1. Module number	*SET10107*
2. Module title	*Computational Intelligence*
3. Module leader	*Neil Urquhart*
4. Tutor with responsibility for this Assessment Student's first point of contact	*Neil Urquhart* E: n.urquhart@napier.ac.uk T: 0131 455 2655
5. Assessment	*Practical*
6. Weighting	*60%*
7. Size and/or time limits for assessment	*The report should be no longer than 6 pages as specified in the coursework description.*
8. Deadline of submission Your attention is drawn to the penalties for late submission	*The report must be handed in* ***by the Friday of week 12*** *(please check the closing time of the School Office)***.** *Each student will be required to make an informal demonstration of their work during weeks 11 or 12. No demonstration may result in a mark of 0.*
9. Arrangements for submission	*The report will be submitted to the School Office.*
10. Assessment Regulations All assessments are subject to the University Regulations**.**	*No exemptions*
11. The requirements for the assessment	*See below.*
12. Special instructions	.
13. Return of work and feedback	One to one verbal feedback will be offered during the demonstration session. Written feedback will be made available within 3 weeks of submission.
14. Assessment criteria	See attached. Please note that checks for plagiarism will be made on electronic submissions. Students may be required to attend a further demonstration if there exists doubts as to the authorship of work.

1. Introduction

For this coursework you are required to investigate the *bicycle courier problem*, described below, then produce a report describing your attempts to solve the problem.

You should initially conduct research to establish which techniques have been used to solve the problem (or similar problems) previously. You should then design a problem solver. Some Java code is supplied to assist you. Although we encourage the use of Java, you may use any programming language and libraries available, providing that the final programme can be demonstrated within the normal JKCC clusters used for the practical. Any code libraries used should be acknowledged in your report.

Using your chosen technique you should attempt to solve ***all*** the problem instances described in the problem description, collecting and *analysing* results. Finally in your conclusions you should critically evaluate the effectiveness of your approach and your results.

2. Report specification

Your report **must** be formatted as follows, 12 pt Arial text should be used throughout. The entire document should be no more than 6 A4 sides in length (including any tables or diagrams). Where you reference the work of others you should use citations in your text and include a list of references at the end of your report. You **must** include all of the sections in the table below using the headings given. You should pay attention to the presentation of your report. Marks may be deducted for spelling, grammar and any other presentational issues.
Note that the guidelines for the length of each section are approximate. However, the report must be no longer than 6 pages in total.

Section	**Description of content**	**Suggested Length**	**Mark**
Abstract	1 Paragraph summarising the content of the report including a summary of the results.	0.25 page	/5
Introduction	A description of the problem to be solved.	0.25 page	/5
Previous Work	Discuss at least one previous published attempt at solving this or a similar problem. Critically comment on the method used and results obtained.	0.5 page	/5
Methodology	Discuss the method(s) that you will use to solve the problem. Describe the implementation used and any APIs/Libraries used to assist.	1 page	/10

Experimental Plan	Present an experimental plan to be used to investigate your approach.	1 page	/10
Results	Present your results along with a basic statistical analysis.	2 pages	/10
Conclusions	Critically analyse and reflect on the approach that you have taken. How do your results compare to those published? If your results are not as good as envisaged would you adopt a different approach to solving the problem.	0.5 page	/10
References	Reference any websites, blogs or published papers consulted whilst writing the report.	0.5 page	/5

3. Demonstration

You will be required to give a short demonstration to the teaching team. In this, you should show your problem solver running and discuss the approach that you took. **There are no marks associated directly with the demonstration, but failure to demonstrate will result in a mark of 0 being awarded for the coursework.**

4. Deliverables

The 6 page report to the specification outlined above.

5. The Bicycle Courier Problem

A bicycle courier has an innovative flexible pricing system that it is hoped will appeal to customers. The key to making money with this pricing system is to schedule each day's deliveries so as to maximise your income for that day.

Every day, the courier receives a list of delivery jobs. Each job is specified by a job id, a pickup location, a pickup time, a drop-off location and a sliding scale of payments such that you get paid more for faster deliveries. The sliding scale of payments is specific to each job and is defined in the problem.

Your task is find a schedule that deliver all jobs while maximising the amount of money earned. So, e.g., you might choose to deliver package A at a later time because delivering package B earlier pays you more money.

You also have a map specifying all of the pickup and dropoff locations as well as the time (in minutes) required to move between locations (assume that you can travel directly between any two locations).

A package cannot be picked up before the specified pickup time, also, assume that on your bicycle you can only carry one package at a time. In a valid solution all packages must be delivered even if the reward for delivery if £0.

The problem instances are specified as text files, and Java source code is provided to load the data into data structures.

The following problem instances should be investigated

Problem1.txt	10 Jobs
Problem2.txt	20 Jobs
Problem3.txt	30 Jobs
Problem4.txt	40 Jobs
Problem5.txt	50 Jobs

See section 6 for a worked example of solution.

Hints

The problem can be solved using a number of different techniques. For example, it can be solved using A*, using an Evolutionary Algorithm. However, you are not restricted to these and may use any appropriate method, with justification.

Some hints for A* and Evolutionary Algorithms are given below:

Hints for using A*:

You need to use a modified version of the standard A* algorithm. The search is over a graph that specifies jobs; the goal is to visit each **job** once. The search terminates when all jobs have been visited.

a) Think about the **neighbourhood** of any job: this will be given by the set of jobs that have not yet been visited
b) Think about the **cost** of going from one job to another: A* returns a *minimal* cost solution : you want to find a solution that *maximizes* profit. You can do this by considering the maximum possible revenue and thinking of the cost of a solution as the *loss of revenue*. For example, if for a given solution you could potentially earn a maximum of £100, but the solution found yielded only £75, so you can consider this solution to cost £25 (your lost revenue).

   This means you can compute the cost of completing each job for A* as being the difference between the payment you receive for the job and the maximum you could have obtained for the job. Jobs that are completed on time cost zero.

c) Think about the **heuristic estimate** of the cost of completing the remaining jobs from any given node. This is the key to a good algorithm! From a given node in the search, you want to estimate the cost of delivering the remaining jobs. Remember it's just an estimate – there are many heuristic cost functions you can come up with - you might want to try out several as part of your coursework (e.g. think about estimating the profit/loss you might make on the jobs not yet delivered ....)

Hints for an Evolutionary Algorithm:

a) First, you will need to design a representation that specifies a solution. You might think about using a *permutation* to specify the order in which the jobs are undertaken and evolve the best permutation.
b) If you use a permutation representation, you will need to consider the crossover and mutation operators carefully. Both must result in a valid permutation. There are many operators that are possible - you will need to do some research : look up permutation operators for travelling salesman problems. A good resource is the slides available here http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html : (Chapter 3, genetic_algorithms.ppt, slides 31-48)
c) Develop a fitness function: the function can either consider profit (maximize) or loss (minimize). Make sure your selection operator correctly identifies if solution A is **better** than solution B!!

6. A Worked Example

Imagine there are 4 locations, A,B,C,D with a matrix specifying the travel time between each pair:

	A	***B***	***C***	***D***
A	0	15	25	10
B	15	0	10	20
C	25	10	0	13
D	10	13	20	0

With this map you might have the following list of deliveries to perform for the day:

Job	**Collect Location**	**Deliver Location**	**Time Available For Collection**	**Money Earned**
1	A	C	08:00	[8.30, £25], [9.30, £20], [10.00, £10], [11.00, $5]

2	B	C	10:00	[10.30, £25], [12.00, £5]
3	C	D	09:00	[9.05, £50], [9.30, £25], [10.00, £5]

This table specifies that you have to do three delivery jobs, Job1, Job2 and Job3. Job1 is a pickup at A that is available for pick up at 8.00 . The package must go to C. If you deliver it by 8:30 you will earn £25, between 8:31 and 9:30 you will earn £20, between 9:31 and 10:00 you will earn £10, between 10:01 and 11:00 you will earn £5, and any time 11:01 or later you will earn £0.

Note that you have to eventually perform all deliveries, and that if you miss the job's last deadline (i.e., 11:00 for Job1, 12:00 for Job2, and 10:00 for Job3 in the above example) you will earn £0 but are still obligated to make the delivery.

A possible schedule might be to arrive at A at 8:00 to do Job1 (note Job1's package isn't available before 8:00), deliver it to C at 8:25 (the map specifies that it takes 25 minutes to get from A to C). Then wait there until 9:00 to pickup Job3's package, deliver that to D at 9:20. Then proceed to B arriving at 9:33, wait until 10:00 to pickup Job2's package, and then finally deliver that to C at 10:10. This delivery schedule will earn you £25+£25+£25 = £75.

7. Useful Java Code.

Two classes are supplied to you to assist with this coursework:

- Job.java
- Problem.java

Job.Java

We can hold details of each delivery in a simple Java class (note that it is best practice to declare properties as private, but in this case to aid clarity they have been left as public):

```java
public class Job {
	int id;
	int pickup;
	int setdown;
	int available;
	int[][] payments = new int[4][2];
}
```

Pickup and setdown represent the ids of the locations to start and end from. Available represents the time from which the package is available (Note that we will use an abstract notation of time commencing at 0). The array payments contains a set of times and payments as follows:

Time 1	Time 2	Time 3	Time 4
Payment 1	Payment 2	Payment 3	Payment 4

If we deliver before or at<time 1> we receive <payment 1> if we deliver after <Time 1> but before or at <Time 2> we receive <Payment 2>. For instance:

100	150	200	250
64	32	16	8

If we deliver at 75 we would receive £64, we deliver at 199 we will receive £16. If we deliver after 250 we will receive nothing.

Problem.Java

Problem.Java can be used to hold an instance of a problem instance. Consider the following example:

```java
public static void main(String[] args) {

            Problem.loadProblem("Problem1.txt");

            Job[] myJobs = Problem.getJobs();

            //Shuffle the jobs
            Collections.shuffle(Arrays.asList(myJobs));

            System.out.println("Delivery order ");
            for(Job j: myJobs){
                  System.out.print(j.id +",");
            }
            System.out.println();
            System.out.println ("Reward =" +Problem.score(myJobs));
      }
```

The problem is read in from the text file using the method Problem.loadProblem(String filename). Once loaded an array of job objects can be obtained using Problem.getJobs() . You need to find the optimum ordering of jobs, the method Problem.Score(Job[] j) will calculate the total reward payable if the jobs were undertaken in the order specified in an array j . The above example illustrates this by randomly sorting the array and then calculating the reward if delivered in that order.

When writing your own solver you will need to know the time it takes to travel between the various pickup and setdown points. Problem provides a method that will return the time taken to travel between two points

```
Job j1 = myJobs[0];
Job j2 = myJobs[3];

int time = Problem.getTime(j1.setdown,j2.pickup);
```

getTime() takes two location IDs (as contained in Job.setdown or Job.pickup) and returns the time taken to travel between the two locations.