**Sarah Wallis  
Data Structures and Algorithms II  
Project 3  
User's Manual**

**Setup and Compilation**  
1. Download and unzip the submission from eLearning on a Linux box in the multi-platform lab.  
2. The submission includes:

* city-matrix.cpp
* city-matrix.hpp
* distances.txt
* main.cpp
* makefile
* min-distance-controller.cpp
* min-distance-controller.hpp
* Results.xlsx
* test-run.txt (Default input for input redirection for testing)
* two-d-array.cpp
* two-d-array.hpp
* UMLDiagram.png
* UsersManual.docx (this file)

3. Environment: This program has been tested in the multi-platform lab and will run there.

4. Compiling. This program includes a Makefile. At the command line in Linux, type make. The program produces an executable entitled main.

**Running the program.** Issue the command ‘./main’ No command line arguments are required or checked.

NOTE – No input validation was implemented for this program. Program may produce unexpected behavior if the following input rules are not followed.

User input:

* Number of cities to run <int> [10-20]
* How many individual tours are in a given generation <int> (Generation population size ; Tested with small values ~5)
* How many generations to run <int> (Should be kept in range 10,000 – 1,000,000 ; Higher values improve accuracy at diminishing returns and lower efficiency)
* Percentage of mutations in a generation <int> [0,100] (Should be kept high ~90 as lower values greatly reduce efficiency and efficacy)

**Output:** All output goes to the console. Output will be similar to this:

Enter the number of cities to run [10-20]: 10

Enter the number of tours per generation: 5

Enter the number of generations to run: 30000

Enter the percentage of a generation's mutations (0-100]: 90

Running brute force algorithm...

Brute force algorithm finished. Running Genetic Algorithm...

Genetic algorithm finished. Displaying results...

------------------------------

RESULTS

------------------------------

Number of cities run: 10

Brute force approach

--------------------

Optimal permutation: 0 -> 7 -> 3 -> 9 -> 2 -> 6 -> 8 -> 4 -> 1 -> 5 -> 0

Optimal cost: 360.26

Brute force algorithm time: 0.180448s

Genetic algorithm approach

--------------------------

Best genetic permutation: 0 -> 7 -> 3 -> 9 -> 2 -> 6 -> 8 -> 4 -> 1 -> 5 -> 0

Genetic cost: 360.26

Genetic algorithm time: 0.168055s

Percent of optimal cost: 100%

Percent of brute force time: 93.1321%