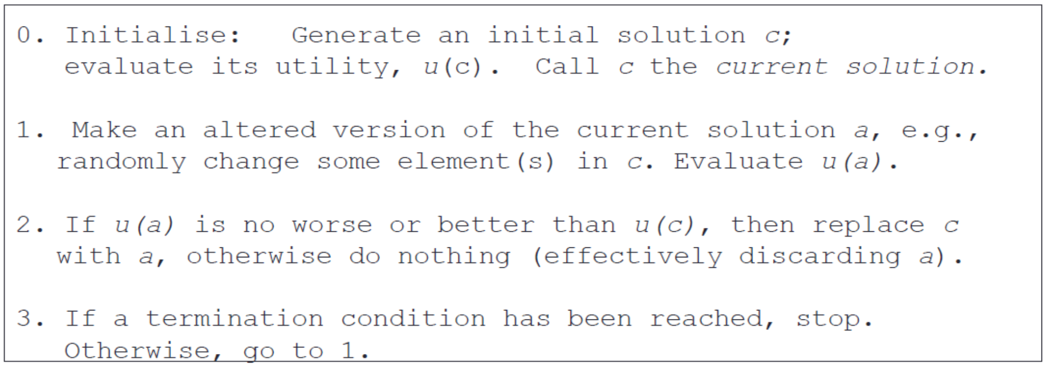
BioComputation

Worksheet 0: Simple Random Hillclimber

In a computer language of your choice, implement a simple random hillclimber as below:



```
0. Initialise:   Generate an initial solution c;
   evaluate its utility, u(c).  Call c the current solution.

1.  Make an altered version of the current solution a, e.g.,
   randomly change some element(s) in c. Evaluate u(a).

2. If u(a) is no worse or better than u(c), then replace c
   with a, otherwise do nothing (effectively discarding a).

3. If a termination condition has been reached, stop.
   Otherwise, go to 1.
```

A solution could be represented by a data structure consisting of an array of N variables and a performance value. For example, in C:

```
typedef struct {
        int variable[N];
        int utility;
} solution;

solution individual;
```

Or in Python to achieve the same thing:

```
class solution:
        variable = [0]*N
        utility = 0

individual = solution()
```

The initial individual solution is created randomly. Make sure you know how to both seed and subsequently call a random number generator and then fill in the variables of your solutions, eg, say with N=10 in Python:

```
for j in range (N):
        individual.variable[j] = random.randint(0,100)

individual.utility = 0
```

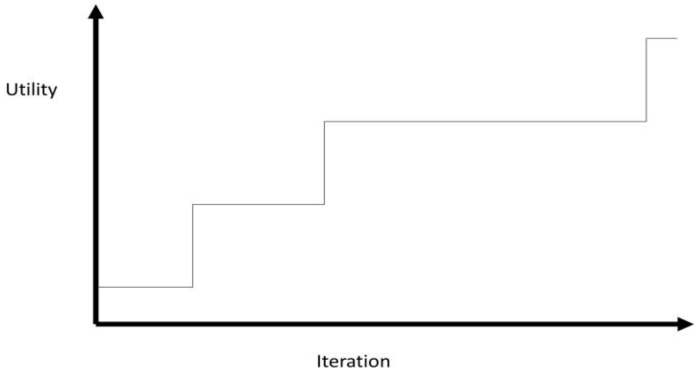
A very simple test function is one in which the utility of a solution is equal to the sum of the variables it represents (you can find and try others afterwards). Use this to test your code – write a test function that is passed an individual and returns the utility value.

```
def test_function( ind ):

        utility=0
        for i in range(N):
                utility = utility + ind.variable[i]
        return utility
```

After that you need to loop creating a random variation to the current solution and testing it, eg:

```
newind = solution()

for x in range (LOOPS):

        for i in range(N):
                newind.variable[i] = individual.variable[i]
        change_point = random.randint(0, N-1)
        newind.variable[change_point] = random.randint(0,100)

        newind.utility = test_function( newind )

        if individual.utility <= newind.utility:
                individual.variable[change_point] = newind.variable[change_point]
                individual.utility = newind.utility
```

It would be nice to put the utility of the individual at the end of each loop in a file so that you can load it into Excel or something similar to see the progress over time. The braver amongst can look into Pyplot. You're after a graph that looks a bit like this:


Utility
Iteration