Keyboard Optimization

Lodha Soumya Sachin

EE23B140

1 Approach

1. The aim of this assignment is to find the optimal keyboard layout for any given string of input characters such that distance travelled by the finger is minimum.

2. We use the classic approach to the 'Traveling Salesman Problem' to solve our issue using the method of Simulated Annealing.

3. We have pre-defined coordinates for our keys from the QWERTY layout given for Assignment 4.

4. By randomly swapping two of the keys and doing this for a certain 100 iterations will finally give us the minimum distance and the optimal layout.

2 Swapping the keys

```
def generate_new_layout(layout):
    new_layout = copy.deepcopy(layout) # Create a copy of the current layout
    keys = list(new_layout)

    # Select two distinct keys to swap
    key1, key2 = random.sample(keys, 2)
    # Swap their positions

    new_layout[key1]['pos'], new_layout[key2]['pos'] = new_layout[key2]['pos'],
        new_layout[key1]['pos']
    for key in layout:
        if new_layout[key]['start']==key1:
            new_layout[key]['start']='temp'
    for key in layout:
        if new_layout[key]['start']==key2:
            new_layout[key]['start']=key1
    for key in layout:
        if new_layout[key]['start']=='temp':
            new_layout[key]['start']=key2

    return new_layout
```

Here, we need to be careful about swapping the keys in the 'start' section of the dictionary as well for which we need to define a temporary variable 'temp' and run three separate for loops.

3 Simulated Annealing

For this, we just make basic changes to the simulated annealing function given for TSP and return the list of best distances, current distances and best layouts.

Also used the deepcopy function instead of copy everywhere to ensure error-free changes throughout.

Use intial temperature of 10(100 does not give very good results) and a cooling rate of 0.995.

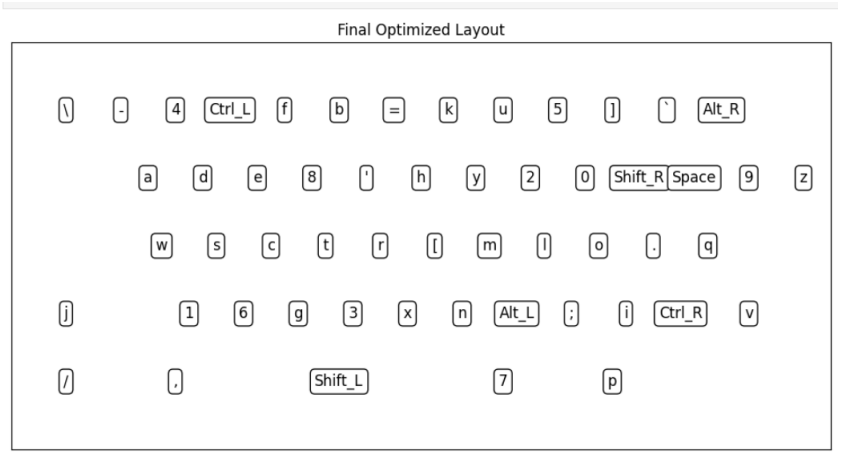
4 Animation and Output

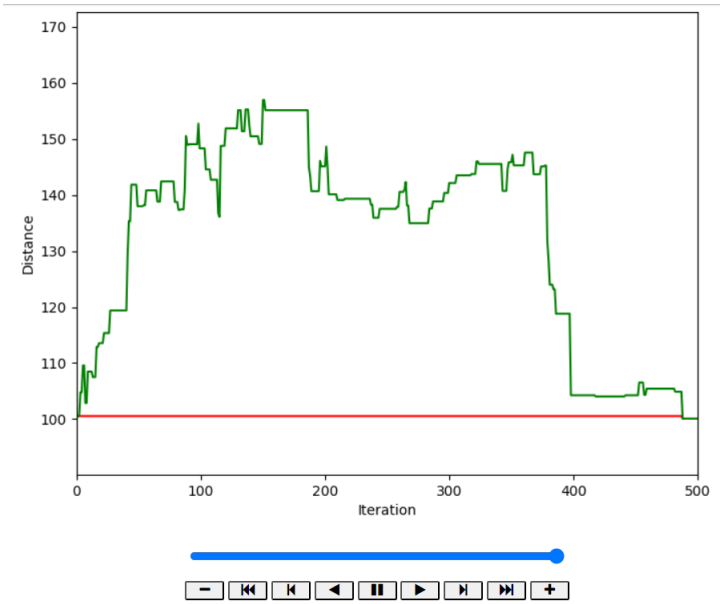
Plot the final keyboard layout using the coordinates of the last element in the best layouts list.

For showing current and best distances over iterations make suitable changes to given animation code. Use colour coding to display both on the same graph.

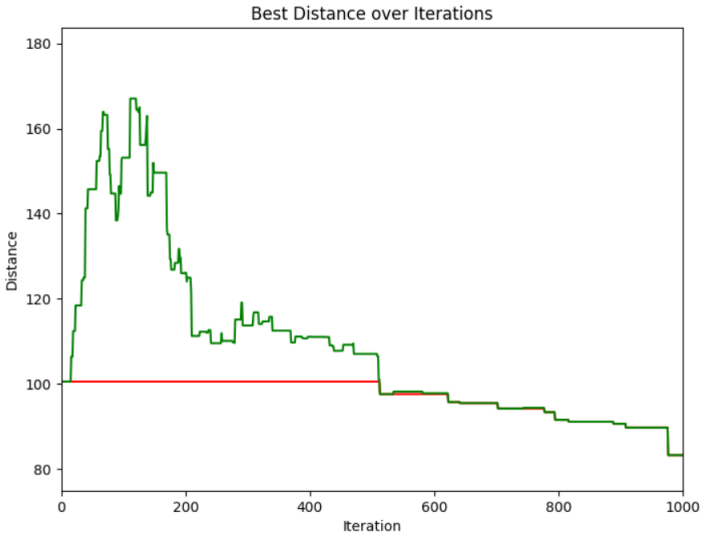
Due to memory constraints, the code on Jupyter server was unable to perform animation for more than 500 iterations. The results come out to be better if ran for 1000+ iterations.

This is what a sample output will look like -





If ran for 1000 iterations we get better output -



NOTE : QWERTY Layout is already a highly optimised layout. This is why for most of the small strings there is barely any change in the final layout. To get proper results, use big paragraphs and possibly higher number of iterations.