Novel Keyboard Layout Optimization

Michael C. Mortenson* and Cassandra S. Sweeten†
Brigham Young University, Provo, UT, 84602

Since the advent of the invention of the typewriter in 1897 by Shoes, Glidden, and Soule, keyboard layouts have been the subject of extensive research. Amidst many competing keyboard layouts, the 1972 QWERTY keyboard has gained traction and standard acceptance with the rise of personal computers. However, the various drawbacks of the QWERTY layout, which have been extensively researched and documented, have spurred research into alternate keyboard layouts. Herein, a method for optimizing a alternate 30-key keyboard is presented using a custom gradient-free genetic algorithm. The optimization relies on an effective distance objective function which incorporates digraph frequency analysis, effectively bypassing uncertainty in the literature regarding the commonly used Fitts-Digraph objective function. Finally, a resulting optimal keyboard is presented and suggestions for further inquiry are elaborated.

I. Introduction

KEYBOARDS are ubiquitous in today's society. With an increase in computer technologies, including touch screen computers, the need for an optimal keyboard has climbed. The now commonplace QWERTY keyboard was initially introduced for the typewriter in 1972, primarily as a means of slowing down the typing speed to reduce jamming in mechanical parts [1]. The mechanical issues of the keyboard have long since been resolved, but the QWERTY keyboard remains through force of habit. Arguments in favor of replacing the QWERTY layout point to research that poorly design keyboard layouts decrease efficiency, productivity, and potential health impacts [2–4]. While many might still consider replacing the QWERTY keyboard inconceivable, alternate keyboard layouts have gained a small measure of popularity, most notably the Dvorak keyboard, proposed by August Dvorak and William Dealey, which places the vowels in the home row [5].

Three main courses of inquiry are often pursued in keyboard optimization problems: ergonomic optimization (physical keyboard shape), ambiguous layout optimization (free-format key layout, especially popular in touch screen applications), and normal layout optimization (set-format key layout focusing on pairing characters with keys). This paper focuses on a normal layout optimization of a standard 3 x 10 grid, inspired by the current QWERTY layout which is shown in Fig. 1. One popular objective function for keyboard layout was initially proposed by Paul M. Fitts in 1954, known as the Fitts-digraph model or Fitts' law [6]. A digraph is a sequential combination of two characters. The Fitts-digraph model estimates typing time by taking into account key distances and digraph frequencies using empirically determined coefficients. These coefficients have been the center of much scholarly debate, so this research has opted to use the computationally inexpensive and physically unambiguous effective distance objective used by Bi et al. [7–9].

Q	W	E	R	T	Y	U	I	O	P
A	S	D	F	G	H	J	K	L	
	Z	X	C	V	B	N	M		

Fig. 1 Typical 30-key QWERTY Keyboard Layout

Within normal layout optimization, some have pointed out that keyboard optimization depends heavily upon the intended language of use [5, 7]. Additionally, the QWERTY keyboard layout favors left-handed users, with a disproportionate number of the most common letters on the left half of the keyboard. This work seeks to address both of

*Student, Mechanical Engineering Department.
†Student, Mechanical Engineering Department.

these challenges: intended language by the introduction of a C++ program for parsing through a text corpus for digraph frequencies and handedness by proposing a mirrored keyboard layout option.

As a final note, it must be mentioned that a keyboard layout design is partially objective and partially subjective. It can be objective in terms of the frequency the letters are used and how they are paired, as well as the ease of motion to then select certain letters and symbols, but the subjective side is shown through human preferences often based on aesthetics and ergonomics. While some have attempted to address this complexity through multi-objective optimization [8], the optimal JESAN keyboard presented in this work is based on the authors subjective sensibilities.

This report compiles the work put towards creating a novel keyboard optimizer. The problem's methodological background and development, as well as rationales for decisions made and the resulting keyboard layout optimization will be presented. Final results and findings will be presented and compared to similar works.

II. Methods

Optimizing to reduce the distance the fingers travel, we used an approach borrowed from Bi et al. [7] that uses an effective distance objective function (1). The effective distance equation is formulated as follows:

$$d = \sum_{i=1}^{n} \sum_{j=1}^{n} F_{ij} * D_{ij} \tag{1}$$

Often optimizations use the value of n is 26, corresponding to the 26 Roman characters of the English alphabet. F_{ij} is the frequency of the sequential combination of the i^{th} and j^{th} letters, known as the digraph frequency. D_{ij} is the distance between the centers of the i^{th} and j^{th} keys. This creates our objective function.

To create a matrix of digraph (two-character combination) frequencies, C++ script was created that takes an input .txt file and creates an F matrix of digraph frequencies. This script is adaptable to other input files and could be used to create a custom keyboard based on a specific text corpus. See Appendix A for an example F matrix of digraph frequencies for Lewis Carroll's *Alice's Adventures in Wonderland* and the C++ code that created it.

In exploring optimization methods to solve the objective function, both gradient-based and gradient-free approaches were considered. We selected the gradient-free approach to focus on developing further. Both methods are described below.

A. Gradient-Based Approach

First a continuous gradient-based formulation of the problem was considered. This continuous formulation used the x- and y-coordinates from each key as continuous design variables. The distances between keys were then calculated from the x- and y-coordinates for the center of each key. Assuming circular keys with constraints to keep the centers of each key a diameter apart from each other to avoid overlapping. Based on the predicted letter combination use, as provided in the "Digraph Frequencies" file, the objective function determined the effective distance for the keyboard layout.

By assuming circular keys, we were able to generate a matrix of constraints by ensuring that the centers of each key were at least one unit away from all over key centers. Inequality constraints were of the form:

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

where the x- and y-values represent key center coordinates. To account for every key the problem required 25^2 constraints. The effective distance objective function and the constraint function were optimized using MATLAB's native constrained optimizer, fmincon.m. Due to the additional number of constraints that would be needed to constrain the keyboard layout into 3 x 10 grid, this continuous formulation with gradient-based optimization was discarded as the primary optimization method. See Appendix B for code.

B. Gradient-free Approach

Due to troubles with constraints in the gradient-based optimization, we turned to a gradient free approach with discrete variables which avoided the need for constraints but required a different setup for the problem. Other researches have used gradient-free optimization approaches including simulated annealing [10, 11], colony optimization [12], and genetic algorithm optimization (GA) [13**?**]. For our gradient-free approach we selected a GA approach. Using the

same effective distance equation as the objective function, a predetermined keyboard format was introduced. This new formulation used a rectangular keyboard shape with 3 rows of 10 keys (30 keys total) instead of the floating circular design with only 26 keys. Indices 1 through 30 were used as discrete design variables. The 30 keys and 30 indices were correlated with the 26 letters of the English alphabet and 4 punctuation marks: hyphen (-), period (.), comma (,), and the question mark(?). Each key was given an index value based on a right-handed, alternating hand heuristic. The key indices are shown in Fig. 2.

18	16	14	12	20	19	11	13	15	17
8	6	4	2	10	9	1	3	5	7
28	26	24	22	30	29	21	23	25	27

Fig. 2 Key indices for 3 x 10 keyboard layout

The genetic algorithm follows a typical process of population generation, selection, crossover, and mutation. For population generation the two most common first letters of words in /text.itAlice's Adventures in Wonderland, the letters A and T, were set as a "seed" as the top two index positions to encourage a satisfactory output. The algorithm began with with an initial population of 200 randomized keyboard layouts. In the selection portion of the code, the effective distance was calculated and a mating pool was created. Child keyboard layouts were created using a tournament style crossover. Then the the keyboard layouts we standardized in the mutation phase to remove duplicates and add in missing values, ensuring each member of the population was a permutation of the numbers 1 through 30.A convergence check was then completed by comparing the average effective distance of the population against the best fitness in the population. The problem was considered converged when the population average and the population best were within an epsilon value of 10e-6. For comparison purposes, the results of the genetic algorithm were compared against results obtained using MATLAB's gradient-free Nelder-Mead optimizer, fminsearch.m. See Appendix B for code.

III. Results and Discussion

A. Gradient-based Results

The greatest advantage to the gradient-based method for the keyboard optimization problem was that it was relatively efficient in converging to a solution. However, the number of constraints necessary to create a normal layout result would have overwhelmed any efficiency gained from the gradient-based method.

With the continuous formulation of the problem, we didn't have as much control over the shape of the keyboard. However, the keyboard consistently formed a the honeycomb pattern as shown in Fig. 3. Although at times, the optimizer ignored the constraints and overlapped keys, the gradient-based method worked well. If our objective had been to pursue an ambiguous keyboard layout optimization, gradient-based optimization would have been the approach of choice. Despite the lack of consistency in the output keyboard layout, the gradient-based optimization was integral in understanding that the effective distance function has many local minima.

B. Gradient-free Results

Using Matlab's built in Nelder-Mead approach, fminsearch.m, the 30 key layout was optimized, outputting many solutions, one is shown in Fig. 4 below. Beginning with a random permutation of 30 indices, the average number of function calls to solve the problem based on a sample of 100 starting vectors was 729 function calls, based on a range of function evaluations from 479 to 6028 calls. This was likely due to the proximity of a given random starting point to an optimum.

To compare to the custom genetic algorithm, on average, the algorithm converged in 50 generations (see Fig. 5.) With 400 function calls made every generation, a average of about 20,000 function calls were made to solve the GA. Due to the objective function being so simple, this was not a significant problem sine the algorithm solved in less than 3 seconds on our machines.

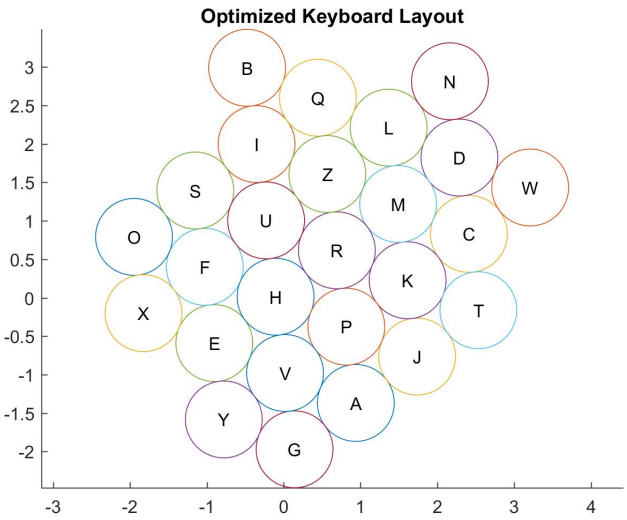

Fig. 3 A fmincon Optimized Keyboard

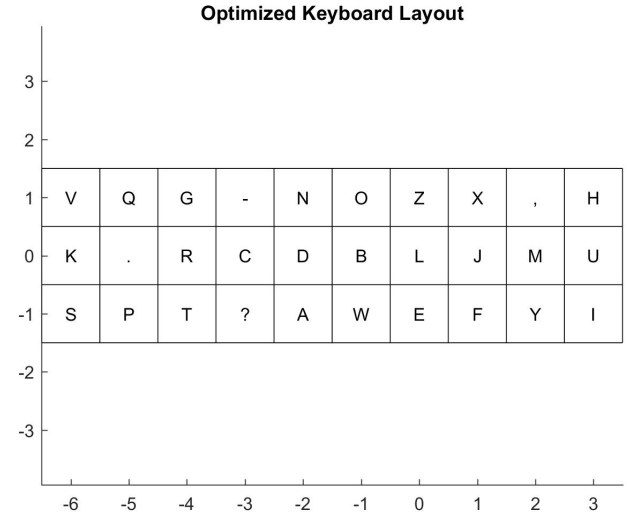

Fig. 4 A fminsearch optimized keyboard

A favorite optimized keyboard solution, the JESAN keyboard, is shown in Fig. 6.

IV. Conclusion

A common theme across all the optimizations was significant variation mainly due to the objective function's many local minima. The genetic algorithm compounds the output variety due to randomness in the starting population. In a way, this is useful because it allows for some human selectivity and aesthetic preference to come into play when choosing an optimal layout. On the other hand, it is frustrating to not have the optimizer land on the "best" solution every time. Something that may be worth developing in future code is a feature to prevent punctuation marks on the keyboard's home row to help assure more aesthetically pleasing keyboards. Since the overall keyboard aesthetics is a human judgement call, we chose to allow user preference. Future studies could study qualitative opinion of the desirability of the aesthetics of different layouts. Consistent with our aesthetic preferences, we chose a favorite optimal

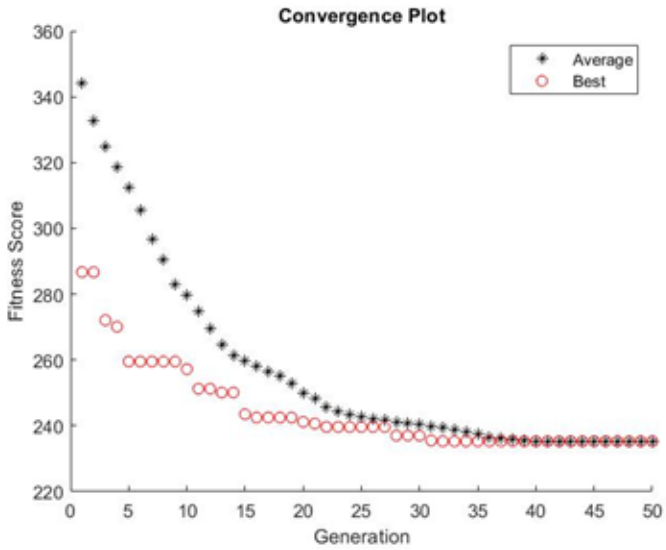

Fig. 5 Convergence of the custom genetic algorithm

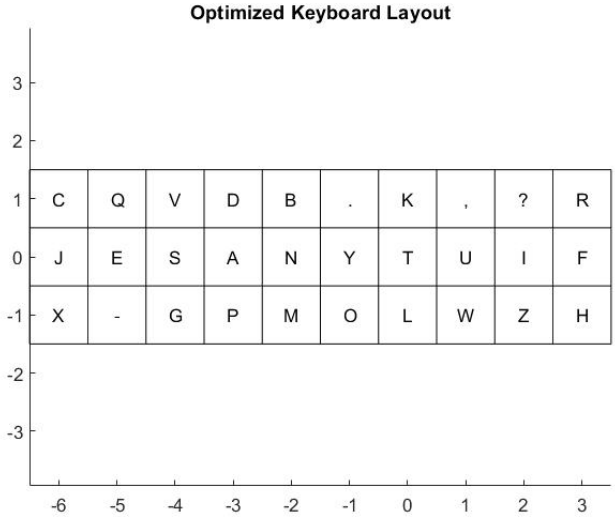

Fig. 6 JESAN Keyboard created with the custom genetic algorithm

keyboard: the JESAN keyboard (Fig. 6.)

There are many additional routes to consider in improving this optimization. To improve the genetic algorithm, different population sizes might be considered, as well as additional work on scaling the problem to try to decrease the likelihood of finding local minima. The digraph frequency code could easily be adapted to work with different text inputs. Foreign languages as well as different text formats, such as computer code or math-heavy writing, are attractive candidates. Additionally, the optimization could be expanded to include shift functionality (doubling characters to key locations) or to include another row of keys. Finally, a customized script could be created to track a user's keystrokes in real-time which could then be used to generate a personalized optimal keyboard. If the results of such personal optimizations could be aggregated, a truly universal optimal keyboard might be possible. Such future research remains untenable for now, but, should people's opinions of the QWERTY keyboard layout sour, one thing is certain: there will be an optimal keyboard layout ready to rise and take its place.

Appendix A

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	-	.	,	?
A	0.001	0.247	0.192	0.505	0.002	0.068	0.185	0.027	0.794	0.013	0.137	1.073	0.215	1.826	0.003	0.123	0.000	0.815	1.002	1.330	0.087	0.188	0.086	0.008	0.306	0.005	0.021	0.013	0.008	0.001
B	0.086	0.071	0.002	0.001	0.611	0.000	0.000	0.000	0.129	0.008	0.000	0.128	0.001	0.000	0.230	0.000	0.000	0.064	0.030	0.009	0.235	0.000	0.000	0.000	0.105	0.000	0.000	0.003	0.000	0.000
C	0.361	0.000	0.020	0.004	0.799	0.000	0.000	0.516	0.043	0.000	0.193	0.088	0.002	0.000	0.421	0.000	0.000	0.123	0.002	0.144	0.116	0.000	0.000	0.000	0.009	0.000	0.001	0.001	0.004	0.000
D	0.106	0.002	0.001	0.072	0.524	0.012	0.034	0.002	0.308	0.001	0.000	0.066	0.001	0.058	0.490	0.000	0.000	0.096	0.099	0.005	0.071	0.026	0.005	0.000	0.051	0.000	0.042	0.222	0.104	0.009
E	0.865	0.017	0.210	1.074	0.536	0.098	0.142	0.036	0.118	0.001	0.016	0.504	0.296	1.096	0.034	0.167	0.013	2.097	0.761	0.429	0.001	0.218	0.055	0.184	0.240	0.015	0.075	0.556	0.249	0.040
F	0.119	0.000	0.000	0.000	0.173	0.134	0.000	0.000	0.205	0.000	0.000	0.036	0.000	0.000	0.395	0.000	0.000	0.106	0.000	0.088	0.110	0.000	0.000	0.000	0.008	0.000	0.009	0.065	0.026	0.005
G	0.239	0.000	0.001	0.000	0.332	0.000	0.020	0.352	0.105	0.001	0.000	0.066	0.000	0.014	0.217	0.002	0.000	0.226	0.078	0.003	0.076	0.000	0.000	0.000	0.001	0.001	0.019	0.109	0.047	0.009
H	1.270	0.005	0.000	0.001	4.170	0.001	0.000	0.000	0.895	0.001	0.000	0.006	0.005	0.003	0.641	0.000	0.000	0.090	0.005	0.257	0.064	0.000	0.000	0.000	0.054	0.000	0.006	0.094	0.021	0.004
I	0.051	0.030	0.683	0.762	0.227	0.199	0.241	0.000	0.010	0.000	0.105	0.414	0.325	2.303	0.234	0.024	0.000	0.244	0.714	1.497	0.011	0.135	0.001	0.011	0.000	0.031	0.004	0.003	0.000	0.001
J	0.008	0.000	0.001	0.000	0.057	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.018	0.000	0.000	0.000	0.000	0.000	0.110	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
K	0.011	0.000	0.000	0.000	0.397	0.000	0.000	0.000	0.227	0.000	0.000	0.024	0.000	0.150	0.000	0.000	0.000	0.001	0.024	0.001	0.001	0.000	0.000	0.000	0.010	0.000	0.013	0.050	0.021	0.004
L	0.368	0.004	0.005	0.369	0.849	0.155	0.001	0.000	0.966	0.000	0.063	0.790	0.012	0.000	0.365	0.013	0.000	0.005	0.062	0.065	0.027	0.013	0.016	0.000	0.490	0.000	0.003	0.086	0.025	0.005
M	0.398	0.070	0.001	0.000	0.668	0.011	0.002	0.000	0.202	0.000	0.000	0.003	0.017	0.018	0.365	0.094	0.000	0.000	0.031	0.000	0.138	0.000	0.001	0.000	0.074	0.000	0.016	0.062	0.018	0.002
N	0.116	0.025	0.210	1.445	0.635	0.030	1.267	0.002	0.195	0.003	0.119	0.096	0.000	0.059	0.614	0.002	0.006	0.002	0.174	0.798	0.061	0.024	0.008	0.018	0.111	0.000	0.024	0.232	0.120	0.018
O	0.042	0.042	0.111	0.115	0.051	0.733	0.045	0.048	0.131	0.035	0.230	0.189	0.336	1.265	0.490	0.137	0.012	0.854	0.149	0.493	1.769	0.103	0.595	0.014	0.016	0.002	0.019	0.074	0.021	0.008
P	0.144	0.003	0.002	0.001	0.329	0.000	0.000	0.068	0.143	0.000	0.000	0.209	0.001	0.000	0.197	0.127	0.000	0.176	0.033	0.064	0.091	0.000	0.000	0.000	0.031	0.000	0.002	0.026	0.008	0.003
Q	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.236	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000
R	0.315	0.001	0.065	0.194	1.354	0.011	0.092	0.020	0.437	0.000	0.072	0.061	0.100	0.102	0.456	0.075	0.000	0.103	0.415	0.289	0.091	0.010	0.011	0.000	0.406	0.000	0.024	0.178	0.089	0.014
S	0.708	0.001	0.049	0.000	0.905	0.002	0.003	0.923	0.301	0.000	0.056	0.074	0.036	0.060	0.477	0.098	0.006	0.000	0.220	0.701	0.182	0.001	0.035	0.000	0.009	0.000	0.045	0.303	0.114	0.019
T	0.300	0.003	0.050	0.001	0.973	0.016	0.000	3.931	0.572	0.000	0.000	0.314	0.030	0.015	1.139	0.005	0.000	0.197	0.323	0.370	0.215	0.000	0.090	0.000	0.083	0.000	0.041	0.326	0.110	0.039
U	0.023	0.038	0.194	0.073	0.181	0.005	0.191	0.000	0.096	0.000	0.000	0.378	0.064	0.287	0.003	0.229	0.000	0.596	0.479	0.693	0.000	0.008	0.000	0.000	0.000	0.015	0.006	0.036	0.004	0.012
V	0.029	0.000	0.000	0.000	0.810	0.000	0.000	0.000	0.074	0.000	0.000	0.000	0.000	0.000	0.069	0.000	0.000	0.000	0.000	0.001	0.001	0.000	0.000	0.000	0.003	0.000	0.000	0.000	0.000	0.000
W	0.640	0.000	0.000	0.011	0.391	0.005	0.000	0.555	0.418	0.000	0.000	0.033	0.000	0.152	0.310	0.000	0.000	0.038	0.026	0.001	0.000	0.000	0.001	0.000	0.002	0.000	0.011	0.059	0.033	0.011
X	0.014	0.000	0.016	0.000	0.028	0.000	0.000	0.000	0.025	0.000	0.000	0.000	0.000	0.000	0.000	0.032	0.000	0.000	0.000	0.103	0.000	0.000	0.000	0.008	0.000	0.000	0.001	0.004	0.002	0.001
Y	0.014	0.011	0.002	0.005	0.107	0.001	0.000	0.000	0.056	0.000	0.000	0.005	0.006	0.001	0.604	0.062	0.000	0.023	0.048	0.043	0.000	0.001	0.003	0.000	0.003	0.001	0.033	0.205	0.088	0.015
Z	0.008	0.000	0.000	0.000	0.038	0.000	0.000	0.000	0.013	0.000	0.000	0.015	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.015	0.000	0.000	0.000	0.000
-	0.045	0.041	0.009	0.008	0.018	0.017	0.004	0.018	0.032	0.002	0.001	0.006	0.011	0.012	0.035	0.019	0.000	0.005	0.018	0.058	0.006	0.001	0.024	0.000	0.008	0.000	0.283	0.000	0.000	0.000
.	0.004	0.001	0.000	0.000	0.000	0.000	0.000	0.001	0.002	0.000	0.000	0.001	0.000	0.001	0.003	0.000	0.000	0.000	0.001	0.001	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.001	0.000	0.000
,	0.093	0.016	0.026	0.010	0.010	0.005	0.006	0.028	0.106	0.003	0.001	0.005	0.003	0.027	0.030	0.012	0.000	0.003	0.030	0.124	0.002	0.004	0.072	0.001	0.032	0.001	0.009	0.000	0.002	0.000
?	0.001	0.001	0.001	0.001	0.000	0.001	0.000	0.002	0.019	0.000	0.000	0.001	0.002	0.003	0.004	0.001	0.000	0.001	0.003	0.016	0.000	0.000	0.005	0.000	0.005	0.000	0.004	0.001	0.001	0.000

Here is the Digraph Frequency C++ code:

```
/* Digraph Frequency Analyzer
Michael Mortenson March 6, 2020

This program parses through a .txt file and generates
a matrix of digraph frequencies, which it then outputs
into a text document.

Created for ME EN 575
*/

#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <array>
#include <cctype>
#define UNICODE
#define _UNICODE

using namespace std;

int main(int argc, char* argv[])
{

// Total number of keys (or letters) to be analyzed
size_t const numKeys(30);



// Initialize digraph frequency array variables
double freqArray[numKeys][numKeys] = { 0 };
double totalDigraphs(0);


// Choose characters to go with keys
string allChars;
allChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ-,.?";
string chars = allChars.substr(0, numKeys);


// Parse input text file
if (argc < 3)
{
cerr << "Please provide the name of the input and output files";
return 1;
}
ifstream in(argv[1]);
if (!in)
{
cerr << "Unable to open " << argv[1] << endl;
return 2;
}
```

```
ofstream out(argv[2]);
if (!out)
{
in.close();
cerr << "Unable to open " << argv[2] << " for output";
return 3;
}


// Read in file
string word;

while (!in.eof())
{
getline(in, word, ' ');
if (word[0] == '\n') { continue; }

//If word is one character, start loop over
if (word.size() == 1) { continue; }


//Convert to Uppercase
for (size_t i = 0; i < word.size(); ++i)
{
word[i] = toupper(word[i]);
}

//Remove extraneous characters
for (size_t i = 0; i < word.size();)
{
if (chars.find(word[i]) == string::npos)
{
word.erase(i, 1);
if (i != 0)
{
i = i - 1;
}

}
else if (word[i] == '\n')
{
word.erase(i, 1);
if (i != 0)
{
i = i - 1;
}
}
else
{
i = i + 1;
}
}

//If word is one character or empty, start loop over
```

```
if (word.size() <= 1) { continue; }


//Cycle through characters of word to get digraphs
char char1;
char char2;
size_t indexChar1;
size_t indexChar2;


for (size_t i = 0; i < word.size() - 1; ++i)
{
char1 = word[i];
char2 = word[i + 1];
indexChar1 = chars.find(char1);
indexChar2 = chars.find(char2);
freqArray[indexChar1][indexChar2] = freqArray[indexChar1][indexChar2] + 1;
totalDigraphs = totalDigraphs + 1;
}
}

//Normalize digraph frequencies by the total number of digraphs
for (size_t i = 0; i < numKeys; ++i)
{
for (size_t j = 0; j < numKeys; ++j)
{
freqArray[i][j] = freqArray[i][j] * 100 / totalDigraphs; // multiply by 100 to get percentages

out << setw(15) << left << freqArray[i][j];
}
out << '\n';
}

//Close file streams
in.close();
out.close();

return 0;
}
```

Appendix B

Here is the Gradient-Based Method code:

```
clc; clear all; close all;

numkeys = 26;                       %Number of Keys, can be changed
position = rand(numkeys*2,1);       %Initial Guess

%-------Function Testing-------%
%[score] = obj(position);
%[c] = con(position);


%-------Digraph Weights Matrix-------%
global F
F = rand(numkeys,numkeys).*ones(numkeys,numkeys);       %To be replaced
F = round(F,1,'significant');


%-------fmincon settings-------%
func = @obj;
x0 = position;          %starting guess
lb = [];                %lower bound
ub = [];                %upper bound
options = optimoptions('fmincon');
options.Display = 'iter-detailed';
options.Algorithm = 'active-set';
options.PlotFcn = @optimplotfirstorderopt;
%options.PlotFcn = {@optimplotx, @optimplotfval, @optimplotfirstorderopt};
global mu;

%-------Generate Solution-------%
[xopt2, fopt2]  = fmincon(func, x0, [],[],[],[], lb, ub, @con, options);
xopt2 = [xopt2(1:numkeys),xopt2((numkeys+1):end)];
xcoords = xopt2(1:numkeys);
ycoords = xopt2((numkeys+1):end);

figure()
plotkeyboard(xcoords,ycoords)
hold off

%% FUNCTIONS

%-------Objective function: effective distance function-------%
function [score] = obj(position)
    numkeys = length(position)/2;
    global F

   %Relative Distances [D]
   D = zeros(numkeys,numkeys);
   for i = 1:numkeys
        for j = 1:numkeys
            D(i,j) = dist(position(i),position(i+numkeys), position(j),position(j + numkeys));
        end
    end
```

```
   %Score - function value to return
    score = 0;
    for i = 1:length(D)
        for j = 1:length(D)
            score = score + F(i,j).*D(i,j);
        end
    end
end


%-------Constraint Function-------%
function [c, ceq] = con(position)
    numkeys = length(position)/2;
    c = zeros(numkeys,numkeys);
    for i = 1:numkeys
        for j = 1:numkeys
            c(i,j) = (1 - dist(position(i),position(i+numkeys),position(j),
                position(j+numkeys))).*100;
        end
    c(i,i) = 0;
    end
    c = c';         %Transpose
    c = c(:);       %Turn Matrix to column vector
    ceq = [];
end


%-------Distance between 2 points-------%
function [d] = dist(x1, y1, x2, y2)
    d = sqrt( (x1 - x2).^2 + (y1 - y2).^2 );
end

%-------Plot the Keyboard-------%
function [] = plotkeyboard(x,y)
    hold on
    n = length(x);
    r = 0.5;
    for i = 1:n
        angle = linspace(0,2*pi);
        xpoints = r.*cos(angle) + x(i);
        ypoints = r.*sin(angle) + y(i);
        plot(xpoints,ypoints)
    end

    %Add letters to keys
    L = strsplit(sprintf('%c\n','A':'z'));  % Letter Labels
    text(x, y, L(1:length(x)), 'HorizontalAlignment','center', 'VerticalAlignment','middle')
    title("Optimized Keyboard Layout")
    axis equal
end
```

Here is the Custom Keyboard Genetic Algorithm Optimizer code:

```
%% Keyboard Genetic Algorithm Optimizer
% Michael Mortenson & Cassie Sweeten
% ME EN 575
%
% This program builds a optimum rectangular keyboard layout
% based on digraph (two-character combination) frequencies.
% The program uses a seed of the top two most common letters.
% A digraph frequency matrix can be created by running the
% DigraphFrequencyGenerator C++ program with an input .txt
% file.

clc; clear all; close all;

%-------Read in First Letter Frequency File-------%
Firsts = dlmread("FirstLetterFrequencies.txt"); %Must be in folder with the .m file

%-------Complete List of Keyboard Characters-------%
str = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ-,.?';
for i = 1:length(str)
    L(i) = strsplit(str(i));
end

%-------Select Anchor Letters-------%
firstIndex = find(Firsts == max(Firsts));
Firsts(firstIndex) = 0;
display("Anchor Letter RH: "  + str(firstIndex));
secondIndex = find(Firsts == max(Firsts));
Firsts(secondIndex)= 0;
display("Anchor Letter LH: " + str(secondIndex));

%-------Read in Digraph Frequency File-------%
global F;
F = dlmread("DigraphFrequencies.txt");  %Must be in folder with .m file

%-------D Matrix-------%
global D;

D = [0,3,1,4,2,5,3,6,2,1,1,3.16227766016838,1.41421356237310,4.12310562561766,2.236067977
49979,5.09901951359278,3.16227766016838,6.08276253029822,2.23606797749979,1.41421356237310,
1,3.16227766016838,1.41421356237310,4.12310562561766,2.23606797749979,5.09901951359278,
3.16227766016838,6.08276253029822,2.23606797749979,1.41421356237310;3,0,4,1,5,2,6,3,1,2,
3.16227766016838,1,4.12310562561766,1.41421356237310,5.09901951359278,2.23606797749979,
6.08276253029822,3.16227766016838,1.41421356237310,2.23606797749979,3.16227766016838,1,
4.12310562561766,1.41421356237310,5.09901951359278,2.23606797749979,6.08276253029822,3.16
227766016838,1.41421356237310,2.23606797749979;1,4,0,5,1,6,2,7,3,2,1.41421356237310,4.123
10562561766,1,5.09901951359278,1.41421356237310,6.08276253029822,2.23606797749979,7.07106
781186548,3.16227766016838,2.23606797749979,1.41421356237310,4.12310562561766,1,5.0990195
1359278,1.41421356237310,6.08276253029822,2.23606797749979,7.07106781186548,3.16227766016
838,2.23606797749979;4,1,5,0,6,1,7,2,2,3,4.12310562561766,1.41421356237310,5.09901951359278,
1,6.08276253029822,1.41421356237310,7.07106781186548,2.23606797749979,2.23606797749979,
3.16227766016838,4.12310562561766,1.41421356237310,5.09901951359278,1,6.08276253029822,
1.41421356237310,7.07106781186548,2.23606797749979,2.23606797749979,3.16227766016838;2,5,
1,6,0,7,1,8,4,3,2.23606797749979,5.09901951359278,1.41421356237310,6.08276253029822,1,
```

7.07106781186548,1.41421356237310,8.06225774829855,4.12310562561766,3.16227766016838,
2.23606797749979,5.09901951359278,1.41421356237310,6.08276253029822,1,7.07106781186548,
1.41421356237310,8.06225774829855,4.12310562561766,3.16227766016838;5,2,6,1,7,0,8,1,3,4,
5.09901951359278,2.23606797749979,6.08276253029822,1.41421356237310,7.07106781186548,1,
8.06225774829855,1.41421356237310,3.16227766016838,4.12310562561766,5.09901951359278,
2.23606797749979,6.08276253029822,1.41421356237310,7.07106781186548,1,8.06225774829855,
1.41421356237310,3.16227766016838,4.12310562561766;3,6,2,7,1,8,0,9,5,4,3.16227766016838,
6.08276253029822,2.23606797749979,7.07106781186548,1.41421356237310,8.06225774829855,1,
9.05538513813742,5.09901951359278,4.12310562561766,3.16227766016838,6.08276253029822,
2.23606797749979,7.07106781186548,1.41421356237310,8.06225774829855,1,9.05538513813742,
5.09901951359278,4.12310562561766;6,3,7,2,8,1,9,0,4,5,6.08276253029822,3.16227766016838,
7.07106781186548,2.23606797749979,8.06225774829855,1.41421356237310,9.05538513813742,1,
4.12310562561766,5.09901951359278,6.08276253029822,3.16227766016838,7.07106781186548,
2.23606797749979,8.06225774829855,1.41421356237310,9.05538513813742,1,4.12310562561766,
5.09901951359278;2,1,3,2,4,3,5,4,0,1,2.23606797749979,1.41421356237310,3.16227766016838,
2.23606797749979,4.12310562561766,3.16227766016838,5.09901951359278,4.12310562561766,1,
1.41421356237310,2.23606797749979,1.41421356237310,3.16227766016838,2.23606797749979,
4.12310562561766,3.16227766016838,5.09901951359278,4.12310562561766,1,1.41421356237310;1,
2,2,3,3,4,4,5,1,0,1.41421356237310,2.23606797749979,2.23606797749979,3.16227766016838,
3.16227766016838,4.12310562561766,4.12310562561766,5.09901951359278,1.41421356237310,1,
1.41421356237310,2.23606797749979,2.23606797749979,3.16227766016838,3.16227766016838,4.12
310562561766,4.12310562561766,5.09901951359278,1.41421356237310,1;1,3.16227766016838,1.41
421356237310,4.12310562561766,2.23606797749979,5.09901951359278,3.16227766016838,6.082762
53029822,2.23606797749979,1.41421356237310,0,3,1,4,2,5,3,6,2,1,2,3.60555127546399,2.23606
797749979,4.47213595499958,2.82842712474619,5.38516480713450,3.60555127546399,6.324555320
33676,2.82842712474619,2.23606797749979;3.16227766016838,1,4.12310562561766,1.41421356237
310,5.09901951359278,2.23606797749979,6.08276253029822,3.16227766016838,1.41421356237310,
2.23606797749979,3,0,4,1,5,2,6,3,1,2,3.60555127546399,2,4.47213595499958,2.23606797749979,
5.38516480713450,2.82842712474619,6.32455532033676,3.60555127546399,2.23606797749979,
2.82842712474619;1.41421356237310,4.12310562561766,1,5.09901951359278,1.41421356237310,
6.08276253029822,2.23606797749979,7.07106781186548,3.16227766016838,2.23606797749979,1,4,
0,5,1,6,2,7,3,2,2.23606797749979,4.47213595499958,2,5.38516480713450,2.23606797749979,
6.32455532033676,2.82842712474619,7.28010988928052,3.60555127546399,2.82842712474619;
4.12310562561766,1.41421356237310,5.09901951359278,1,6.08276253029822,1.41421356237310,
7.07106781186548,2.23606797749979,2.23606797749979,3.16227766016838,4,1,5,0,6,1,7,2,2,3,
4.47213595499958,2.23606797749979,5.38516480713450,2,6.32455532033676,2.23606797749979,
7.28010988928052,2.82842712474619,2.82842712474619,3.60555127546399;2.23606797749979,
5.09901951359278,1.41421356237310,6.08276253029822,1,7.07106781186548,1.41421356237310,
8.06225774829855,4.12310562561766,3.16227766016838,2,5,1,6,0,7,1,8,4,3,2.82842712474619,
5.38516480713450,2.23606797749979,6.32455532033676,2,7.28010988928052,2.23606797749979,
8.24621125123532,4.47213595499958,3.60555127546399;5.09901951359278,2.23606797749979,
6.08276253029822,1.41421356237310,7.07106781186548,1,8.06225774829855,1.41421356237310,
3.16227766016838,4.12310562561766,5,2,6,1,7,0,8,1,3,4,5.38516480713450,2.82842712474619,
6.32455532033676,2.23606797749979,7.28010988928052,2,8.24621125123532,2.23606797749979,
3.60555127546399,4.47213595499958;3.16227766016838,6.08276253029822,2.23606797749979,7.07
106781186548,1.41421356237310,8.06225774829855,1,9.05538513813742,5.09901951359278,4.1231
0562561766,3,6,2,7,1,8,0,9,5,4,3.60555127546399,6.32455532033676,2.82842712474619,7.28010
988928052,2.23606797749979,8.24621125123532,2,9.21954445729289,5.38516480713450,4.4721359
5499958;6.08276253029822,3.16227766016838,7.07106781186548,2.23606797749979,8.06225774829
855,1.41421356237310,9.05538513813742,1,4.12310562561766,5.09901951359278,6,3,7,2,8,1,9,0,
4,5,6.32455532033676,3.60555127546399,7.28010988928052,2.82842712474619,8.24621125123532,
2.23606797749979,9.21954445729289,2,4.47213595499958,5.38516480713450;2.23606797749979,
1.41421356237310,3.16227766016838,2.23606797749979,4.12310562561766,3.16227766016838,
5.09901951359278,4.12310562561766,1,1.41421356237310,2,1,3,2,4,3,5,4,0,1,2.82842712474619,

```
2.23606797749979,3.60555127546399,2.82842712474619,4.47213595499958,3.60555127546399,5.38
516480713450,4.47213595499958,2,2.23606797749979;1.41421356237310,2.23606797749979,2.2360
6797749979,3.16227766016838,3.16227766016838,4.12310562561766,4.12310562561766,5.09901951
359278,1.41421356237310,1,1,2,2,3,3,4,4,5,1,0,2.23606797749979,2.82842712474619,2.8284271
2474619,3.60555127546399,3.60555127546399,4.47213595499958,4.47213595499958,5.38516480713
450,2.23606797749979,2;1,3.16227766016838,1.41421356237310,4.12310562561766,2.23606797749
979,5.09901951359278,3.16227766016838,6.08276253029822,2.23606797749979,1.41421356237310,
2,3.60555127546399,2.23606797749979,4.47213595499958,2.82842712474619,5.38516480713450,
3.60555127546399,6.32455532033676,2.82842712474619,2.23606797749979,0,3,1,4,2,5,3,6,2,1;
3.16227766016838,1,4.12310562561766,1.41421356237310,5.09901951359278,2.23606797749979,
6.08276253029822,3.16227766016838,1.41421356237310,2.23606797749979,3.60555127546399,2,
4.47213595499958,2.23606797749979,5.38516480713450,2.82842712474619,6.32455532033676,
3.60555127546399,2.23606797749979,2.82842712474619,3,0,4,1,5,2,6,3,1,2;1.41421356237310,
4.12310562561766,1,5.09901951359278,1.41421356237310,6.08276253029822,2.23606797749979,
7.07106781186548,3.16227766016838,2.23606797749979,2.23606797749979,4.47213595499958,2,
5.38516480713450,2.23606797749979,6.32455532033676,2.82842712474619,7.28010988928052,3.60
555127546399,2.82842712474619,1,4,0,5,1,6,2,7,3,2;4.12310562561766,1.41421356237310,5.099
01951359278,1,6.08276253029822,1.41421356237310,7.07106781186548,2.23606797749979,2.23606
797749979,3.16227766016838,4.47213595499958,2.23606797749979,5.38516480713450,2,6.3245553
2033676,2.23606797749979,7.28010988928052,2.82842712474619,2.82842712474619,3.60555127546
399,4,1,5,0,6,1,7,2,2,3;2.23606797749979,5.09901951359278,1.41421356237310,6.082762530298
22,1,7.07106781186548,1.41421356237310,8.06225774829855,4.12310562561766,3.16227766016838,
2.82842712474619,5.38516480713450,2.23606797749979,6.32455532033676,2,7.28010988928052,
2.23606797749979,8.24621125123532,4.47213595499958,3.60555127546399,2,5,1,6,0,7,1,8,4,3;
5.09901951359278,2.23606797749979,6.08276253029822,1.41421356237310,7.07106781186548,1,
8.06225774829855,1.41421356237310,3.16227766016838,4.12310562561766,5.38516480713450,2.82
842712474619,6.32455532033676,2.23606797749979,7.28010988928052,2,8.24621125123532,2.2360
6797749979,3.60555127546399,4.47213595499958,5,2,6,1,7,0,8,1,3,4;3.16227766016838,6.08276
253029822,2.23606797749979,7.07106781186548,1.41421356237310,8.06225774829855,1,9.0553851
3813742,5.09901951359278,4.12310562561766,3.60555127546399,6.32455532033676,2.82842712474
619,7.28010988928052,2.23606797749979,8.24621125123532,2,9.21954445729289,5.38516480713450,
4.47213595499958,3,6,2,7,1,8,0,9,5,4;6.08276253029822,3.16227766016838,7.07106781186548,
2.23606797749979,8.06225774829855,1.41421356237310,9.05538513813742,1,4.12310562561766,
5.09901951359278,6.32455532033676,3.60555127546399,7.28010988928052,2.82842712474619,
8.24621125123532,2.23606797749979,9.21954445729289,2,4.47213595499958,5.38516480713450,6,
3,7,2,8,1,9,0,4,5;2.23606797749979,1.41421356237310,3.16227766016838,2.23606797749979,
4.12310562561766,3.16227766016838,5.09901951359278,4.12310562561766,1,1.41421356237310,
2.82842712474619,2.23606797749979,3.60555127546399,2.82842712474619,4.47213595499958,
3.60555127546399,5.38516480713450,4.47213595499958,2,2.23606797749979,2,1,3,2,4,3,5,4,0,1;
1.41421356237310,2.23606797749979,2.23606797749979,3.16227766016838,3.16227766016838,
4.12310562561766,4.12310562561766,5.09901951359278,1.41421356237310,1,2.23606797749979,
2.82842712474619,2.82842712474619,3.60555127546399,3.60555127546399,4.47213595499958,
4.47213595499958,5.38516480713450,2.23606797749979,2,1,2,2,3,3,4,4,5,1,0];

%-------Generate Initial Population-------%
global popSize;
global numKeys;
popSize = 200;
numKeys = 30;
population = zeros(popSize,numKeys);    %pre-allocate space

for i = 1:popSize
    %Generate random population. Each row is an individual.
    individual = randperm(numKeys);
```

```
    %Save values in first two slots
    slot1 = individual(1);
    slot2 = individual(2);

    %Find indices of most frequent letters
    topGuyIndex = find(individual==firstIndex);
    nextGuyIndex = find(individual == secondIndex);

    if (topGuyIndex ~= firstIndex || topGuyIndex ~= secondIndex)
        individual(topGuyIndex) = slot1;
        individual(1) = firstIndex;
    end

    if (nextGuyIndex ~= firstIndex || nextGuyIndex ~= secondIndex)
        individual(nextGuyIndex) = slot2;
        individual(2) = secondIndex;
    end
    individual
    population(i,:) = individual;
end


%Generate 100 Optimal Keyboards, plot top 10
%for l = 1:100


%-------Genetic optimizer-------%
generation = 0;
convergence = 0;
%while generation < 100
while convergence < 10

    %Compute obj for each individual of the population
    individualScore = zeros(popSize,1);
    for i = 1:popSize

        individualScore(i) = obj(population(i,:));
    end

    %Choose "Mating Pool"
    meanScore = mean(individualScore);
    pairs = randperm(popSize);
    parentIndex = zeros(popSize/2,1);
    for i = 1:length(pairs)/2
        if individualScore(pairs(2*i)) > individualScore(pairs(2*i-1))
            parentIndex(i) = pairs(2*i-1);
        else
            parentIndex(i) = pairs(2*i);
        end
    end
    for i = 1:length(parentIndex)
        newPopulation(i,:) = population(parentIndex(i),:);
    end
```

```
    %Crossover
    children = zeros(length(parentIndex),numKeys);
    for i = 1:length(parentIndex)/2
        crosspoint = randi(numKeys);
        children(2*i-1,:) = [newPopulation(2*i-1,1:crosspoint),
            newPopulation(2*i-1,(crosspoint+1):end)];
        children(2*i,:) = [newPopulation(2*i-1,1:crosspoint),
            newPopulation(2*i,(crosspoint+1):end)];
    end

    %Mutate (fix) children
    missingNum = [];
    doubleNum = [];
    missingIndex = [];
    doubleIndex = [];

    for i = 1:length(parentIndex)
        individual = children(i,:);
        for j = 1:numKeys
            k = find(individual == j, 2);
            if isempty(k) == 1
                missingNum = [missingNum, j];
            elseif length(k) > 1
                doubleNum = [doubleNum, j];
                doubleIndex = [doubleIndex, k];
            end
        end

        for p = 1:length(missingNum)
            individual(doubleIndex(2*p-1)) = missingNum(p);
        end
        children(i,:) = individual;
        doubleNum = [];
        doubleIndex = [];
        missingNum = [];
    end

    %Add children to population
    population = [newPopulation; children];

    %Increment generation
    generation = generation + 1;

%     %Plot Generation
    score = zeros(1,length(population));
    for i = 1:length(population)
        score(i) = obj(population(i,:));
        %plot(gen,score(i),'*')
    end
    best = min(score);
    avg = mean(score);
    diff = avg - best;
    if diff < 10e-6
```

```
        convergence = convergence + 1;
    end
    hold on
    plot(generation,avg,'k*',generation,best,'ro')
    xlabel("Generation")
    ylabel("Fitness Score")
    title("Convergence Plot")
    legend("Average","Best")
end

%bestKeyboard(l,:) = population(1,:);
%    l
%end

figure
plotKeyboard(population(1,:))


 %% Plot 10 Best Keyboards
%
%  for y = 1:10
%     k = find(best == min(best));
%   %Plot Keyboard Layout%
%     figure()
%     plotKeyboard(bestKeyboard(k,:))
%     best(k) = 10;
%  end

%% Functions Prison at the Bottom of the Sea

function [score] = obj(x)

    global F;
    global D;
    global numKeys;
    score = 0;

    %-------Calculate Score-------%
    for i = 1:numKeys
        for j = 1:numKeys
            score = score + F(i,j)*(D(x(i),x(j)));
        end
    end

end

%-------Plot Keyboard-------%
function [] = plotKeyboard(letterIndices)
global numKeys;

xKeyCoords =  [ 0, -3, 1, -4, 2 -5, 3, -6, -2, -1, ...
                0, -3, 1, -4, 2 -5, 3, -6, -2, -1, ...
                0, -3, 1, -4, 2 -5, 3, -6, -2, -1 ];
```

```
yKeyCoords =  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
                -1, -1, -1, -1, -1, -1, -1, -1, -1, -1];

hold on
    %Horizontal Lines
        xhor = [-6.5, 3.5];
        yhor = [-1.5, -1.5];
    plot(xhor,yhor,'k')
        xhor = [-6.5, 3.5];
        yhor = [-.5, -.5];
    plot(xhor,yhor,'k')
        xhor = [-6.5, 3.5];
        yhor = [.5, .5];
    plot(xhor,yhor,'k')
        xhor = [-6.5, 3.5];
        yhor = [1.5, 1.5];
    plot(xhor,yhor,'k')

    %Vertical Lines
        xver = [-6.5, -6.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [-5.5, -5.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [-4.5, -4.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [-3.5, -3.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [-2.5, -2.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [-1.5, -1.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [-.5, -.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [.5, .5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [1.5, 1.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [2.5, 2.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
        xver = [3.5, 3.5];
        yver = [-1.5, 1.5];
    plot(xver,yver,'k')
    axis equal;
```

```
    %Add letters to keys
    str = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ-,.?';
    for i = 1:numKeys
        L(i) = strsplit(str(letterIndices(i)));
    end

    text(xKeyCoords, yKeyCoords, L(1:numKeys), 'HorizontalAlignment','center',
        'VerticalAlignment','middle')
    title("Optimized Keyboard Layout")
end
```

References

[1] Zecevic, M. D., A., and Harburn, K., "An evaluation of the ergonomics of three computer keyboards," *Ergonomics*, Vol. 43, No. 1, 2000, p. 55–722.

[2] Lin, H. R.-H. C. J.-H., Ming-I B., and Ke, X.-M., "Usage position and virtual keyboard design affect upper-body kinematics, discomfort, and usability during prolonged tablet typing," *PLoS ONE*, 2015. https://doi.org/10.1371/journal.pone.0143585.

[3] McLoone, J. M. H.-C., Hugh E., Johnson, and W., P., "User-centered design and evaluation of a next generation fixed-split ergonomic keyboard," *Work*, Vol. 37, 2010, p. 445–456. https://doi.org/10.3233/WOR-2010-1109.

[4] Swanson, N. G., Galinsky, T. L., Cole, L. L., Pan, C. S., and Sauter, S. L., "The impact of keyboard design on comfort and productivity in a text-entry task," *Applied Ergonomics*, Vol. 28, 1997, pp. 9–16.

[5] Deshwal, P. S., and Deb, K., "Ergonomic Design of an Optimal Hindi Keyboard for Convenient Use," *2006 IEEE Congress on Evolutionary Computation*, 2006, pp. 2187–2194. https://doi.org/10.1109/cec.2006.1688577.

[6] Fitts, P. M., "The information capacity of the human motor system in controlling the amplitude of movement," *Journal of Experimental Psychology*, Vol. 47, No. 6, 1954, pp. 381–391. https://doi.org/10.1037/h0055392.

[7] Bi, X., Smith, B. A., and Zhai, S., "Multilingual Touchscreen Keyboard Design and Optimization," *HUMAN–COMPUTER INTERACTION*, Vol. 27, 2012, p. 52–382. https://doi.org/0.1080/07370024.2012.678241.

[8] Qin, Z. S. L. Y.-H. K. Y.-J. B. X., R., "Optimal-Tp: An optimized T9-like keyboard for small touchscreen devices," *ISS '18: Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces*, 2018, p. 137–146. https://doi.org/10.1145/3279778.3279786.

[9] Yazdia, M. A. A., Negahbanb, A., Cavuotoc, L., and Megahed, F. M., "Optimization of Split Keyboard Design for Touchscreen Devices," *INTERNATIONAL JOURNAL OF HUMAN–COMPUTER INTERACTION*, Vol. 35, No. 6, 2019, pp. 468–477.

[10] Li, Y., Chen, L., and Goonetilleke, R. S., "A heuristic-based approach to optimize keyboard design forsingle-finger keying applications," *International Journal of Industrial Ergonomics*, Vol. 36, 2006, pp. 695–704.

[11] Light, L. W., and Anderson, P. G., "Typewriter keyboards via simulated annealing," *AI Expert*, Vol. 27, 1993, p. 20.

[12] Wagner, Y. B. K.-S. F. D., Marc O., and Eggers, J., "Ergonomic modelling and optimization of the keyboard arrangement with an ant colony optimization algorithm," *Technical report*, 2001.

[13] Glover, D. E., "Solving a complex keyboard configuration problem through generalized adaptive search," *Genetic Algorithms and Simulated Annealing*, Vol. 27, 1987, pp. 12–31.