Keyboard Layout Optimization Using Fitts' Law

Introduction

QWERTY Layout dominates the keyboard world currently. However, there exist many possible layouts that have potentially higher performance than QWERTY. In this programming exercise, we will generate layouts that optimized for movement speed for a 6 by 5 virtual keyboard utilizing the Fitts' Law.

Environment Assumption

Touchscreens are the dominating input devices on modern smartphones. In this exercise, we assume the virtual keyboard will work on a smart phone such as Google Pixel 2. Thus, the given parameters are based on the screen size of Pixel 2. Also, to simplify the model, we assume
1. The user will only use one hand to operate this phone.
2. The key width is fixed.

Given:
Word frequency list:
The word frequency list were extracted from ANC and containing top 100 most frequently used word in modern american English.

Keyboard dimensions:
The keyboard is a 6 rows by 5 column virtual keyboard like this:
The keywidth is XX cm for each key. In this exercise, we just we key width= 1 key.
The key coordinates were given in a 2d list, where the first index is the row from top to bottom, and the second index the column from left to right. See the template code.

Template code in python
The template code containing the following skeleton functions:

1. Read the corpus and compute the table for Pij
2. A function to evaluate the Fitts' law
3. The SA optimization

Please complete these 3 functions based on the objectives described above.
For the required interface, please see the comments in the template code.
You can declare any additional functions that can be used in the skeleton functions, however, please do not change the interface of these skeleton functions!

Objective

Base on Fitts' Law, find the optimal layout.
Formally speaking, we are finding the best arguments that minimize the following function:

$$t = \sum_{i=1}^{N} \sum_{j=1}^{N} P_{ij} MT_{ij} \quad (1)$$

Where t is the average movement time, N = 26 is the number of characters in English, Pij is the probability of the movement from key i to key j, MT is the movement time of from key i to key j.
We use the Fitts' Law to model MT:

$$MT_{ij} = a + b\ log_2(\frac{D_{ij}}{W_{ij}} + 1) \quad (2)$$

Where a, b are two coefficient determined empirically, Dij is the distance from the center of key i to the center of key j, Wij is the keywidth. You will be given a, b , and Wij and the coordinate of each key.

To evaluate Equation (1), you should first generate a di-gram table containing the probability of all movement pairs. It's a routine NLP task. To get Pij, you could first compute the number of all possible movement pair in the given corpus. For example, for the word "nice", you could break the string into 3 digrams: "n,i","i,c","c,e", and then use the word frequency as the digram frequency. Use a python dictionary to accumulate the frequency of each digram, and in the end you divide the accumulated number by total digram frequency to get Pij.
We assume the user will following the straight line from one key center to another key center. Thus, to get Dij we could just find the euclidean distance between 2 keys from their coordinates. After complete these two task, we could start using the SA algorithm to approximate the optimal solution.

The Simulated Annealing Algorithm

One possible way to find the optimal layout is to first generate all possible layouts and then for each layout, we compute the average movement time based on Equation (1). However, this brute-force approach is not computable since the number of possible layout (>10^26) is far beyond our computing power. To tackle this problem, we use a kind of genetic algorithm- the simulated annealing to approximate the optimal solution. The algorithm works like this:
Given a random starting layout, we randomly exchange the position of 2 keys. If the resulting layout has lower average movement time, then we keep this layout and use this layout as the starting layout for next iteration. After k iterations, if the average layout is not lowering anymore by exchange 2 keys, then we stop and output the result.

This algorithm is very simple, however, it may ends up finding local optimum instead of global optimum. There’re many ways to deal with this issue, here, you could just run it many times and use the best one as the result.

Sample Results

Your results might be better or worse than this one.

What you should turn in

A zip file named [studentID_1]_[studentID2]_HW1.zip containing the following files:
1.Your modifed code renamed to [studentID1]_[studentID2]_HW1.py
2.A brief readme reporting your results including the optimal layout and the predicted performance.

Example: 111156001_111156002_HW1.zip
-111156001_111156002_HW1.py
-README.txt

How to turn in

Using Blackboard.
Upload your zip file before deadline, each student should upload 1 copy.

Grading procedure

The TA will download all student's code and then runs by batch. For those got less than 90 pts, we'll check your readme.

Grading Criteria

30 pts: The Fitts' Law function works correctly
30 pts: The di-gram table constructed correctly
30 pts: The SA function finish with results better than QWERTY layout.
    If the function does not finish within 10 mins, -15pt
    If the function finishes with results worse than QWERTY, -15pt
10 pts: (your results-out optimal results)/(QWERTY-our optimal results)*10, if your results end up better than our results, then you can get higher than 100 pts.