Local Search Optimization in the Virtual Keyboard Design Problem

Micah Pietraho [1]

Abstract

Two main keyboard layouts are traditionally used for text entry in English – QWERTY and the lesser-known Dvorak. Both were designed during the early the twentieth century and were constrained by the mechanics of the typewriter. In this paper, I investigate keyboard design in contexts without such restraints, such as electronic text entry devices. I evaluate the relative efficiencies of traditional layouts with regard to the speed of text entry and propose new keyboard arrangements optimized for typing speed.

The speed of text entry relies on three factors: the frequencies of digraphs, or ordered pairs of letters, in a corpus of texts representative of what is being typed, the keyboard layout itself, and the facility with which the user can strike these pairs of keys consecutively. I collect data on the first and third factors, and use methods of computational mathematical optimization to determine the second. More specifically, I use three local search algorithms: simple hill climbing, steepest descent, and simulated annealing.

The layout produced when optimizing for stylus input and the layout produced when optimizing for touch typing input both achieve considerable gains in speed relative to the QWERTY and the Dvorak designs. Additionally, I investigate the robustness of my results with regards to the keystroke biometrics collected experimentally.

1. Introduction

Designed in 1874, the layout of the QWERTY keyboard was constrained by the mechanics of the typewriter. Despite changes in technology, economic lock-in has made QWERTY the dominant keyboard layout for nearly 150 years. Even with modern computer keyboards and virtual keyboards that have no mechanical elements, such as those used in cell phones and tablets, none of QWERTY's competitors have been able to catch on.

This paper quantitatively evaluates QWERTY and its most famous rival, the Dvorak keyboard layout, and uses mathematical methods of optimization to design new layouts. Its goals are as follows:

1. Determine if the Dvorak keyboard layout allows for faster typing than QWERTY does.
2. Design a new keyboard layout optimized for the speed of text entry using mathematical optimization techniques.
3. Compare three different hill-climbing optimization methods in the context of the virtual keyboard design problem.
4. Investigate the robustness of my results.

This paper pursues these goals for both stylus text entry and touch-typing in the English language.

2. Related work

Since QWERTY's introduction, the field of keyboard design has been very active. In this section, I recount the development of the QWERTY and Dvorak layouts and summarize more recent work in the field.

2.1. QWERTY

Having invented the up-stroke typebar mechanism, Christopher Latham Sholes designed the first version of the QWERTY layout in 1873 (Starr, 2016). The typewriter would jam if its typist typed too fast, so Sholes strove to limit typing speed. When he sold the design to E. Remington and Sons, the company shifted the design so that the brand name, 'Type Writer,' could be spelled by typing on only the top row. Here the QWERTY layout reached its final form, soon becoming the industry standard (David, 1985). It is interesting to note that E. Remington and Sons made no claims about typing speed on their layout, marketing it instead on its ubiquity (Howells, 2005).

[1]Brunswick High School. Correspondence to: Micah Pietraho <micah.pietraho@gmail.com>.

Northern New England Junior Science and Humanities Symposium, Durham, New Hampshire, 2020.

2.2. Dvorak

In 1936, August Dvorak patented his `Dvorak Simplified Keyboard` layout. He hand designed it according to nine guiding principles, such as balancing the load between the right and left hands and directing typing from the edges of the keyboard inward. The US Navy ran a study in 1944, concluding that typists could both type 51% faster on the `Dvorak` layout than on the `QWERTY` layout and learn to type on `Dvorak` three times faster (Dvorak, 1944). August Dvorak himself ran the study and questions have been raised as to its validity due to his financial stake in the results and the lack of transparency surrounding it. In 1956, a study by the General Services Administration study found results opposite those of the Navy study. It concluded that `Dvorak` typists "showed less improvement in speed" during the training, that during short periods of typing the typists performed at an equal speed on both layouts, and that when typing for longer periods of time, the `QWERTY` typists were "better in both speed and accuracy and that the differences are significant" (Strong, 1956). Evidence for `Dvorak`'s superiority is mixed. In 1985, Liebowitz and Margolis found the case for `Dvorak`'s dominance to be "scant and suspect" (David, 1985). Both scholarly and popular debate still remain; see `QWERTY` vs. `Dvorak` Efficiency by Ricard Torres in 2013 (Torres, 2013) and the podcast Planet Money's "Why Do We Still Use `QWERTY` Keyboards?" from April, 2019 (C. Garcia, 2019).

2.3. Subsequent Work

In this subsection, I outline recent work in the field of keyboard layout design and compare their methods to mine.

Zhai et al. in 2000, proposed two keyboard layouts optimized for stylus text entry in English, called Hooke's and Metropolis. Their layouts were optimized using simulated springs and a Metropolis Random Walk algorithm, respectively. The physical distance between keys was used to predict how long a typist would take to move a stylus from one key to another (S. Zhai, 2000).

In 2013, Jan Eggers et al. introduced their own keyboard layouts optimized for touch-typing in English as well as German by using ant colony optimization. They measured the merit of each keyboard layout with a system of heuristics that represented load, hand alteration, consecutive usage of the same finger, big steps, and hit direction. These heuristics were weighted with chosen parameters and summed to give a keyboard layout what they called a global grade (et al., 2013). Işeri and Ekşioğlu, however, note in a 2015 paper that Eggers et al. "did not use valid digraph cost parameters, instead relied on subjective opinions of a few experts in determining the values of the parameters" (A. Iseri, 2015).

In the same paper, Işeri and Ekşioğlu collected experimental data on the delay times between pressing different keys instead of using physical distance or a heuristic approach. Seven typists toggled for two minutes between each pair of keys where both keys were assigned to the left hand or both to the right hand by touch-typing conventions. Using their results, they designed a layout, the I-layout, for for the Turkish language (A. Iseri, 2015).

In 2019, Palin et al. collected data from 37,370 volunteers to compare text entry rates on computer keyboards and on mobile devices with autocorrect, word prediction, and gesture keyboards. They found that, on average, text entry on a mobile device, such as with two thumbs, is 29% slower than on a computer keyboard. (et al., 2019).

This paper differs from the above in the following ways:

1. Unlike Zhai et al., I model touch-typing text entry as well as stylus text entry.
2. Unlike Eggers et al., I use experimental data to model touch-typing text entry instead of arbitrary heuristics.
3. Unlike Işeri and Ekşioğlu, I am considering text entry in English, not Turkish.
4. Unlike Palin et al., I am not considering thumb text entry or text entry augmented by autocorrect, word prediction, or a gesture keyboard.

I use a model of stylus text entry similar to the one used by Zhai et al. and a model of touch-typing using experimental data collected by methods similar to Işeri's and Ekşioğlu's.

3. Keyboard Evaluation

I would like to measure how fast text can be entered by typing on a given keyboard. I will make the following simplifying assumptions about text entry. Each letter will be assigned to one of the first ten keys on any of the three central rows of a standard keyboard. Punctuation, accents, other non-letter symbols, and capitalization will be ignored.

The speed with which a typist can enter a text on a keyboard is determined by three factors: the frequencies of digraphs in a corpus representative of the typed text, how letters are assigned to keys, and the rate at which consecutive keys are struck.

3.1. Digraph Frequencies

A digraph is an ordered pair of characters, like 'AY,' 'TH,' or 'XQ.' Due to my simplifying assumptions, all the digraphs considered in this project will be pairs of capitalized letters. I will use D_i to represent the frequency of the i^{th} digraph of the 676 in English and $\mathcal{D}$ as the set of all D_i. I collect data

that estimate digraph frequencies in English and describe the collection process in Section 4.1.

3.2. Keyboard Layout

A keyboard layout is an assignment of each letter in the English alphabet to one of the thirty keys shown in Figure 1 and Figure 2. Note that there are four keys traditionally assigned to punctuation. Because I am ignoring punctuation, those keys are blank. I will denote a keyboard layout with the capital letter L. `QWERTY` and Dvorak are two examples of keyboard layouts. They are shown in Figures 1 and 2. We will use $L(i)$ to denote the ordered pair of keys used to type the i^{th} digraph in English on the layout L. It is important to understand that a layout within one swap of L is called a neighbor of L.

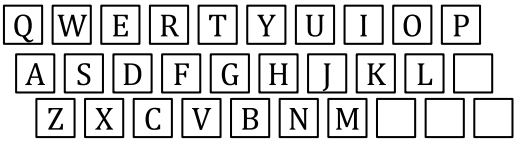

Figure 1. The `QWERTY` keyboard layout

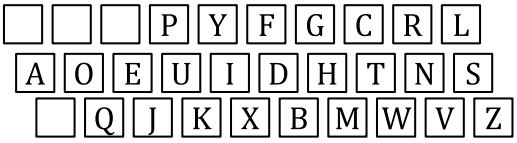

Figure 2. The `Dvorak Simplified Keyboard` layout

3.3. Key Typing Speed

Define a key-pair to be an ordered pair of unlabeled keys on a keyboard. The time it takes to type a key-pair is written as C_k for an ordered pair of keys k. This 'key-pair cost' C_k depends on the method of text entry but not on the letters assigned to those keys by a layout. $\mathcal{C}$ is the set of all key-pair costs. I collect data that estimate $\mathcal{C}$ with methods I describe in Section 4.2. Note that $C_{L(i)}$ is the time it takes to type the i^{th} digraph in English on the keyboard layout L.

3.4. Scoring

I will use the following function to score a given keyboard layout L.

$$S(L, D, C) = \frac{1}{676} \sum_{i=1}^{676} D_i \cdot C_{L(i)}.$$

In this paper, I estimate $\mathcal{D}$ and $\mathcal{C}$ and find an L such that $S(L, D, C)$ is minimized. A score $S(L, D, C)$ can be interpreted as the average time it takes to type a digraph in a text whose digraph counts are similar to $\mathcal{D}$ on a keyboard whose layout is L at a rate captured in $\mathcal{C}$. A low score corresponds to fast text entry.

4. Data

In order to evaluate S, I need to estimate $\mathcal{D}$ and $\mathcal{C}$, respectively the list of digraph frequencies in English and the times it takes to type each ordered pair of keys.

4.1. Digraph Frequencies

In this section, I describe the process by which I estimate $\mathcal{D}$, the list of the frequencies of all the digraphs in English. My algorithm begins by selecting a random Wikipedia article. From this article, a sample of one hundred consecutive digraphs are chosen. The number of times each digraph appears in the sample is recorded and added to a running total. This process is repeated with another random article and random string of of hundred digraphs in that article. I keep track of both the running total and frequency of each digraph. I can plot the list of the frequencies of all the digraphs as a point in 676 dimensions. I record the points before and after each new sample is added and compute the distance between them. The program stops after the distance between successive steps is below 0.001. The program used 491 articles. The data is summarized in Figure 3.

4.2. Key-Pair Costs

In this section, I describe the process by which I estimate $\mathcal{C}$, the time it takes to type each ordered pair of keys on the keyboard. I do so in two ways, one modeling stylus text entry and the other touch-typing.

4.2.1. Stylus

When typing a pair of keys with a stylus or a single finger, the stylus must press the first key, be moved to the second key, and press the second key. The vertical motion is the same for each pair of keys and the rate of movement is the same between every pair of keys, so I set C_k equal to the distance between the centers of the pair of keys k. Notice that using this set of key-pair costs, the cost of going from one key to another, say 'A' to 'R,' is the same the cost going in the opposite direction, from 'R' to 'A.'

4.2.2. Touch-Typing

I follow a method similar to that of Işeri and Ekşioğlu, experimentally collecting a time between keystrokes to estimate each C_k in $\mathcal{C}$. I take special care to avoid bias towards the `QWERTY` layout. Rather than only collect data on key-pairs typed with solely the right or solely the left hand, like in Işeri and Ekşioğlu's study, I run my experiment over all key-

pairs. I toggled between each pair of keys for thirty seconds, using standard touch-typing conventions in my assignment of fingers to keys. I collected each transition time between every pair of keys. In this set of key-pair costs, direction matters. Going from 'A' to 'R' has a different cost than going from 'R' to 'A.'

4.3. Implementation

I wrote two pieces of code in Python to collect this data. One accessed Wikipedia and ran the algorithm that estimated $\mathcal{D}$ for English. It used the Python package Beautiful Soup to access the internet. The other measured the time between keystrokes while I ran the experiment that collected the key-pair costs for touch-typing. Here, I used the Python package Time.

5. Optimization

The goal of this part of the project is to find the keyboard layout L which minimizes S, the scoring function. One way to do this is to test all

$$26! \cdot \binom{30}{26} \approx 1.105 \times 10^{31}$$

possible layouts. If it takes 1 millisecond to score each one, this process would take 350463041989301986402 years. Instead, I use three methods of local search optimization: simple hill climbing, steepest descent, and simulated annealing. All of these begin with a random keyboard layout, making small changes hoping to improve the layout's score.

5.1. Steepest Descent

This algorithm starts with a random keyboard layout. It evaluates all possible swaps of two keys, and chooses the swap that will decrease the score S the most. The process continues until there are no swaps which improve the score. I describe this more precisely in the inset Algorithm 1: Steepest Descent.

This is a deterministic algorithm that produces a sequence of keyboard layouts with successively better scores. Each step is expensive, requiring a check of all possible swaps. It is greedy, so it will likely terminate in a local minimum that is not a global minimum for the function S. This shortcoming is addressed by running the program repeatedly, starting from different random keyboard layouts.

5.2. Simple Hill-Climbing

The optimization method of simple hill climbing works as follows. The algorithm begins with a random keyboard layout and scores it. Two keys are chosen randomly and swapped. The resulting layout is scored. If the score on the new layout is lower, the process begins again, starting with this new layout. If not, another random pair of keys is swapped. This continues until a stopping condition is met. I describe this more precisely in the inset Algorithm 2: Simple Hill Climbing.

Algorithm 1 Steepest Descent

```
currentLayout = randomLayout()
while True do
    L = findAllNeighbors(currentLayout)
    newLayout = NULL
    newScore = infinity
    for all x in L do
        if score(x) ≤ newScore then
            newLayout = x
            newScore = score(x)
        end
    end
    if newScore > score(currentLayout) then
        return currentLayout
    end
    currentLayout = newLayout
end
```

Algorithm 2 Simple Hill Climbing

```
currentLayout = randomLayout()
while True do
    L = FindAllNeighbors(currentLayout)
    newLayout = pickRandomElement(L)
    if score(newLayout) ≤ score(currentLayout) then
        currentLayout = newLayout
    end
    if endingCondition = True then
        return currentLayout
    end
end
```

This is a fast algorithm which creates a list of successively lower-scoring keyboard layouts. Initially, each step requires the evaluation of a single, or just a few, swaps. Because it is not greedy, it explores a wider-range of keyboard layouts than steepest descent before terminating. However, may still terminate a local minimum that is not a global minimum.

Again, this shortcoming is usually addressed by starting at many different initial keyboard layouts. Furthermore, as in steepest descent, simple hill climbing will never examine a keyboard layout that is higher-scoring than the initial random layout, limiting the range of layouts that it can consider.

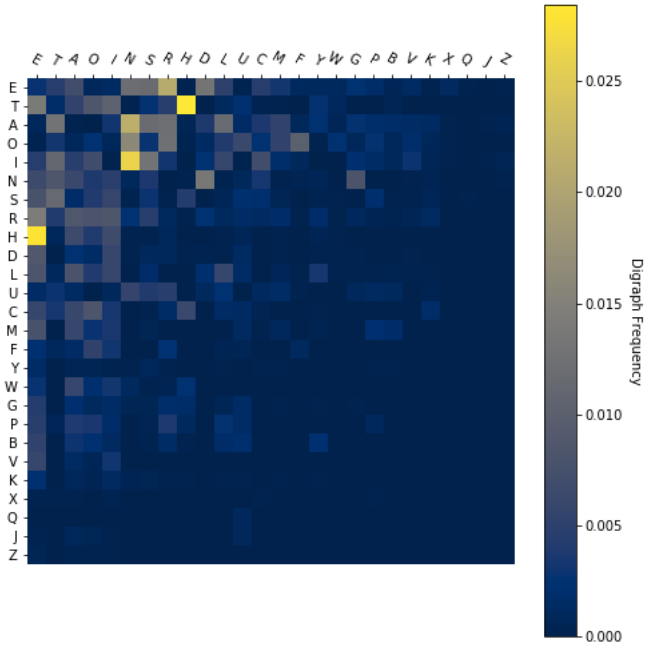

Figure 3. Frequencies of digraphs in English. The square in the i^{th} row and the j^{th} column represents the frequency of the digraph starting with the i^{th} most common letter in English and ending with the j^{th} most common letter in English. The letters along each access are ordered by frequency in English. Notice that 'TH' and 'HE' are the most frequent digraphs in English.

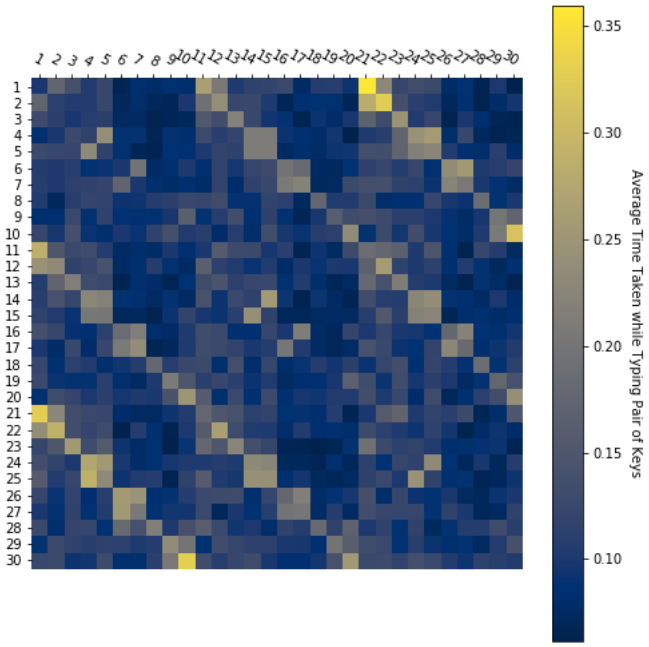

Figure 4. The average time taken to type each pair of keys. Each key has been assigned a number from one to thirty from left to right and then from top to bottom. The square in the i^{th} row and the j^{th} column represents the average transition time from key i to key j using the finger assignments from touch-typing conventions. Some keys take much longer to transition between. For example, the keys 1 and 21, assigned to 'Q' and 'Z' respectively, take a long time to type in either order.

5.3. Simulated Annealing

Simulated annealing attempts to improve on both simple hill climbing and steepest descent. Initially it is very random and grows less so over time becoming more similar to a simple hill climb. "The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects."[1]

The algorithm starts with a random keyboard layout. At every step, the program selects two keys to swap. If the new layout is better, it is adopted and the procedure continues. If not, the algorithm may still choose to adopt the new layout based on a "temperature function". The temperature function represents a probability of moving to an inferior layout and depends on two things: the number of steps taken in the algorithm and the difference between the scores of the current and proposed layouts. This probability decreases with the number of steps taken and the difference between the scores of the two layouts being considered. In my work, I used:

$$\text{temp}(k, c, p)) = e^{-8(S(p)-S(c))0.9999^{-k}}$$

where k is the number of steps taken, $\mathcal{C}$ is the current layout, and p is the proposed layout. The form of this equation is standard, and the constants were chosen after extensive testing.

The process repeats until an ending condition is met. I describe this more precisely in the inset Algorithm 3: Simulated Annealing.

Algorithm 3 Simulated Annealing

```
currentLayout = randomLayout()
currentScore = score(currentLayout)
steps = 1
while True do
    newLayout = pickRandomNeighbor(currentLayout)
    newScore = score(newLayout)
    randomNum = pickRandomNumber(0, 1)
    if newScore < currentScore or
       randNum < temperature(steps) then
        currentLayout = newLayout
        currentScore = newScore
    end
    if ending condition = True then
        return currentLayout
    end
    steps = steps + 1
end
```

[1] "Simulated annealing," Wikipedia, accessed February 2, 2020, https://en.wikipedia.org/wiki/Simulated_annealing.

Simulated annealing offers some advantages over the previous algorithms. Because it may accept layouts with worse scores, it is able to explore a wider range of keyboard layouts. However, this exploration can take significantly more time and it is not clear that is it better than restarting a simpler algorithm at a variety of random layouts. Furthermore, simulated annealing can be too random to be effective if the parameters temperature function are not chosen very carefully.

5.4. Implementation

I wrote several programs to run these optimization methods. I wrote one piece of code for each method and then another to run each one over each set of key-pairs ten thousand times. This last was in java, while the other three were in Python.

6. Results

I now answer the main questions posed in the beginning of this paper.

6.1. QWERTY vs. Dvorak

Using the scoring function defined in Section 3.4 together with the data about the English language and key-pair costs, I am now able to evaluate both the `QWERTY` and `Dvorak` keyboard layouts. Supporting Dvorak's original claim, his keyboard layout allows for faster touch-typing. In fact, 13.8% faster. However, when considering stylus text entry, standard `QWERTY` is 24.5% faster. The precise results are given in Table 1.

	`QWERTY`	`Dvorak`
Stylus	2.489	3.297
Touch-Typing	0.118	0.102

Table 1. A direct comparison of the score S for the two standard keyboard layouts using both the stylus and touch-typing text entry methods.

It is interesting to compare these scores to scores of randomly-chosen layouts. Figure 5 shows the `QWERTY` and `Dvorak` scores in comparison to the distribution of scores for ten thousand randomly-chosen layouts. For stylus entry, `QWERTY` is somewhat better than average while `Dvorak` is much worse. For touch-typing, `QWERTY` is similar to average and `Dvorak` is considerably better.

6.2. Optimized Keyboard Layouts

One of the main conclusions of this paper is that there are keyboard layouts considerably better than either `QWERTY` or `Dvorak`. I propose the `KWHT` and `LMOE` layouts shown in Figure 6.

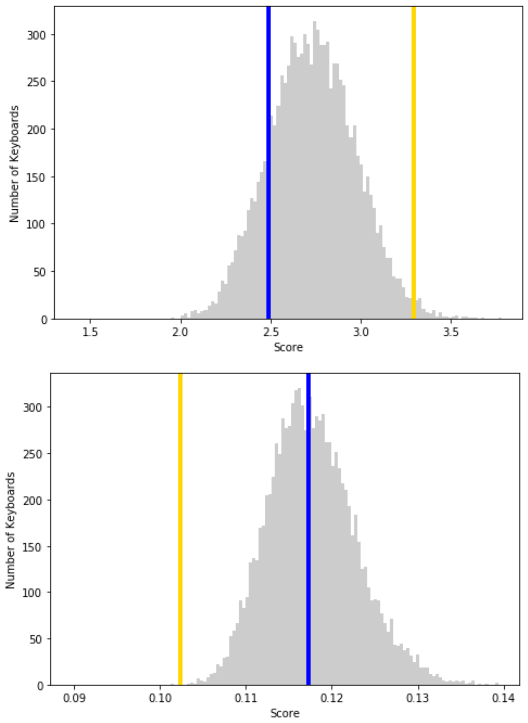

Figure 5. QWERTY (blue) and Dvorak (yellow) scores in relation to the distribution of scores for random keyboard layouts. Stylus entry is displayed on top, and touch-typing on the bottom.

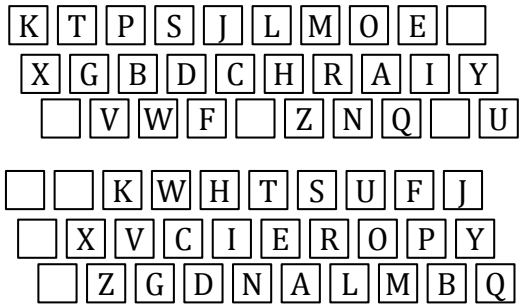

Figure 6. The LMOE (top) and KWHT (bottom) keyboard layouts

The LMOE layout considerably outscores both QWERTY and Dvorak for touch-typing, QWERTY by 24% and Dvorak by 12%. Using stylus text entry, the KWHT layout scores 43% better than QWERTY and 57% better than Dvorak. The precise scores are given in Table 2. In Figure 7, I compare the scores of LMOE and KWHT to the score distributions of random layouts. Both are substantially better than even the best random keyboards. In the next section, I describe the process by which these new layouts were obtained.

It is surprising that the blank keys in the LMOE layout are not clustered together at the edge. The one at the center bottom suggests that striking that area of the keyboard is a relatively awkward action. It would be interesting to study the physiology of these motions.

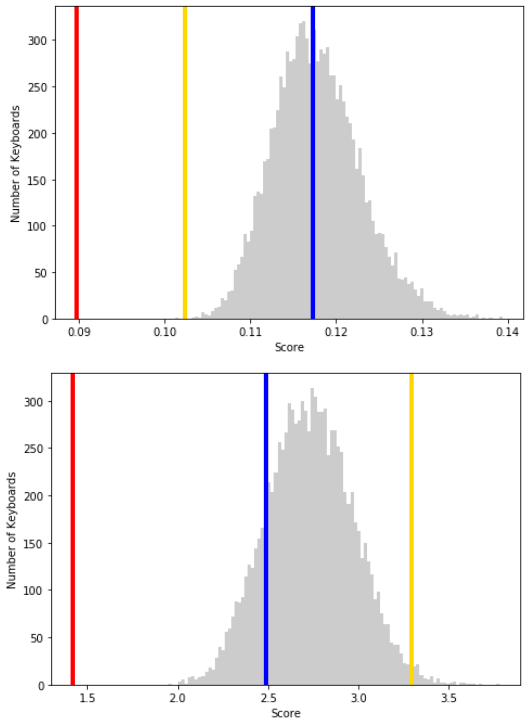

Figure 7. QWERTY (blue) and Dvorak (yellow) scores based on the touch-typing (top) and stylus (bottom) costs as compared to distributions of scores for random layouts. The red line indicates the score of my proposed layouts, LMOE (top) and KWHT (bottom).

6.3. Optimization Techniques

The LMOE and KWHT layouts are both results of the optimizations methods described in Section 5, the former using touch-typing costs and latter using stylus costs. For both, I used my estimate of English digraph frequencies from Section 4.1.

Starting with a different random layout each time, all three optimization algorithms were run ten thousand times. I recorded the best layout and score from each run. In Figure 8, I show the distribution of scores for each optimization method using touch-typing costs. The distributions for simple hill-climbing and steepest descent are nearly identical. The distribution for simulated annealing is better on average. Stylus costs produced qualitatively similar distributions.

Perhaps surprisingly, all three optimization methods resulted in the same overall best keyboard. This was true for both touch-typing and stylus costs.

While each method produced the same best overall keyboard after ten thousand iterations, simulated annealing is clearly

	QWERTY	Dvorak	LMOE	KWHT
Touch-Typing	0.118	0.102	0.090	
Stylus	2.489	3.300		1.418

Table 2. A comparison of scores S for the QWERTY, Dvorak, LMOE, and KWHT layouts.

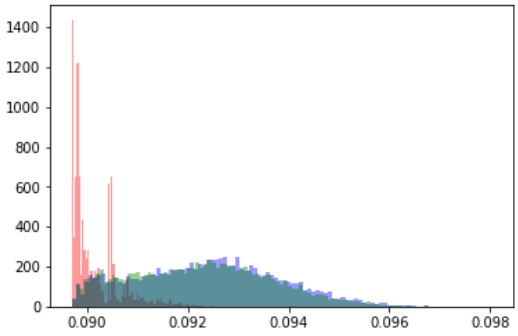

Figure 8. The distributions of the scores of the 10,000 layouts produced by simple hill-climbing (blue), steepest descent (green), and simulated annealing (red).

the best method if only one run is possible. Despite considerable differences in the algorithms, simple hill climbing and steepest descent are indistinguishable after a single run.

6.4. Robustness

I test the robustness of my results in two ways. First, I check that minor variations in my costing data have only a small impact on the optimal keyboard produced. Second, I check that all the low-scoring touch-typing keyboards are fundamentally similar.

6.4.1. Cost Robustness

To account for potential error in my touch-typing data collection, I added 5% white noise to each key-pair cost and ran the full optimization procedure. I then repeated this one hundred times, resulting in one hundred best overall keyboards. While technically different, all the keyboards share similar structural qualities. The vowels always occupy five of the same seven keys. Certain groups of consonants, like 'TH' are always in the same locations. This suggests that there is some fundamental structure of an optimal keyboard layout using touch-typing.

6.4.2. Optimization Robustness

Again, the vowels are always located in the same few squares on the right hand side of the keyboard, and common consonant digraphs, like 'TH,' are found in the same areas on every layout.

6.4.3. Robustness Implementation

I wrote a program in Python that perturbed the touch-typing costs for this section.

Statement of Outside Assistance I was inspired to do this project by Planet Money's April, 2019 episode, "Why Do We Still Use QWERTY Keyboards?" I wrote all the code for the project - the optimization algorithms and the programs necessary for data collection. I designed the data collection processes myself. The protocol for estimating digraph frequencies I borrowed from my science fair project from last year. I used Python 3 and Java to write the code and draw the diagrams. To run the optimization, I used the the Bowdoin high-performance computing cluster. Mrs. Stephanie Dumont was the advisor of this project. I periodically updated her on its status. I had useful discussions with my father about optimization algorithms, and my parents helped to edit this paper. Research involving human subjects was conducted under the supervision of an experienced researcher and followed state and federal regulatory guidance applicable to the humane and ethical conduct of such research.

Source Code The source code in the form of Python Jupyter notebooks and data for this project are available on my code repository located at https://github.com/WillySquid/Local-Search-Keyboards.

References

M. Eksioglu A. Iseri. Estimation of digraph costs for keyboard layout optimization. *International Journal of Industrial Ergonomics*, 2015.

S. Smith C. Garcia. Why do we still use qwerty keyboards? *Planet Money*, 2019.

P. David. Clio and the economics of qwerty. *Annual Meeting of the American Economic Association*, 1985.

A. Dvorak. A practical experiment in simplified keyboard retraining. 1944.

J. Eggers et al. An ant colony optimization algorithm for the optimization of the keyboard arrangement problem. *European Journal of Operational Research*, 2013.

K. Palin et al. How do People Type on Mobile Devices? Observations from a Study with 37,000 Volunteers. In *Proceedings of 21st International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI'19)*. ACM, 2019.

J. Howells. *The Management of Technology and Innovation*. SAGE, 2005.

B. Smith S. Zhai, M. Hunter. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. 2000.

M. Starr. A brief history of the qwerty keyboard. *C—NET*, 2016.

E. Strong. A comparative experiment in simplified keyboard retraining and standard keyboard supplementary training. 1956.

R. Torres. Qwerty vs. dvorak efficiency: A computational approach. 2013.