Keyboard Layout Optimizer Using Simulated Annealing

Krutarth Patel

October 13, 2024

1 Introduction

This report presents a Python program designed to optimize keyboard layouts using a simulated annealing algorithm. The program aims to minimize the total distance traveled by fingers when typing a given text.

NOTE: this will NOT work on the Jupyter server since a real-time graph is displayed as the layout is optimized. Please run it on a local machine, Windows or Linux (haven't tested in MacOS).

2 Usage

If your machine allows, you will see a real-time graph of the score as it is optimized. Once the optimization is over, quitting the real-time graph will produce the optimized keyboard layout. To use the program, run it from the command line with two arguments:

1. path to the keyboard layout XML file
2. path to the text file to optimize for
3. number of iterations to run

Example:

```
python keyboard_optimizer.py layout.xml sample_text.txt
```

3 Methodology

The optimization process employs simulated annealing, a probabilistic technique for approximating the global optimum of a given function. In this case, the function to be minimized is the total distance traveled by fingers when typing a specific text.

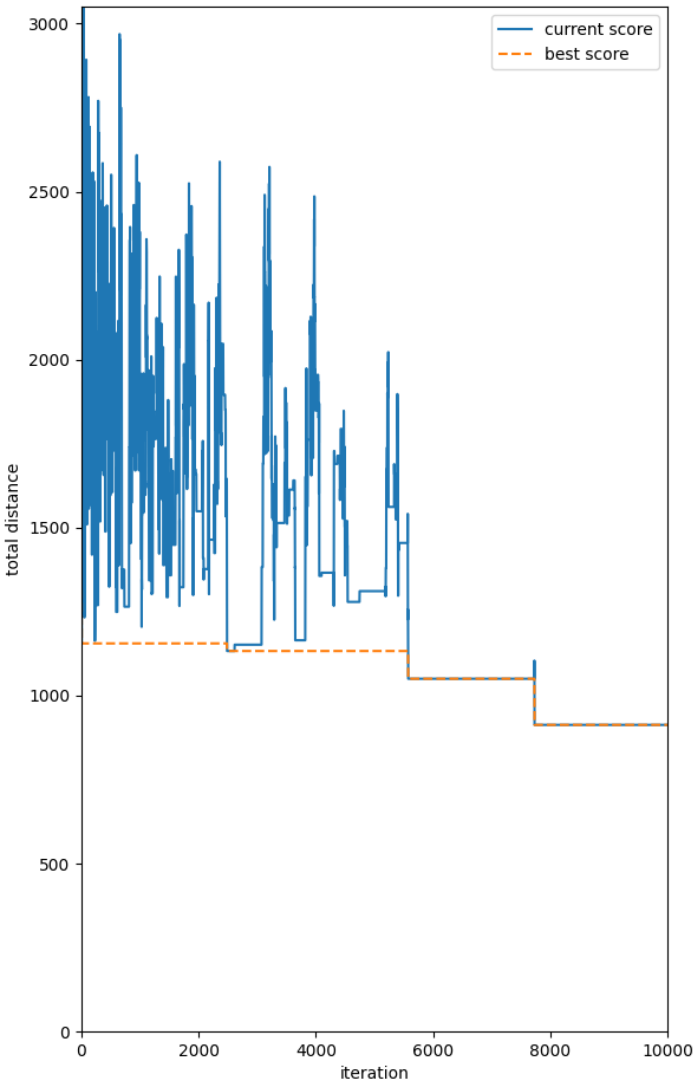

Figure 1: Opimization

3.1 Simulated Annealing

The simulated annealing algorithm works as follows:

1. Start with an initial layout (current solution).
2. Generate a new candidate layout by making random changes to the current layout.
3. Calculate the difference in the evaluation metric (total distance traveled) between the candidate and current layouts.
4. Accept the candidate layout if it's better than the current one, or accept it with a probability that decreases over time if it's worse.
5. Repeat steps 2-4 for a predetermined number of iterations.

4 Key Components

4.1 Optimizer

The `Optimizer` class implements the core optimization algorithm. Its main functions include:

- Calculating distances traveled by fingers
- Generating random layouts by swapping keys
- Implementing the simulated annealing algorithm
- Visualizing the optimization process in real-time

4.2 Painter

The `Painter` class is responsible for creating a visual representation of the optimized keyboard layout using matplotlib. It draws:

- Individual keys with their labels
- Shift-mapped characters
- Special highlighting for home row keys

5 Implementation Details

5.1 Distance Calculation

The finger travel distance is calculated by finding the minimum distance of the key from the home row keys defined in the layout. You can define a row to be the home row by setting the `home` attribute.

The distance traveled by fingers is calculated using the Euclidean distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

where (x_1, y_1) is the position of the home key for a finger and (x_2, y_2) is the position of the key to be pressed.

5.2 Temperature Schedule

The temperature in the simulated annealing process decreases over time according to the formula:

NOTE: I have taken the sqrt in the denominator since I found that the temperature was reducing too fast, leading to less "randomness"

$$T(t) = \frac{T_0}{\sqrt{t+1}} \tag{2}$$

where T_0 is the initial temperature (set to half the number of iterations) and t is the current iteration.

5.3 Acceptance Probability

The probability of accepting a worse solution is calculated using the Metropolis criterion:

$$P(\text{accept}) = \exp(-\frac{\Delta E}{T}) \tag{3}$$

where ΔE is the difference in evaluation metric between the candidate and current layouts, and T is the current temperature.

6 Conclusion

By minimizing the total distance traveled by fingers during typing, it has the potential to improve typing efficiency and reduce strain. Future work could include:

- Incorporating additional ergonomic factors into the optimization metric
- Implementing machine learning techniques to personalize layouts based on individual typing patterns
- Extending the program to support multi-language optimization

7 Documentation for custom xml format

I am using a custom xml format for easy parsing and defining of the layout, this allows me to specify a great deal of detail in a widely supported format which can be read by other programming languages as well. Given the time constraints I have not implemented these ideas, but I list them here: Two example formats have been provided. `kbd.xml` implements the classic QWERTY layout and `split.xml` implements a split keyboard layout. While writing your own keyboard format do note some of the escape sequences you would have to use for certain keys like &,<, > etc. Look at the example formats for further referance.

- specify UI elements like size and color of the keys and heatmap.
- specify the hand position
- use faster languages to optimize the layout and generate the xml to test using python

8 Root Element: `<kbd>`

The root element `<kbd>` represents the entire keyboard layout.

- **Attributes:**
  - `name`: The name of the keyboard (e.g., "qwerty keyboard").

9 Row Element: `<row>`

The `<row>` element represents a single row of keys on the keyboard. NOTE: your layout must have a Row element with the attribute `'home'`. This row will be used for distance calculations.

- **Attributes:**
  - `id`: An identifier for the row (e.g., `'row1'`, `'row2'`).
- **Child Elements:**
  - Contains multiple `<key>` elements, each representing a key within the row.

10 Key Element: `<key>`

The `<key>` element represents an individual key on the keyboard.

- **Attributes:**

- `lower`: The character produced when the key is pressed without any modifier (e.g., lowercase letters, numbers).
- `upper`: The character produced when the key is pressed with the Shift modifier (e.g., uppercase letters, special characters). This attribute may be omitted for keys without an upper case, such as `Backspace` or `Enter`.

- **Child Element:**
  - `<pos>`: Specifies the position of the key on the keyboard.
    - `<x>`: The horizontal position of the key within the row.
    - `<y>`: The vertical position of the key (indicating the row number).

11 Special Keys

Some keys such as `Backspace`, `Enter`, `Space`, and modifier keys like `Shift` and `CapsLock` are represented without an `upper` attribute. These keys have specific functionality and do not produce character output.

12 Example Breakdown

12.1 Row 1 (Numbers and Special Characters)

The first row contains the number keys and their corresponding special characters. Each key has both `lower` and `upper` values.

```
<row id='row1'>
    <key lower="1" upper="!">
        <pos><x>1</x><y>0</y></pos>
    </key>
</row>
```

12.2 Row 2 (QWERTY Letters)

The second row contains the QWERTY letter keys. Each key specifies both its lowercase and uppercase variants.

```
<row id='row2'>
    <key lower="q" upper="Q">
        <pos><x>0.5</x><y>1</y></pos>
    </key>
</row>
```

12.3 Special Keys

Special keys, such as `Backspace`, are defined with a `lower` attribute only.

```
<key lower="Backspace">
    <pos><x>13</x><y>0</y></pos>
</key>
```

12.4 Space Bar

The space bar is defined as a large key that spans the center of the keyboard layout.

```
<row id='space'>
    <key lower="Space">
        <pos><x>3.5</x><y>4</y></pos>
    </key>
</row>
```

This XML structure allows easy customization of keyboard layouts by specifying key values and positions.