Acquired Intelligence & Adaptive Behaviour
A Hill Climbing GA
Lab Session 3

Goals: To implement a population of hill climbers to solve a resource allocation (or knapsack) problem To design a suitable fitness function and selection method. To investigate the effects of mutation rate, etc.

Background: Hill climber may be used to find solutions to a wide variety of problems. Each hill climbing individual increases its fitness through trial and error e.g.,

1. Create a random individual
2. Change (mutate) the individual
3. Measure fitness. If it is worse then keep original
4. Goto 1

We will implement a population of such hill climber all evolving independently. The problem is as follows:

The knapsack (KP) problem is an example of a combinatorial optimization problem. It is concerned with a knapsack that has positive integer volume (or capacity) **V**. There are **n** distinct items that may potentially be placed in the knapsack. Item **i** has a positive integer volume **Vi** and positive integer benefit **Bi**. In the most basic form of the problem we will consider there are only one of each item available (0-1 KP). The goal is to maximize value,

$$\Sigma_i^N B_i$$

subject to the constraint,

$$\Sigma_i^N V_i \leq V.$$

For example suppose we have a knapsack that has a capacity of 20 cubic inches and N=10 items of different sizes and different benefits. We want to include in the knapsack only these items that will have the greatest total benefit within the constraint of the knapsack's capacity.

Item	a	b	c	d	e	f	g	h	i	j
B	5	6	1	9	2	8	4	3	7	10
V	3	2	4	5	8	9	10	1	6	7

To solve this, our Genetic Algorithm will require the following components:

1. An individual encoding the information solutions to the task

2. A **genotype → phenotype** mapping. How should the genotype be interpreted as encoding a solution to our problem? This is analogous to the development of an organism from birth to adulthood. However, for this current problem, this should turn out to be quite trivial.
3. A **fitness function**. We need a way to evaluate how good each phenotype is as a potential solution to the card-sorting problem. How might this be implemented?
4. A method for **mutation**. Is it necessary to allow random changes in the offspring produced by reproduction in order to maintain variability. In this example, we will design our algorithm first, before attempting to implement and test it using Matlab.

Task 1: Write down a pseudo-code algorithm (i.e. a rough sketch) which combines the components described above. This should be based around a loop.

Task 2: Code a single hill climbing individuals to solve the above task. Implement your algorithm in full and run it for at least 100 *generations* (i.e. repetitions of the algorithm). Recording the fitness at each generation. Plot the fitness versus the generation number. Have you found a solution? Try changing the mutation rate and observe the effects.

Task 3: Code a population of hill climbers that attempt to solve the task in *parallel*.
How many individuals are successful for each run. With a brute force method, there would be 10^2=1024 possible genotypes to evaluate. Is a population of hill climbers less or more computational expensive? What happens when you make the problem bigger or change the problem.

Task 3: Demonstrate that a local minima exist. Can you find suboptimal solution that get worse with every mutation but is not the globally optimal solution.

Tips:
1. Code a single vector for one individual and use binary digits (0 or 1) for each gene.
2. The fitness function should return a *single number* which quantifies how close to the ideal solution a phenotype is.
3. Mutate the single individual by randomly flipping a gene i.e pick a random gene by selecting a random number between [1,10] and then set (1->0 or 0->1)
4. Overwrite the current individual if the new solution is better
5. For a population of hill climbers code many individuals in a matrix and keep fitness values in vector (see lecture notes)