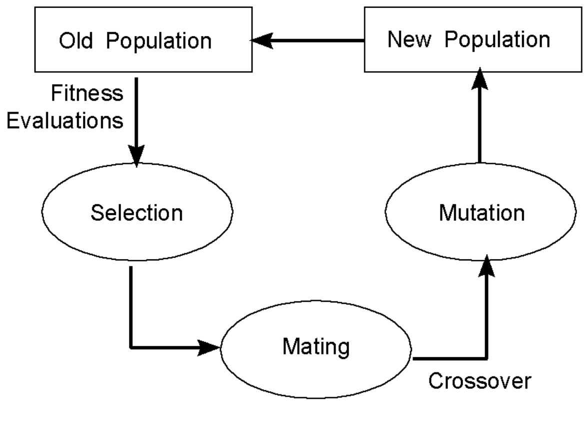
Aim:

This assignment is intended to provide basic experience in solving difficult problems with Genetic Algorithms (GAs). After having completed this assignment you should know how to implement a GA in C++ and find a near optimal solution to hard problems like the Traveling Sales Person Problem (TSP). Note: while doing this assignment please read the messages page of the subject web site regularly to be up to date on any suggestions or changes made to the assignment specification.

Preliminaries:

In the TSP, the goal is to find the shortest distance between N different cities. The path that the sales person takes is called a tour. The following example shows a near optimal tour between 100 cities.





Basic GA Cycle

To test every possible path for an N city tour requires N! math additions. For example, a 30 city tour would be 2.65 X 10^{32} additions. Assuming 1 billion additions per second, this would take over 8,000,000,000,000,000 years. Adding just one more city would cause the number of additions to increase by a factor of 31. Obviously, this is an impossible solution to find mathematically. However, a genetic algorithm (like that shown above) can be used to find a near perfect solution in very little time. The basic concept of Genetic Algorithms is to simulate Darwinian evolution by implementing survival of the fittest. To solve the travelling sales person problem using a GA, first create a group of many random tours in what is called a **population**. These tours are stored as a sequence of numbers representing cities. Second, produce a new population by repeatedly picking 2 of the better (shorter) tours from the population (ie parents) and by combining them using the **crossover operator** to create 2 new solutions (ie children) in the hope that they create a better solution. The **mutation operator** can also be applied to improve the chance of finding a shorter tour. Crossover is performed by picking a one or two random points in the parents' sequences and switching the numbers (representing cities) in the sequence between the points.

The difficulty in using a GA to solve the TSP is in crossing over the cities of parent solutions to produce child solutions. For example, as show below, crossing over the cities in the parents has resulted in City 1 occurring twice in Child 1 and City 5 occurring twice in Child 2. Furthermore, City 1 is missing in Child 2 and City 5 is missing in Child 1. Consequently, a **more complicated crossover operator** must be used to prevent this occurring.

Parent 1	1 2 3 4 5
Parent 2	3 5 2 1 4
Child 1	1 2 3 1 4
Child 1	3 5 2 4 5

Assignment Specification:

To complete this assignment you are to implement a genetic algorithm for solving the TSP Toll Road problem. The TSP Toll Road problem is the same as the general TSP problem except that the cities have types (1:capital, 2:regional & 3:country) and the cost (cents/km) of traveling between two cities depends on their type. **The objective is to visit all cities at minimum cost**. The table below shows the cost of traveling between different types of cities:

	1	2	3
1	10	7.5	5
2	7.5	5	2.5
3	5	2.5	1

For example, to travel between a regional city (type-2) and a capital city (type-1) that are 100km apart it would cost 100km x 7.5c = $7.50 .

Data: Three data files are provided containing 100, 200 and 500 data items. Each data item represents the coordinates (x, y) of each city (in kms) and the city type (1:capital, 2:regional, 3:country).

To assist with this task, **a bare-bones GA written in C++ is provide in the file "ga.cpp"**. However, this GA is for solving problems represented with bit strings and will need considerable modifications and enhancements to make it capable of finding solutions to the TSP. Your completed program should accept the filename of the TSP data file as a command line argument. If no filename is given your program should ask the user to enter the filename via the keyboard. The output of your program should show the minimum tour distance achieved by each generation. However, if this produces more than two pages of output, in your final program, modify your code so that every 5th, 10th or 20th generation is displayed. **To receive full marks for this assignment the following steps must be completed. (Note: these steps do not have to be implemented in the order given. Step 1 is worth 4 marks. Step 2 is worth 1 mark, Step 3 is worth 2 marks and Step 4 is worth 3 marks.)**

Step 1: To implement the TSP on the code in the file ga.cpp, begin by writing a function to read a data file containing the locations of the cities and their type into a dynamically allocated array. Then alter the Init(), Crossover(), Mutate() and EvaluateFitness() functions appropriately to handle the TSP Toll Road problem. **You are encouraged to design your own algorithms for these functions to get your GA performing optimally** (see Step 4). However, to help you get started, the following suggestions are offered.

Init() must be modified to load the initial pop members with random cities. Cities are represented with the numbers 0..n-1 where n is the number of cities in the data file. **Make sure no city occurs more than once in each member**. (Note: the first number in the given data files is the number of cities in the file. Use this number to appropriately allocate the size of member arrays etc. so your code will run on any of the given data files. City locations are represented by (x,y) coordinates (ie integers: 0..999).)

Crossover() can be implemented in a variety of ways. A search of the internet should reveal some useful examples. **You should make sure that the crossover operation does not result in any offspring containing duplicate cities.** This can be done by locating any duplicate cities and replace each duplicate city with a missing city. (But which missing city?)

Mutate() can be implemented by swapping two or more randomly selected cities in the member. Feel free to experiment with this to obtain increased performance.

EvaluateFitness() should calculate the fitness by adding all the distances between visited cities times their cost to get the tour cost. This will result in the fittest cities having the smallest cost. Thus you may have to modify how the BestMember is obtained and how parent selection is done. (Step 3 has more info on this.)

Step 2: To avoid the large expense of calculating the same distances during each fitness evaluation, provide an n x n lookup table of the cost of traveling between cities where n is the number of cities to be visited. Thus, when fitnesses are calculated the cost of traveling between two cities can be looked up from the table more quickly.

Step 3: involves providing your GA with two parent selection options. To do this, implement roulette wheel selection and provide a global constant to switch your GA between Tournament or Roulette Wheel selection. The following algorithm is offered as a suggestion for implementing roulette wheel selection:

```
Copy fitnesses into a temp array
Subtract the minimum pop fitness from each fitness to normalise them
Subtract each normalised fitness from the normalized maximum pop fitness to get the scaled fitnesses
Generate a random number between 0 and the sum of all the scaled fitnesses
Set TempSum to 0
for i=0;i<PopSize;i++){
    TempSum += Fitness[i]
    if(TempSum>RandomNumber) return i;
}
```

This should work, however better performance may be achieved by scaling the fittnesses such that the fittest member occupies no more than 2 times as much space on the roulette wheel as members with average fitness.

Step 4: Run your GA on all three data sets and experiment with different parameters (e.g crossover & mutation types and mutation & crossover probabilities) so that your GA finds the cheapest path in the minimum number of generations. Write up the results in your report.

Step 5: Now modify your crossover and mutation operators so that the amount of change they cause to individuals becomes less as evolution progresses. Describe the modifications you have chosen to make in your report. **Provide a brief comment block at the bottom of your program** to explain the measures you have taken and the parameters you have chosen to achieve your result. Submit your code with your optimal parameters in place so your optimal solution can be demonstrated. Make sure the output from your program does not occupy more than one or two pages. (Note: it is ok to just print the fitness of every 5th or 10th or 20th generation.)

The above is to be taken as a guide only. It is advised to do research on GAs and the TSP problem and make any changes you think fit to improve the performance of the GA on the TSP problem. Please provide references if you use techniques found via your research. **Your report should contain sufficient description, discussion and analysis**, in particular,

(1) **Details** of each step (above) of your implementation and any improvements.
(2) The settings, results and performance (**graphs**) of your GA based on your experiments.
(3) Attach a printout of your code. Ensure it is neat, well commented and easy to understand.

Submit:

Submit your program on UNIX via the submit command **before the deadline** and hand in your report with a cover page to Room 3.219 **before the deadline**.

For part 1: No need to submit any code;

For part 2: Before submitting your code check the format to ensure the format and newlines appear correct on UNIX. (Marks will be deducted for untidy or incorrectly formatted work.) To avoid formatting problems avoid using tabs and use 4 spaces instead of tab to indent you code. Make sure your file is named: **tsp.cpp**.

Put both part 1 and part 2 into a single report. Do not write two separated reports.

Submit using the submit facility on UNIX ie:

```
$ submit -u login -c CSCI964 -a 2 tsp.cpp
```

where 'login' is your UNIX login ID.

We will attempt to run your program on banshee. If problems are encountered running your program, you may be required to demonstrate your program to the coordinator at a prearranged time. If a request for a demonstration is made and no demonstration is done, a penalty of 2 marks (minimum) will be applied. **Marks will be awarded for a comprehensive report, correct program design, implementation, style and performance.** Any request for an extension of the submission deadline must be made by applying for academic consideration before the submission deadline. Supporting documentation must accompany the request for any extension. Late assignment submissions without granted extension will be marked but the mark awarded will be reduced by 1 mark for each day late. Assignments will not be accepted if more than one week late.