# Optimizing stylus keyboard layouts with a genetic algorithm: customization and internationalization

Chad R. Brewbaker
Dept. of Computer Science
Iowa State University
Ames, IA 50010
Email: crb002@iastate.edu
Web: `http://www.public.iastate.edu/˜crb002/`

***Abstract*— We explore the space of keyboard character assignments with evolutionary search. A simplified Fitts' cost model is used, and optimized key mappings are produced for Shakespearian text and Linux kernel source. Internationalization issues are discussed, and software for automatically generating UNICODE key mappings from sample text is produced.**

***Index Terms*— QWERTY, FITALY, genetic algorithm, keyboard optimization, Fitts' model**

## I. Introduction

The goal of this project was to create a genetic algorithm for doing optimizing keyboard layouts. After a brief literature review we decided to expand upon [1] to see if we could get similar performance to their spring model algorithm.

For evaluation we focus on two classes of typists. The first will be English language authors. For this corpus we will use the complete works of Shakespeare[27] from Project Gutenberg [23]. The second will be C programmers. For this corpus we will use the Linux kernel version 2.6.11.7[24].

## II. Physical layouts

For this study we will looked at two physical keyboard layouts. The first is the ubiquitous QWERTY keyboard.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $q$ | $w$ | $e$ | $r$ | $t$ | $y$ | $u$ | $i$ | $o$ | $p$ |
| $a$ | $s$ | $d$ | $f$ | $g$ | $h$ | $j$ | $k$ | $l$ | |
| $z$ | $x$ | $c$ | $v$ | $b$ | $n$ | $m$ | | | |

The second is the FITALY keyboard which uses a 6 by 5 matrix of square keys.

| | | | | | |
|---|---|---|---|---|---|
| $z$ | $v$ | $c$ | $h$ | $w$ | $k$ |
| $f$ | $i$ | $t$ | $a$ | $l$ | $y$ |
| | | $n$ | $e$ | | |
| $g$ | $d$ | $o$ | $r$ | $s$ | $b$ |
| $q$ | $i$ | $u$ | $m$ | $p$ | $x$ |

Both keyboards were modeled as integer coordinates corresponding to the rows and columns. Keys that are adjacent in a row or column have distance one. The Euclidean distance between keys will be cost for traveling between them.

## III. The Cost Models

We chose a modification of the Fitts' model[4] for this study. The Fitts' model was based on Shannon's work with entropy on information channels. It is a good model in that it takes into account things like size, distance, and frequency. The drawback is that it disregards any mental model of the language we are trying to type in.

This is the full Fitts model:

$t$ the average time cost per key press.

$P$ is the probability matrix of going from one key to another.

$D$ is the matrix of distances between keys.

$W$ is an array of key sizes.

$IP$ is the Fits performance index in bits per second.

$N$ is the size of our alphabet.

$$t = \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{P_{i,j}}{IP} Log_2(\frac{D_{i,j}}{W_{i,j}} + 1)$$

We simplify the model by setting $W = IP = 1$. This way we only need to input a corpus probability matrix and a keyboard distance matrix.

$$\hat{t} = \sum_{i=1}^{N}\sum_{j=1}^{N} P_{i,j} Log_2(D_{i,j} + 1)$$

The modified Fitts' model we have is for stylus input keyboards.

With some work the model can be transformed for two handed keyboards. According to Donald Norman and David Rumelhart extra optimizations for two handed keyboards are that:

-The loads on the right and left hands are equalized.

-The load on the home (middle) row is maximized

-The frequency of alternating hand sequences is maximized and the frequency of same finger typing is minimized.

Norman's model could fit on top of our modified Fitts' model by changing the keyboard distance matrix to include a load penalty, middle row bonus, and a same hand & finger penalty.

## IV. A Evolutionary algorithm for Keyboard layout

In [1] a monte-carlo algorithm on top of a spring model was used to come up with optimized key mappings. We propose the use of an evolutionary algorithm for the task. Evolutionary algorithms mimic the process of natural selection whereby members of a population with higher fitness have a higher probability of surviving to the next generation. We also simulate sexual reproduction by picking members of the population, exchanging information they contain, and passing them on to the next generation. Furthermore, offspring are mutated to give widen the search space.

Our approach is to encode every character as a real number between zero and one, and evolve this set of arrays. A static key index map from (1,..., No. of Keys) will be given to the physical keyboard, with each physical key given a unique number. The real value encoded array will be sorted, and the sorted index will correspond to a permutation. This permutation determines which character is mapped to which key. This encoding would also work with other real valued non-linear technique.

```
read in keyboard cost matrix
read in corpus probability matrix
for(runs=0;runs<5;runs++)
   randomly initialize population
   for(gens=0;gens<4000;gens++)
      evaluate fitness of population
      store genome and score of best member
      roulette selection to keep 24 members
      two point crossover for 12  members
      apply mutations to children of tpc
```

With roulette selection you select members of the population based on a roullete wheel model. Make a pie chart, where the area of a member's slice to the whole circle is the ratio of the members fitness to the total population. As you can see if a point on the circumfrence of the circle is picked at random those population members with higher fitness will have a higher probability of being picked. This ensures natural selection takes place.

Two part crossover can be thought of as the sexual reproduction. Two members are picked from the population and two points on their genome are taged for the crossover. Both children keep the portion of its genome within these two inexes, and exchanges the outside portion. Example:

```
Parent1:
ACCTTACCTTCA
Parent 2:
GATTACACCTGG
Xover points:
  x     x

Child 1:
GA|CTTACCT|TGG
Child 2:
```

```
AC|TTACACC|TCA
```

Mutations are applied to each of the two point crossover children by:

```
void mutate (member a)
i=rand();
switch(rand()%10){// pic a case
case 0: a[rand()%key_num]=drand();
  break;
  case 1:
  case 2:
  case 3:
  case 4: //swap two
    j=rand()%key_num;
    tmp = a[i];
    a[i]=a[j];
    a[j]=tmp;
    break;
  case 5:
  case 6:
  case 7:
  case 8:
  case 9: //Randomly tweak
    if(drand()>.5){
      a[i]+=.1*drand();
      if(a[i]>=1.0)
        a[i]=.99;
    }
    else{
      a[i]-=.1*drand();
      if(a[i]<=0.0)
        a[i]=.01;
    }
    break;
```

## V. Implementation Challenges

There were more than a few implementation challenges we hit upon. The first was having to use the ANSI C wchar.h library for Unicode characters. The learning curve wasn't too bad. Most familiar ANSI C functions are replicated by adding a w somewhere in the name. For example:

```
int isalpha(char)  vs int iswalpha(wchar\_t)
char fgetc(FILE*) vs wchar\_t fgetwc(FILE*)
```

The second challenge was to figure out what to do with common keys that are mapped by control sequences. For example, on a QWERTY keyboard the number of keys is reduced by having upper case letters be mapped by pressing the shift button before the desired character. For other keys multiple mappings are not quite as intuitive. We decided to handle double mappings of alpha keys in the standard way of hitting the shift key before an alpha key to get a capital letter. Functionality for this is provided in wchar.h by using iswalpha(), and towlower() functions.

## VI. Corpus preparation

We choose to generate our own corpora. We wanted corpora based on large pieces of text, and not just on a hundred or so pages. The Linux corpus contains 6,624,141 lines of text, the the Shakespeare corpus contains 123,404 lines of text.

Our first corpus was the complete works of William Shakespeare[27]. This was obtained from project Gutenberg[23]. The header and footer with copyright information was stripped along with copyright information embedded before each act. The a corpus of the alpha characters was constructed automatically converting upper case letters to lower case. Shift characters, punctuation, and spacing characters were ignored.

For our second corpus we used the Linux 2.6.11.7 kernel source. The first two lines were cut off the tarball we obtained from www.kernel.org. The resulting file had all files in the kernel concatenated.

Some error was introduced in our corpus preparation because of concatenated files. The character at the end of one file and at the beginning of the next adds effects our probability matrix. We disregard these errors as inconsequential.

## VII. Generating the corpus matrix

To generate the corpus matrix we wrote a program that inputs a Unicode text file and outputs a the transition probability matrix for symbols in the document. We used a two pass algorithm. On the first pass all symbols were put into a list.

For the second pass a table is kept counting the number of transitions from one letter to another. After this table is created it is normalized by taking sum of each row and dividing all elements of that row by the sum. This results in the creation of our corpus transition probability matrix.

## VIII. Constrained Keyboards

Our main experiment was generating constrained keyboards, i.e. keyboards with no blank keys. Standard refers to the standard keyboard mapping (QWERTY/FITALY), and random refers to the average score of 1000 randomly generated keyboard mappings. Here are the results:

Evolved QWERTY Shakespeare

score=42.2747

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $q$ | $u$ | $p$ | $m$ | $s$ | $n$ | $d$ | $k$ | $c$ | $x$ |
| $j$ | $r$ | $o$ | $t$ | $a$ | $i$ | $e$ | $v$ | $z$ | |
| $g$ | $f$ | $y$ | $h$ | $w$ | $l$ | $b$ | | | |

The evolved QWERTY Shakespeare keyboard puts OTAIE in the middle of the home row, this seems to fit with the English language where vowels A,I,E,O are frequently used. Infrequently used letters such as X,Z, and Q can be seen in the upper corners away from other letters.

Evolved QWERTY Linux

score=43.4699

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $k$ | $z$ | $e$ | $d$ | $v$ | $a$ | $m$ | $c$ | $x$ | $j$ |
| $w$ | $r$ | $n$ | $i$ | $t$ | $s$ | $p$ | $u$ | $q$ | |
| $g$ | $h$ | $f$ | $o$ | $l$ | $y$ | $b$ | | | |

The evolved QWERTY Linux keyboard also seems to be an intelligent design. The letters FOR are near each other. Also frequently used variable names like N and I are together on the home row.

Standard QWERTY

(Linux score=49.552376, Shakespeare score=51.069189)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $q$ | $w$ | $e$ | $r$ | $t$ | $y$ | $u$ | $i$ | $o$ | $p$ |
| $a$ | $s$ | $d$ | $f$ | $g$ | $h$ | $j$ | $k$ | $l$ | |
| $z$ | $x$ | $c$ | $v$ | $b$ | $n$ | $m$ | | | |

Evolved FITALY Shakespeare

score=41.041719

| | | | | | |
|---|---|---|---|---|---|
| $q$ | $u$ | $f$ | $g$ | $h$ | $c$ |
| $j$ | $p$ | $o$ | $s$ | $t$ | $x$ |
| | | $r$ | $a$ | | |
| $y$ | $m$ | $e$ | $l$ | $i$ | $w$ |
| $b$ | $z$ | $v$ | $d$ | $n$ | $k$ |

The evolved FITALY Shakespeare keyboard puts vowels A,I,O,and U on the upper main diagonal. Borrowing naming convention from standard FITALY by taking the second row we could call this keyboard JPOSTZ.

Evolved FITALY Linux

score=42.302176

| | | | | | |
|---|---|---|---|---|---|
| $k$ | $v$ | $s$ | $g$ | $h$ | $w$ |
| $x$ | $c$ | $t$ | $n$ | $i$ | $f$ |
| | | $a$ | $e$ | | |
| $q$ | $u$ | $l$ | $d$ | $r$ | $z$ |
| $j$ | $b$ | $m$ | $o$ | $p$ | $y$ |

The evolved FITALY Linux keyboard puts common variables like N, I and A in the middle. Another feature is that the word PRINT is contiguous except for a space between R and I.

Standard FITALY Linux score= 44.504894

Standard FITALY Shakespeare score=43.943166

| | | | | | |
|---|---|---|---|---|---|
| $z$ | $v$ | $c$ | $h$ | $w$ | $k$ |
| $f$ | $i$ | $t$ | $a$ | $l$ | $y$ |
| | | $n$ | $e$ | | |
| $g$ | $d$ | $o$ | $r$ | $s$ | $b$ |
| $q$ | $i$ | $u$ | $m$ | $p$ | $x$ |

The FITALY keyboard is very close, but not quite as good as the evolved keyboards. Both standard FITALY and the evolved FITALY keyboards soundly beat the random FITALY keyboard on both corpora.

| $keyboard$ | $corpus$ | $random$ | $standard$ | $GA$ |
|---|---|---|---|---|
| $QWERTY$ | $Linux$ | 50.1256 | 49.552376 | 43.4699 |
| $QWERTY$ | $Shakespeare$ | 50.563567 | 51.069189 | 42.2747 |
| $FITALY$ | $Linux$ | 47.782564 | 44.504894 | 42.302176 |
| $FITALY$ | $Shakespeare$ | 48.161224 | 43.943166 | 41.041719 |

As we can see from the table standard QWERTY is slightly better than random QWERTY for the Linux corpus. Also, the standard QWERTY is WORSE than random for the Shakespeare. This gives support to the urban legend that standard QWERTY was designed to slow typing in the English language.

It seems that our genetic algorithm was a success in evolving high performance keyboards under our modified Fitts' model. No computation we did took more than a few minutes on a PentiumIII 1Ghz. Memory requirements were not that taxing because only the population and cost matrices needed to be held in memory. Our optimization code takes O($n^2$) memory where $n$ is the number of keys on the keyboard. Thus, as long as we stay with small languages memory should not be a problem.

## IX. Evolved unconstrained keyboard

As an example of how our program can evolve unconstrained keyboards we fed in a 10 by 10 keyboard along with the Linux corpus.

Unconstrained 10×10 Linux keyboard

score=40.105400

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | $j$ | | | | | | | |
| $y$ | $p$ | $x$ | $k$ | $z$ | | | | | |
| $q$ | $u$ | $c$ | $e$ | $r$ | $w$ | | | | |
| $b$ | $s$ | $t$ | $d$ | $n$ | $g$ | | | | |
| | $l$ | $a$ | $o$ | $i$ | $h$ | | | | |
| | | $m$ | $v$ | $f$ | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Note that the genetic algorithm is robust enough to group the keys together. This allows us to do keyboards similar to the Metropolis keyboard of [1] that was evolved with a spring model.

Designs might be speeded up if we subtracted a sort of Gaussian distribution starting at the center of the matrix and going outward. This would further encourage keys to fall into this middle "well".

## X. Tutorial

This is a tutorial of how to use our keyboard mapping evolver. First you will need a corpus text and a 2D keyboard layout or pre-computed distance matrix..

First download your corpus and 2D keyboard layout:

```
bash%wget http://myCorpus.txt
bast%wget http://myKeyboard.layout
```

Now create your corpus transition matrix and keyboard cost matrix:

```
bash%corpus2matrix.exe myCorpus.txt
bash%keyboard2matrix.exe myKeyboard.layout
```

This process will create a file named corpus.dat and a file named keyboard.dat. If you have precomputed one of these you can copy them here now. Next we will evolve our keyboard.

```
bash%keyboarproj.exe
```

This produces an evolved keyboard mapping. The output is an array of integers representing the Unicode values of the best mapping found in order of how they map to the keyboard. Also output is the score and a time-stamp of when the keyboard was found. The pseudo-random number generator (a lagged Fibonacci variant) is seeded by the clock, so multiple runs will probably give you multiple keyboards.

Furthermore some extra tools are present. One of them is the program nbbymkeyboard.exe. This program takes two positive integer values and generates an $n$ by $m$ rectangualar keyboard model. Keyboards such as QWERTY and FITALY are very close to a rectangular matrix, so only a few changes are required.

Another helpfull tool might be unicode2int.exe. This program takes file of Unicode charactors and outputs a file of the charactors in integer form.

The other tool used for this paper is randomkeyscore.exe. Given a corpus probability matrix and a keyboard cost matrix it generates the average score of 1000 random key mappings. This was usefull in getting an idea how good our keyboard moiels were. Since we stripped the bitrate and key size out of the Fitts' model we no longer get a valid words per minute score, so we need a baseline to judge from.

The last tool used was keymapscore.exe. This tool takes a sequence of Unicode charactors as a mapping for a keyboard. It applies this permutation to the corpus probability matrix, and comes up with a modified Fitts' score for that keyboard mapping. This was usefull in determining the score of both the standard Fitts' mapping and the standard QWERTY mapping.

## XI. Internationalization

As the content for mobile devices like cell phones evolves toward the web and away from service providers it may soon be critical that a keyboard has localization. Consumers will migrate to devices with better local support as long as the learning curve isn't too great. With our keyboard evolver creating devices with local support could be as:

1. Generate a corpus of local text with the current crude keypad. Privacy and disk space could be helped by generating this probability matrix directly.
2. Use the corpus to evolve a localized key mapping.
3. Use the optimized key mapping a a marketing tool.

We doubt that cellular phone makers in the US will go beyond the keypad alphabetic order interface in the near future, but if they do it might be nice to have faster keypads.

As with all models our modified Fitts' doesn't take into account internal language representations. One reason US device provider's will be weary of switching key mappings is that most users know their "ABC's" and can find a key by its alphabetical order, even if they are not familiar with the keyboard.

Also, we must ask ourselves:,"What are the implications of patents and trademarks on keyboard models and mappings?" There could be a patent bonanza in the next few years coming up with optimized keyboard mappings. This could have a negative effect on consumers and service providers in 3rd world countries. Users would have to pay more for their service, and smaller local service providers would see their profits squeezed by license fees.

We believe that optimized keyboard models should be used as a marketing option, and not as patentable feature. As we have shown in this paper, generating an optimized key mapping will not take more than a few minutes given a corpus and keyboard cost model.

A quick search of the US Patent and Trademark office website yields 110,579 patents containing the word "keyboard".

Furthermore, will efficient algorithms for generating keyboard mappings be patented? There is a big debate going on in the EU right now about software patents. What effect will this have on keyboard optimization research?

In the least our tool could help those not familiar with a language to generate a decent keyboard for it. You may not speak Greek, but if you can find a body of greek text, and you have a keyboard, then you are set.

## XII. Future Work

We have thought of many ways to extend the project.

The first is to make use of our Unicode ability. Take a language like Armenian and generate an optimized keyboard for it. Get in contact with someone that uses the language and see what ingellegent exploits the keyboard found for the language.

One might try Hooke-Jeeves pattern search, Flocking, or other heuristic search non-linear optimization techniques. Also, the choices for parameter values in our genetic algorithm were arbitrary, so the reader might play around with them for better performance. Improved mutation and crossover operators would also be nice.

Another idea is to do a depth first search of the whole keyboard space. Naively the whole space is size n!, but with keyboard symmetry and pruning it might be tractable for small alphabets. The keyboard assignment problem is probably NP complete, and is at least as hard as graph isomorphism. A quick reduction to graph isomorphism:

Assume that your cost matrix is complement of your corpus matrix, and that they have (0,1) values. Of course the rows of the corpus matrix must add up to 1, so divide the rows of both your corpus matrix by their row sums. Now permute the keyboard distance matrix. The minimum cost mapping you can get is when the zeros of each matrix are aligned with the nonzero elements of the other matrix. This only happens in automorphisms of the identity permutation for the distance matrix, thus our problem is at least as bad as graph isomorphism.

Another idea is to use the full Fitts' model and develop a Voronoi keyboard. The Metropolis keyboard[1] uses the optimal centroidal voronoi tessellation for the uniform distribution on the plane in its design (hexagon/honeycomb tessellation). Given a corpus probability matrix evolve a set of points in the unit disk, or unit square, so that the Voronoi diagram of the points is the keyboard, and the Fitts' cost is minimized.

Simplifying the search space might be another strategy. State transitions which happen rarely could be dropped. Also, if the keyboard space is symmetric then that symmetry should be exploited in the search.

Further simplification could be done by putting the keys into clusters and doing a top down design. Take a small and let each "key" be one of these clusters. After the keys have been put into optimized clusters each cluster could then be broken apart and the keys inside given their own mapping.

Yet another idea: Initially, do a random assignment of letters to keys. Then, for each letter in the corpus do the following procedure:

1.Update the cost of that key press, i.e. the distance it had to travel. Add this value to both the current and last key.

2. If the new key pressed has higher cost than an adjacent key that is closer to the previous key pressed then swap the labels of the two keys.

Variations on this might yield some interesting keyboards, and might be faster than our optimization methods.

Yet another improvement would be to give a web interface for our UNIX utility. It would only have to consist of a website that took two parameters: a URL for the corpus text or pre-computed corpus matrix and a URL for the a 2d keyboard layout or a pre-computed keyboard cost matrix. It could output the Unicode characters in order in a text box at the bottom. The application shouldn't take more effort than a small pearl CGI script.

A downside to this is that people might start using your website for scientific computation. There are probably many problems that fit into the "permute a matrix to maximize its elementwise product sum with a second matrix" model.

Another direction would be to implement the a two handed model like the one outlined in the cost models section. It would be interesting to see how this changes the fitness landscape. The clusters partitions for each finger should be very interesting.

Source code of our project is located at http://www.public.iastate.edu/c̃rb002/ie574/code/. It is available for use under the General Public License version 2.0.

## References

[1] Zhai, Shumin; Hunter, Michael; and Smith, Barton A. *The Metropolis Keyboard- An Exploration of Quantitative Techniques for Virtual Keyboard Design*, ACM CHI letters 2(2),UIST'2000, San Diego, CA, USA

[2] Norman, D.A. and Fisher D.*Why alphabetic keyboards are not easy to use:Keyboard layout doesn't much matter*, Human Factors, 24, 509 - 5, (1982)

[3] Cooper, W.E., ed. *Cognitive aspects of skilled typewriting* 1983, Springer-Verlag: New York

[4] Fits, P.M. *The information capacity of the human motor system in controlling the amplitude of movement*, Journal of Experimental Psychology 47(1954), 381-391

[5] MacKenzie, I.S. *Spreadsheet calculation of QWERTY and OPTI keyboards*,1999

[6] MacKenzie, I.S., and Zhang, S.X.*The design and evaluation of a high-performance soft keyboard*, in Proc of CHI'99: ACM

Conference on Human Factors in Computing Systems. 1999 p.25-31

[7] Mayzner, M.S. an Tresselt, M.E. *Tables of singe letter and diagram frequency counts for various word length and letter position combinations*,Psychonomic Monograph Supplements, 1965, 1(2): p.13-32

[8] Silverberg, M. , MacKenzie, I.S. and Korhonen, P.*Predicting text entry speed on mobile phones*, in proc of CHI'2000: Human Factors in Computing Systems, 2000, The Hague, Netherlands, p.9-16

[9] MacKenzie, I.S and Soukoref, W. *Theoretical upper and lower bounds on typing speeds using a stylus and keyboard*, Behavior and Information Technology, 1995, 14:p.379-379

[10] Norman, D. A., (1983, September). *The DVORAK revival: Is it really worth the cost?*, cpnews: Consumer Products Tech Group: the Human Factors Society. 8, No. 3, 5 - 7.

[11] Norman, D. A. (1983). *On human error: Misdiagnosis and failure to detect the misdiagnosis*, Manuscript.

[12] Gentner, D. R., and Norman, D. A. (1984, March). *The typist's touch*, Psychology Today, 18, No. 3, pp. 66-72.

[13] Lewis, C., and Norman, D. A. (1986). *Designing for Error*, In D. A. Norman and S. W.Draper (Eds.), User Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

[14] Norman, D. A. (1990). *Commentary: Human error and the design of computer systems*. Communications of the ACM, 33, 4-7.

[15] Zhang, J. & Norman, D. A. (1995).*A representational analysis of numeration systems*, Cognition, 57, 271-295.

[16] TextwareSolutions, *The Fitaly one-finger keyboard*,1998

[17] Toby Ord and Alan Blair,*Exploitation and peacekeeping: introducing more sophisticated interactions to the iterated prisoner's dilemma*,Proceedings of the 2002 Congress on Evolutionary Computation, 1606–1611

[18] Daniel Ashlock, *Complex Adaptive Systems Text*, in preparation.

[19] Connor Ryan and Michel O'Neil,*Grammatical evolution papers*

[20] Robert Axelrod and William D. Hamilton, *The Evolution of Cooperation*, Science, New Series, Volume 211, Issue 4489 (Mar.27,1981)

[21] Robert Axelrod and Douglas Dion, *The Further Evolution of Cooperation*, Science, New Series, Volume 242, Issue 4884,(Dec. 9, 1988)

[22] Chad Brewbaker and Daniel Wengerhoff, *Optimization and Analysis with Centroidal Voronoi Tessellations*, NSF REU 2001 technical paper, Iowa State University, http://www.math.iastate.edu/reu/2001/voronoi_paper/voronoi_paper.html

[23] *Project Gutenberg*, http://www.gutenberg.org/about

[24] *The Linux Kernel Archives*,http://www.kernel.org

[25] Max Gunzburger, Qiang Du, V. Faber *Centroidal Voronoi tessellations: applications and algorithms*; SIAM Review 41, 1999, 637-676

[26] Aho, Sethi, and Ulman,*Compilers: Principals, Techniques, and Tools*,Addison-Wesley 1986, ISBN 0201100886

[27] William Shakespeare,*The complete works of William Shakespere*,Project Gutenberg,etext 100, downloaded April 15, 2005

[28] Sears, Andrew, Revis,Doreen, Swatski, Jean, Crittenden, Robert, and Schneiderman, Ben, *Investigating touchscreen typing: The effect of keyboard size on typing speed*, Behavior and Information Technology, 12(1), 1993, 17-22