Keyboard Optimization with Genetic Algorithm

Team 5 (고봉석, 김범준, 이성회, 조민기)

School of Computing, KAIST

I. Introduction

Nowadays, most people use "QWERTY" keyboard. However, current keyboard is not designed with considering typing sequence. For fast typing, it needs two condition. First condition is that to minimize the movement of the finger. When using the keyboard, humans' finger always locate middle line of keyboard. It means that, if users typing a sentence without moving finger on middle line, user can type fast. Second condition is that to use left hand and right hand alternately.

Figure 1 represents that comparing Dvorak keyboard with QWERTY keyboard. Dvorak keyboard is well known as faster keyboard than QWERTY keyboard. Dvorak keyboard satisfies two conditions which we mentioned above. First, frequently used alphabets locate middle line of keyboard. Alphabets which locate middle line of Dvorak keyboard use 70% English documents but alphabets which locate middle line of QWERTY keyboard use only 32% English documents. Also, all vowels locate left side of Dvorak keyboard therefore Dvorak keyboard has high potential to use left hand and right hand alternately because almost English words combine consonant, vowel, consonant, vowel… order.

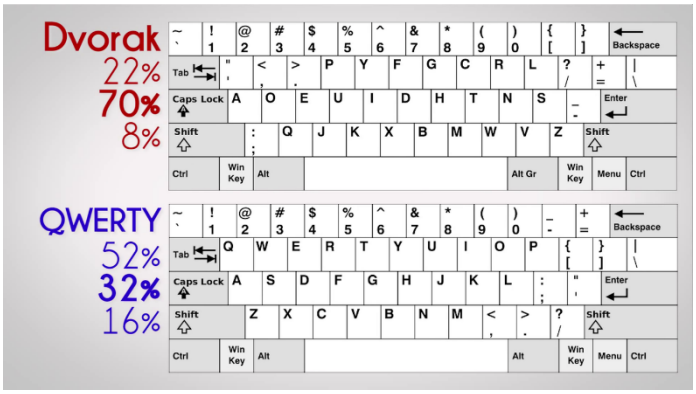


Figure 1. Compare Dvorak with QWERTY

The goal of this project is that to create optimizing keyboard layout using genetic algorithm. We use the Corpus of Contemporary American English (COCA) to optimizing target. We find the best keyboard layout when u9ser typing the COCA.

The contributions of our project are as follow:

1) Our method can apply different situation. We find the best layout for COCA but some people want find the best keyboard layout for writing paper. So, people can find the best keyboard layout that they want.
2) We use Voronoi diagram to make keyboard layout. We do not assume the best shape of keyboard buttons are square. We use Voronoi diagram to fine the best shape of keyboard buttons.
3) We implement crossover technique to apply keyboard layout. There are too many conditions to use crossover generally. So, we implement crossover technique to use keyboard layout.
4) We makes fit function to consider various situation. We consider kind on finger, area of keyboard and distance of finger movement with fitts law. We explain our fit function section 3.

The remainder of this paper is organized as follows. Section II explains background of our project such as Voronoi diagram and fitts law. Section III shows our fit function. Section IV shows our genetic algorithm, crossover and mutation method. Section V shows results our project and Section VI is discussion and conclusion.

II. Background

In this section, we explain Voronoi diagram and fitts law.

II.I Voronoi Diagram

Voronoi diagram is a partitioning of plane, that partitions plane into multiple regions, which are called voronoi cells. Based on seed points existing on the plane, each voronoi cell corresponding to each seed point consist of points which their closest seed point is its corresponding seed point. To calculate distance between points, both Euclidean and Manhattan distance can be used. In the case of using Euclidean distance, each voronoi cells are made up by intersection of half-spaces. Therefore, every voronoi cells are always convex polygon. Voronoi diagram can also be applied to n-dimensional space, and theoretically generalized by Russion mathematician Georgy Voronoy.

II.II Fitt's law

As long as our project is about optimizing keyboard layout, we need to evaluate our keyboard by using HCI(Human Computer Interaction) method. Fitt's Law is a predictive model of human movement primarily used in HCI field. This is a metric to

quantify the difficulty of target task, it predict the time that required to move and hit a target by function of distance and width of target object. Following formula is based on the Fitt's Law.

$$\mathrm{MT} = \mathrm{a} + \mathrm{b} \cdot \log_2\left(\frac{2D}{W}\right)$$

- MT is the average time to complete the movement
- a and b are constants that depend on the environment, so usually these are determined empirically by regression
- D is distance between target and starting point
- W is width of the target object

This law imply that when the target gets bigger and closer, required time to hit the target object decrease, which means it gets faster. For example in case of QWERTY and Dvorak, QWERTY keyboard is slower than Dvorak because the default location of our finger is middle line (asdf…), but many of most frequent characters such like e,r,t,o is located far from the starting point. But in case of Dvorak, most of frequent words are located on the middle line, so it is faster than QWERTY keyboard.

In this way, we can measure the usability of the physical UI that is for touching or pointing a target object by using limbs such as finger or pointing device.

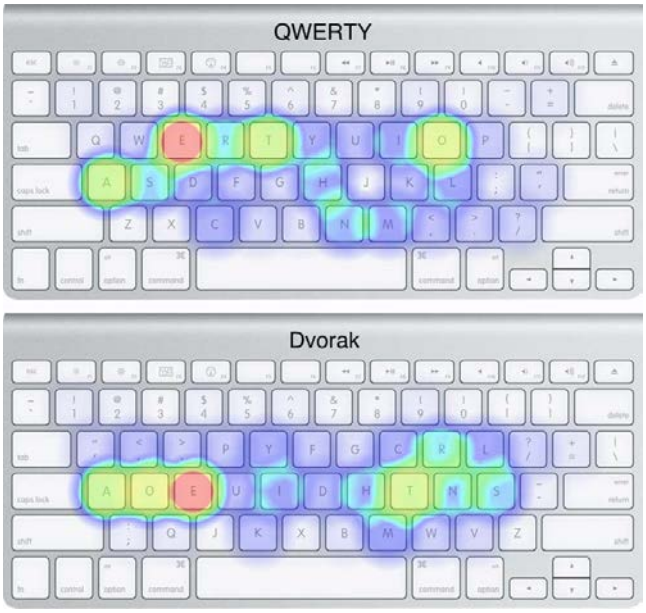


Figure 2 QWERTY and Dvorak

III. Fitness function

Following is our Fitness function

$$\mathrm{F} = F_A + F_S + F_F$$

1. $\mathrm{F_A}$

According to Fitt's function, small size of key affect to usability of the key board. When the key gets too small, the operation speed get slower. In other word, bigger key get better usability. And in the case of keyboard, every finger has their own activity region so we can apply the Fitt's law onto each finger. So when the size of key gets smaller, it gets a penalty and at the and, loss function is calculated by weighted sum with considering frequency of character.

2. $\mathrm{F_S}$

One of the reason why Dvorak is faster than Qwerty is that Dvorak has better hand distribution with considering the sequence of word.

When we compare the standard Korean keyboard and English keyboard, Korean keyboard is faster than English keyboard because of the property of the language. In the case of English, there should be some word that has bad hand distribution such like 'stewardesses'. But in case of Korean language, every character has a concrete structure that constants comes first and the following character should be a vowel. Which means there is a specific sequence of character, Constant-Vowel-Constant, so many sequences like 'ㅍㅍ' or 'ㅏ ㅏ' has 0 frequency in Korean language. Because of this property, we can evenly distribute the pressure to the hand, so it gets better and faster keyboard than English.

We got the sequence data by analyzing the general American English corpus 'COCA'. Following loss function consider the frequency of every sequences and give a penalty to the sequence that has a bad hand distribution. When the sequence use different side of hand (in case of qwerty, such like 'ao', 'me') it is a best case so it gets 0 penalty. And when it use same side of hand but different finger, it gets 0.5 penalty. When it use same finger, in this case the operation is a bit different with other cases.

In other cases, we can press second key while holding the first key. (User usually press 2 or 3 key in the same time when they type) But when we use same finger, in order to press second key we should take off the finger from the first key and move the finger and hit the second key. So in this case, the sequence of the same key like 'aaaaa' is the fastest, and the sequence of the character that are far from each other such like 'qzqzqz' is worst. So it gets penalty from 0.5 to 1 with considering Fitt's law.

$$\mathrm{f_S}(\mathrm{a},\mathrm{b}) = \begin{cases} 0, & if\ \ hand(a) \neq hand(b) \\ 0.5, & if\ \ hand(a) = hand(b) \\ & and\ \ finger(a) \neq finger(b) \\ 0.5\sim1, & if\ \ finger(a) = finger(b) \end{cases}$$

$$F_S = \sum_{a \in S} \sum_{b \in S} f_S(a,b)p(ab) \qquad S \text{ is set of alphabet}$$

p(ab) is frequency of the sequence 'ab'

3. F_F

This term consider about the Fitt's law that applied on the each finger. When the key is located far from the default area it gets penalty. So the keyboard like Qwerty, which the most frequent characters located far from middle gets big loss value, and keyboard like Dvorak, which most frequent characters located close to middle gets less loss value.

$$F_F = \sum_{a \in S} dist(a)p(a)$$

dist(a) is distance from middle line to key a, p(a) is frequency of alphabet a and S is set of alphabet.

A. Validation test

The maximum performance of the keyboard can be measured only when the tester is fully trained, but it will take a year to finish that. As far as the real test is hard for this project, we should make our fitness function suitable for real as possible as we can. So we performed some validation test for well-known solution, Qwerty and Dvorak, 2벌식 and 3벌식. And the result shows right.

F(Qwerty) = 4.111 > F(Dvorak) = 3.002

F(2벌식) = 2.124 > F(3벌식) = 0.954

Our fitness function making sense for existing solutions.

IV. Algorithm

Our main logic is based on genetic algorithm. The pseudocode for main loop is described at Figure 3. NUM_GENERATION is set to 400. NUM_POPULATION is set to 100. We used elitism, as line 4 shows. The number of elites is 20.

At line 6, GA_select_two_parents makes just randomly choosing twice in population. At line 7-9, GA_crossover and GA_mutation are explained at later section.

Algorithm 1 Main Algorithm

```
1:  procedure RUN_GA                                   ▷ Main loop for GA
2:      population ← GA_INITIALIZATION()
3:      for generation_count ← 1 to NUM_GENERATION do
4:          next_population ← elites from population
5:          while length of next_population < NUM_POPULATION do
6:              left, right ← GA_SELECT_TWO_PARENTS(population)
7:              off1, off2 ← GA_CROSSOVER(left, right)
8:              off1 ← GA_MUTATION(off1)
9:              off2 ← GA_MUTATION(off2)
10:             add off1, off2 to next_population
11:         population ← next_population
12:     return population
```

Figure 3 Genetic algorithm

IV.I Program representation

Our keyboard layout is on two-dimensional space. There are boundaries as rectangle R, which vertices are $(0,0), (0,H), (W,0), (W,H)$. 26 points representing alphabets A to Z lie inside the rectangle. If user touched at some site inside the rectangle, a point closest to the site triggers its alphabet. In this way, the rectangle is divided into a Voronoi diagram.

The solution is represented as array of 26 points $[(x_1,y_1), (x_2,y_2), \cdots, (x_{26},y_{26})]$. Each point (x_i,y_i) has constraint on $0 < x_i < W, 0 < y_i < H$.

IV.II Crossover

The idea started from dividing points into two parts, left and right. Simply two parts are crossed to make new generations. The pseudocode for crossover algorithm is in Figure 4.

Algorithm 1 Crossover

```
1:  procedure GA_CROSSOVER(p, q)                       ▷ offsprings for p and q
2:      loc1 ← p.locations
3:      loc2 ← q.locations
4:      common_loc ← common sites of loc1 and loc2
5:      common_left, common_right ← DIVIDE2 (common_loc)
6:      p_left_pos ← common_left
7:      q_left_pos ← common_left
8:      p_right_pos ← common_right
9:      q_right_pos ← common_right
10:     loc1_left, loc1_right ← DIVIDE2 (loc1 \ common_loc)
11:     loc2_left, loc2_right ← DIVIDE2 (loc2 \ common_loc)
12:     for each idx ∈ p \ common do
13:         if len(p_left_pos) < 13 then
14:             p_left_pos.add(idx)
15:         elsep_right_pos.add(idx)
16:     for each idx ∈ q \ common do
17:         if len(q_left_pos) < 13 then
18:             q_left_pos.add(idx)
19:         elseq_right_pos.add(idx)
20:     resoff1 ← newQueue()
21:     resoff2 ← newQueue()
22:     offspring1 ← newLayout()
23:     offspring2 ← newLayout()
24:     for each ALPHA ∈ alphabets_in_frequency_order do
25:         idx ← GETINDEX(ALPHA)                      ▷ in alphabetical order
26:         if LocationforALPHA ∈ p_left_pos then
27:             if LocationforALPHA ∈ p_right_pos then
28:                 offspring1[idx] = loc1[idx]
29:                 offspring2[idx] = loc2[idx]
30:             else
31:                 if not resoff2.isEmpty() then
32:                     pop from resoff2 and swap the location
33:                 elseresoff2.enqueue(idx)
34:         else Do symmetrically
35:     return offspring1, offspring2
```

Figure 4 Crossover algorithm

Call two parents as p and q. In detail, one offspring receives left part of p and right part of q. The other one receives left part of q and right part of p. For each location information, the algorithm recovers for alphabet information.

There are two main challenges. The first one is on line 31 in the Figure 5. There can be some alphabet (call it ALPHA), whose both location from each parent, have gone to single offspring. For this case, it recovers one of them (call it SITE1), and the other

one is reserved (call it SITE2). If other offspring has already reserved site SITE3, then the alphabet would be located to that SITE3. For this case, alphabet made reservation ago (call it BETA), had gotten only one site and made reserved site SITE3. It receives new site as SITE2. This means SITE2 from alphabet ALPHA and SITE3 from alphabet BETA is swapped.

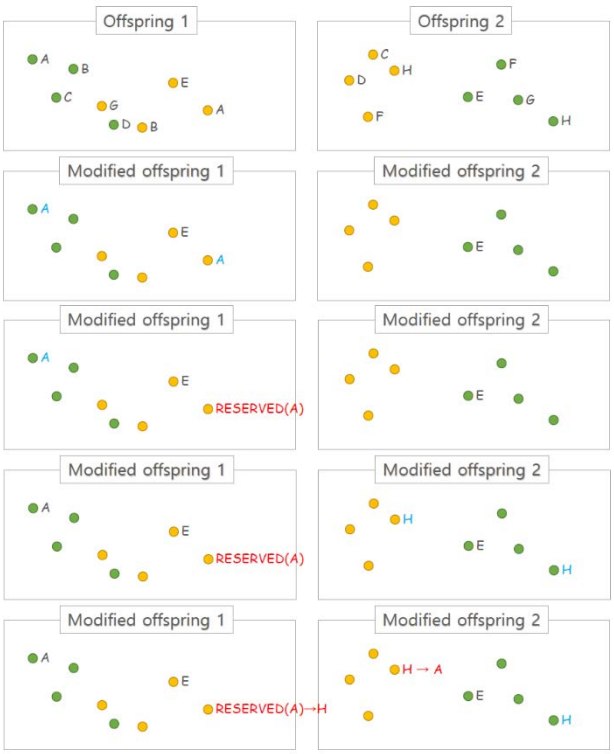


Figure 5 This simple example shows how the recovery is progressed. The recovery makes progress in order E>A>H>D>F>C>G>B. The topmost two figures are the raw offspring without any recovery. The lower figure has done some recoveries from upper one. Reservation site from A is used as site of H.

The second challenge was overlapping problem. Two layout that is similar to each other, and only some points are different in their location. Then, generation can cause duplication on same site for single offspring. We decided to consider making half separately for common sites and for uncommon sites. There is example represented in Figure 6.

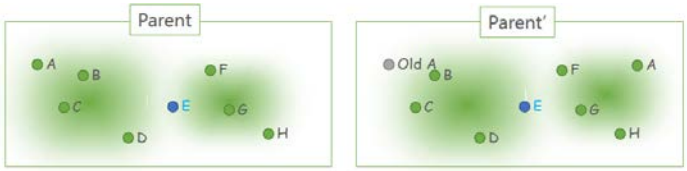


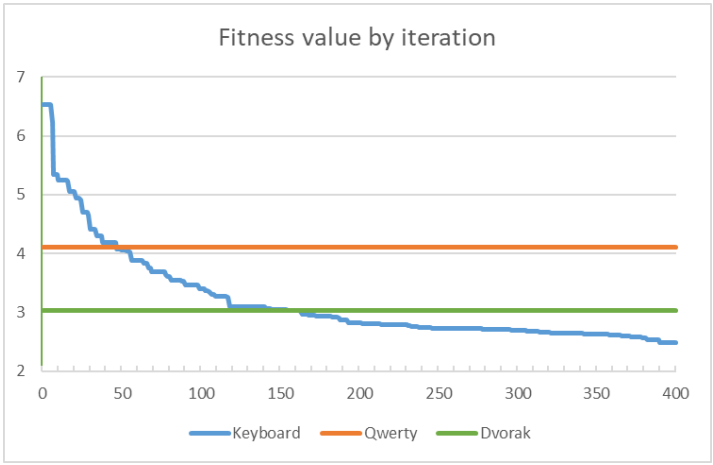
Figure 6 Single offspring would inherits the position of E twice. It makes disturbance on heredity.

IV.III Mutation

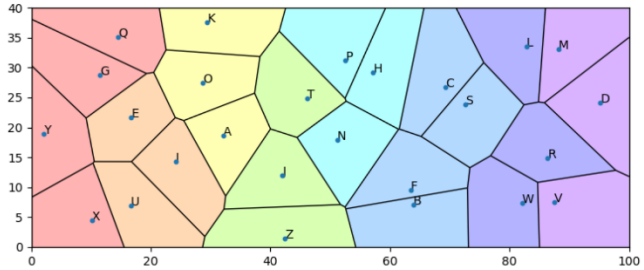
With probability 10%, mutation is progressed after crossover. There are two types of mutation, noise and swap. For each mutation, one of them is chosen randomly with the probability 50%, and is used.

For noise case, each coordinates of all 26 locations is changed by -5 through +5 following uniform random distribution independently. If it is going to be located outside of the rectangle R, the point is revised to the nearest boundary of the rectangle R. For swap case, with probability 60%, inner procedure (call it swap_inner) is invoked. Each swap_inner randomly selects two different alphabets, and swaps their location. And then repeat swap_inner with probability 60%. It seems to be infinite loop for rare probability. We limit whole count for invoking swap_inner as 40. In maximum, 40 swap_inner can be invoked.

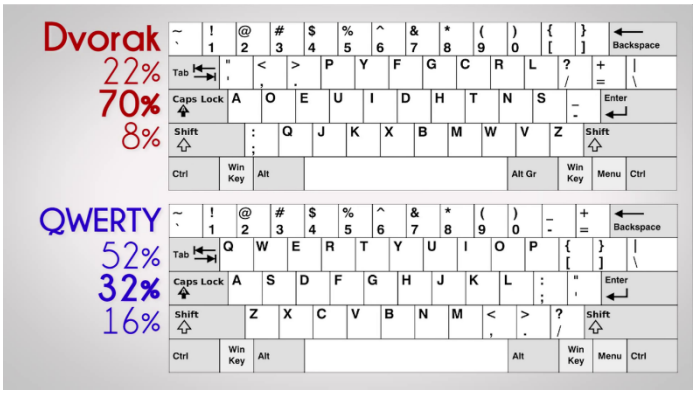
V. Result

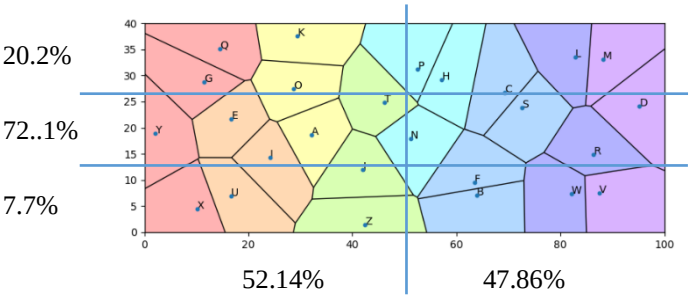


Final fitness value was 2.475. This value is better than Qwerty(4.111) and Dvorak(3.002).



We said that Dvorak is better than Qwerty keyboard because most frequent characters are located in the middle line. So we measured the key frequency distribution for our keyboard.





Every key has similar size and most frequent character are located on the middle line. The keys are evenly distributed left hand side and right hand side (52.14% for left, 47.86% for right) so each hand has similar amount of pressure. And because of middle area has bigger value than Dvorak (72.1%), fingers can stay on the middle area most of time.

VI. Discussion and Conclusion

Since our resulting optimized keyboard layout looks quite different from general keyboards, it is hard to intuitively decide our result is good or not. Therefore, the necessity of proper evaluation method exists. It seems that the best method for evaluating keyboard layout is actual testing of human hands. However, we could not adopt these methods because of some reasons. First, we had not enough time for actual human-hand testing because it will take long time for testers to adapt to new keyboard. Also, we could not produce keyboards on our own because of limitation of time and budget. Lastly, we cannot gurantee the result of human-hand testing because each person has different types of hands and typing habits. Alternative testing option was to simulating human hands by computer. However, it was impossible for us because of its difficulty and our lack of ergonomic and anatomical background knowledge. Therefore, our only option of evaluation was to evaluate our result by using our fitness function. To guarentee our fitness function is appropriate for evaluating, we showed that Dvorak and Korean-three-pairs keyboard is better than Qwerty and Korean-two-pairs keyboard. Of course it is not enough to guarantee our fitness function is right, but we assume that it is reasonable enough to use it as evaluation metric.

Further improvements can be done in three different ways. First, paramaeter optimization and changing genetic algorithm details including crossover and mutation will be possible. Then, solutions with better fitness function can be generated. Second, postprocessing resulting keyboard layout is possible. Since our resulting keyboard layout looks weired, and cannot be considered as good keyboard intuitively, we can improve it by modifying its appearance. Since we used voronoi digram, shapes of each keys of keyboard is convex polygon, which is not familiar to users who are familiar with rectangular shaped keys. Therefore, we might try rectangularize each shape of keys to make its appereance more familiar. Lastly, new evaluation methods can be done. We suggest a game as our new evaluation method to evaluate our keyboard layout. The game is designed for tablet and consists of three big process. The first step is proposing. At this step, client receives our layout result online. The second step is adaptation. The Voronoi diagram of the solution is drawn over the tablet, and each divided part is planned to touch. Offering same order in real corpus, the user tries to touch each divided part in a series. In this step, the user learns which partition represents which alphabet. The final step is evaluation. In evaluation step, the user processes a lot of touches with specific order. We can evaluate how fast the layout be applied. Making game attractive would be challenge for it. Making some interactions with human could be one solution for evaluation.

References

[1] Accot, J., Zhai, S. (1997). Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks. Proceedings of CHI '97, Atlanta, Georgia, ACM Press.