Assignment 5 Keyboard Optimization

Deepak Charan S ee23b022

October 2024

1 Introduction

I am Deepak Charan S and this is my report for the fifth assignment of EE2703 course (Applied Programming Lab).

Disclaimer: Since this algorithm is based on random sampling to optimise, the %improvement keeps fluctuating. Please run the same cell more than once if the improvement doesn't seem good

2 Assumptions:

1. The input layout is of the same format as the one provided in programming quiz 4
2. For paragraph-style strings which are quite long, Space key heavily dominates and **skews the distribution** of my heatmap so I have sidelined it (kept it as grey)
3. Shift keys coming to the centre wouldnt violate any touchtyping convention as long as *Shift_L* doesnt go the right side of the keyboard and vice versa

3 Overall Approach:

I used a class 'kybd_optimise' to perform the desired operations (takes in a layout dictionary as input)

3.1 Methods Used:

- **euc_dist**: takes in two keys as inputs and returns the Euclidean Distance between them.
- **key_dist**: Creates a map of each key and their corresponding travel distances (using *euc_dist* and stores it in *'self.init_travel'* or *'self.travel'* dictionary)

- ❊ **visualise_kyb**: plots the keyboard
- ❊ **travel_dist**: takes in a string and layout as input, goes through the string and calculates the frequency of each key pressed. It returns the total travel distance
- ❊ **gen_layout**: Generates a new layout by swapping two random keys of the current layout
- ❊ **sim_anneal**: Takes in the necessary parameters for simulated annealing and performs it. Returns the most optimized layout
- ❊ **update**: The method used in animation
- ❊ **visualise_kyb**: Generates the animation (using the update method above)
- ❊ **kybd_image**: Generates a static image of the initial keyboard and the optimized one

Upon creating a class object (with the user specified layout) and calling the *sim_anneal* method for a string; my class first maps each key to its travel distance. Then it runs through the string using the initial layout and computes the overall travelling distance and keeps a count of how often each key is pressed.

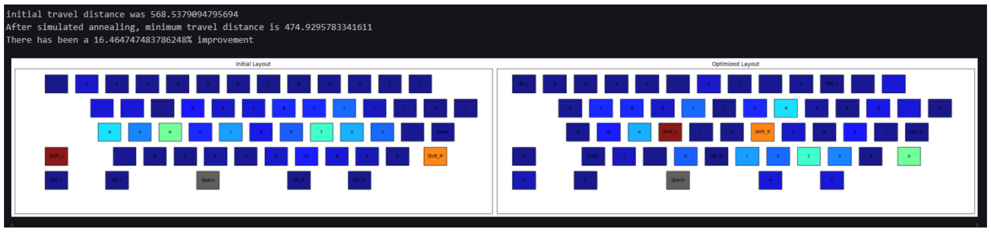
For optimization, the *sim_anneal* method keeps generating neighbouring layouts and compares its performance (It considers the layout if its better or through a probability function)
Thus, It keeps trying to optimise the layout for a certain number of iterations (The probability function gets stricter as we go on) and we can use the *visualise_kyb* method to view the operation happening

3.2 How To Run

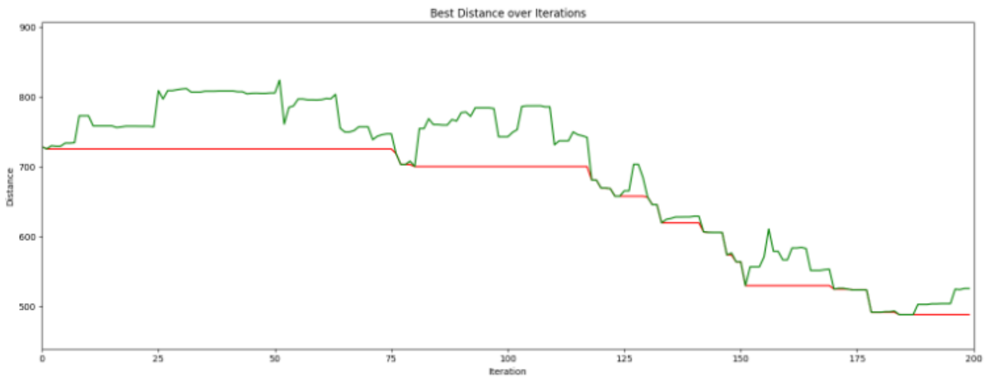
1. Extract the zip file *EE23B022_kybd*
2. Run the prerequisite cells (import libraries, and the class) of *EE23B022_kybd_optimise.ipynb*.
3. The user can provide their own layout (or use 'qwerty' by default) to create a class object.
4. User can then run this object across any string and optimize their keyboard using the *sim_anneal* method.
5. We can run the animation cell to view the process of simulated annealing or just run the *kybd_image* cell to compare the initial and final keyboard layout.
6. I have also implemented **Dvorak** and **Colemak** layout and tried optimising all 3 keyboard layouts against a very long string (I have left the option to run either the animation or view just the final result. Please uncomment whichever one you wish to view).

7. Finally, I have also kept a cell which can run against any custom layout with any given string. Please replace **input_layout** with your layout name and then try it out.

4 Sample Output:



```
initial travel distance was 568.5379094795694
After simulated annealing, minimum travel distance is 474.9295783341611
There has been a 16.464747483786248% improvement
```



(Colormap used is attached in my notebook)

Link to a sample output of my animation is given here

5 Findings:

Upon checking it with a very long string (*attached in my main notebook*, Qwerty keyboard seemed to be the least efficient (high travel distance) with Dvorak coming in second and Colemak being the best.
Qwerty and Dvorak could achieve decent improvements after optimisation while Colemak, being already optimised, never went above 5% improvement

6 References

- To understand how simulated annealing works

- Simulated Annealing Demo provided in Moodle