**CITS3001 Algorithms, Agents and Artificial Intelligence**

**Labsheet 3: Travelling Salesman Problem algorithms – Assessed**

**This lab sheet is worth 3% of CITS3001. You should implement Java solutions to the following problems and you should submit them at https://secure.csse.uwa.edu.au/run/cssubmit by 5pm on Sunday 16 August. In accordance with the UWA Policy on Academic Conduct, you may discuss with other students the general principles required to understand this labsheet, but the work you submit must be the result of your own effort.**

- Implement *exactly* what each question describes. These are deterministic algorithms, and your program is required to produce exactly the right answer in exactly the expected format.
- Sample input data files *TSPxyz.txt* and the expected output *outTSPexpected.txt* is provided. We will test your submission with different data of similar sizes in the same format. The LMS folder also provides a file showing more-detailed workings for one set of data, for debugging purposes.
- A reasonable amount of time will be allowed for your program to run: for the sample data folder you should need no more about fifteen seconds. Submissions that run too slowly will be terminated before they complete.
- Submissions that do not compile will earn zero marks. Submissions that have to be edited before they execute will be penalised, and may get zero.
- Normally your mark will depend only on the results produced by your program, but all programs will be compared for similarity with other submissions and with sources drawn from the Internet.
- All questions to *help3001* please.

Download the Lab 3 folder from the LMS, and complete the two methods in the *NearestNeighbour* class in the code skeleton. You can run the skeleton via the *main* method in the *Lab3* class. You should not modify any of the other classes.

**Submit only the file *NearestNeighbour.java* by the due date above.**

1. Implement the nearest neighbour algorithm, from Page 24 of Lecture 4. Your method should try starting from every city and return the best tour found.
2. Implement a simple hill-climber to improve your solutions. Apply 2-OPT from Page 38 of Lecture 4 iteratively for the improvement step; at each iteration, retain the change which gives the biggest improvement. Return the current tour when it cannot be improved any further.