**CSCE 451 Lab2**

**Due July 15 at 3:00pm**

**A simple Simulated Annealing implementation in Python**

In this lab we implement and investigate, the simulated annealing algorithm for numerical optimization.

**Task 1: Understanding the Algorithm**

We use the same algorithm described in the lecture. Following is the same algorithm with more details for the actual python implementation. Please review the code and make sure you understand it.

Simulated Annealing:

**Let** *s = s0*

**For** *k=0*  **through** *kmax* (exclusive):

*T* := temperature (k∕kmax)

Pick a random neighbour, snew := neighbour(s)

**If** P(E(s),E(Snew),T) ≥ random(0,1):

S := snew

Output: the final state s

**Task 2: Basic but generic Python code**

Let us start with a very generic implementation. Please read the code and make sure you understand the main function. Once you feel comfortable with the code create a python file ( or any other way you usually write your python code) and paste code there.

**def** annealing(random\_start,

cost\_function,

random\_neighbour,

acceptance,

temperature,

maxsteps=1000,

debug=**True**):

*""" Optimize the black-box function 'cost\_function' with the simulated annealing algorithm."""*

state = random\_start()

cost = cost\_function(state)

states, costs = [state], [cost]

**for** step **in** range(maxsteps):

fraction = step / float(maxsteps)

T = temperature(fraction)

new\_state = random\_neighbour(state, fraction)

new\_cost = cost\_function(new\_state)

**if** debug: print("Step #**{:>2}**/**{:>2}** : T = **{:>4.3g}**, state = **{:>4.3g}**, cost = **{:>4.3g}**, new\_state = **{:>4.3g}**, new\_cost = **{:>4.3g}** ...".format(step, maxsteps, T, state, cost, new\_state, new\_cost))

**if** acceptance\_probability(cost, new\_cost, T) > rn.random():

state, cost = new\_state, new\_cost

states.append(state)

costs.append(cost)

*# print(" ==> Accept it!")*

*# else:*

*# print(" ==> Reject it...")*

**return** state, cost\_function(state), states, costs

**Task 3: Other maintenance functions:**

interval = (-10, 10)

**def** f(x):

*""" Function to minimize."""*

**return** x \*\* 2

**def** clip(x):

*""" Force x to be in the interval."""*

a, b = interval

**return** max(min(x, b), a)

**def** random\_start():

*""" Random point in the interval."""*

a, b = interval

**return** a + (b - a) \* rn.random\_sample()

**def** cost\_function(x):

*""" Cost of x = f(x)."""*

**return** f(x)

**def** random\_neighbour(x, fraction=1):

*"""Move a little bit x, from the left or the right."""*

amplitude = (max(interval) - min(interval)) \* fraction / 10

delta = (-amplitude/2.) + amplitude \* rn.random\_sample()

**return** clip(x + delta)

**def** acceptance\_probability(cost, new\_cost, temperature):

**if** new\_cost < cost:

*# print(" - Acceptance probabilty = 1 as new\_cost = {} < cost = {}...".format(new\_cost, cost))*

**return** 1

**else**:

p = np.exp(- (new\_cost - cost) / temperature)

*# print(" - Acceptance probabilty = {:.3g}...".format(p))*

**return** p

**def** temperature(fraction):

*""" Example of temperature dicreasing as the process goes on."""*

**return** max(0.01, min(1, 1 - fraction))

**Task4: Run your code**

Run your code and observe the result. Make sure your code works and convince yourself you understand the code and the output

annealing(random\_start, cost\_function, random\_neighbour, acceptance\_probability, temperature, maxsteps=30, debug=**True**);

**Task5: Visualization**

Comment previous code and add the code below. Here we save the output of annealing function in four variables that we use in see\_annealing function for the visualization.

Note that the debug variable in turn to false here to prevent the annealing function printing the debug information

state, c, states, costs = annealing(random\_start, cost\_function, random\_neighbour, acceptance\_probability, temperature, maxsteps=1000, debug=**False**)

Add functions below:

**def** see\_annealing(states, costs):

plt.figure()

plt.suptitle("Evolution of states and costs of the simulated annealing")

plt.subplot(121)

plt.plot(states, 'r')

plt.title("States")

plt.subplot(122)

plt.plot(costs, 'b')

plt.title("Costs")

plt.show()

**def** visualize\_annealing(cost\_function):

state, c, states, costs = annealing(random\_start, cost\_function, random\_neighbour, acceptance\_probability, temperature, maxsteps=1000, debug=**False**)

see\_annealing(states, costs)

**return** state, c

visualize\_annealing(**lambda** x: x\*\*2)

**Task6:**

Run the program for the functions below and

1. Record the cost and state
2. Generate plots and make snapshot of plots.
3. y = x2 with the interval range of interval = (-10, 10)
4. y = -x3-x2 - with the interval range of interval = (-1, 0)
5. y = x3 - with the interval range of interval = (-10, 10)
6. y = x3+ x2 + x - with the interval range of interval = (-0.7, 0.8)

Paste part a and b for all functions in word doc and submit your doc to the BB. Please include your name and your ID in your document.