Evolving a More Optimal Keyboard

Christopher P Walker

December 5, 2003

Abstract

The need for faster methods of data entry is undeniable. Unfortunately, little has been done since Dvorak's efforts in 1935 to improve the method of data entry into these machines. Therefore, new and improved keyboard layouts must be developed to make data entry more efficient. To aid in this task, evolutionary techniques were utilized to create a system capable of quickly creating optimized keyboard layouts for any language and typing style. The result was a system capable of creating a keyboard layout over 30% more efficient than the widely used QWERTY Layout, and only 3% less efficient than Dvorak's design in only a few hours. This system was designed to be easily expanded to suit multiple languages and typing styles, and could someday revolutionize the computing industry with user-specific keyboards specifically designed for their own typing styles.

Keywords

Keyboard Layout Optimization, Evolutionary Algorithms

Contents

1 Introduction

With the heightened emphasis of computers in every day life it becomes increasingly necessary for an efficient method of data entry. Instead of creating a completely new and possibly unwieldy device to solve this problem, it may be more efficient to merely optimize the tools we currently use for faster entry.

From 1931-35 Dvorak and Dealy tried to come up with a keyboard layout faster and more efficient than the widely known and used QWERTY (Figure 1). Using letter frequency and letter sequences, the result of their experiments is now known as the *Dvorak Simplified Keyboard* (Figure 2) [2]. Dvorak hypothesized that his keyboard layout could be mastered in approximately 50% of the time it took to master the QWERTY layout, would result in an average of 35% faster typing speeds, would allow for more accurate output and result in the typist to be less fatigued due to their fingers having to move a shorter distance per word [2]. Although they presented a potential successor to the QWERTY keyboard, what Dvorak and Dealy did not answer was the question "Is this the most efficient keyboard layout?"

Unfortunately, since Dvorak and Dealy little research has been done in optimizing the key layout of a keyboard. Research has, however, been done in optimizing irregular keyboard ([3]) and Chorded Keyboards ([1]). Substantial time and money has been placed into optimizing the physical look of the keyboard, which is evident by the number of "Natural" or "Ergonomic" keyboards currently on the market, but throughout these advancements the QWERTY layout has reigned supreme [2]. Even the layout proposed by Dvorak has had little success in taking users away from the QWERTY layout.

Fortunately, there are currently a few software packages available to remap the keys of a standard keyboard. They work by remapping the scan codes of the keyboard and acting as a "middleman" between the keyboard and the OS; reinterpreting the key pressed depending on user specifications. One such program is called *RemapperXP* by EuroFONT 2001. This program is freeware and is available at their website, *http://www.multifonts.com*. These utilities will allow anyone to remap their keyboard to whatever layout they wish, meaning the user is now free to find whatever keyboard layout works best for them.

After all this, the question remains, has the most efficient keyboard layout be found? Instead of studying letter frequency, this research proposes to turn the optimization over to the machine using Evolutionary Algorithms to find the most optimal keyboard layout. By quantizing the various factors that influence typing speed, the hope is to have the computer create an optimal keyboard layout without having to perform years of tweaking and research.

2 Algorithm Considerations

The three algorithms that were considered were combination enumeration, random search, and evolutionary algorithms. Since this topic hasn't been well researched, no other suitable algorithms could be found (nor were any avail-

Figure 1: The QWERTY Keyboard Layout

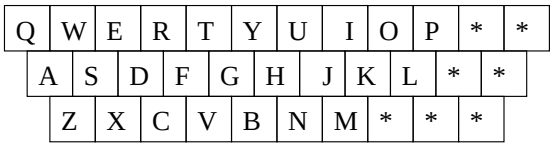
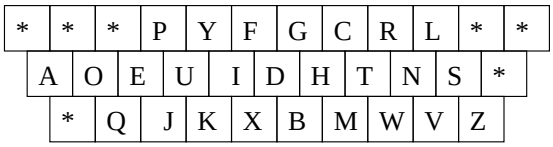


Figure 2: Dvorak's Keyboard Layout



able that could be easily adapted to this study). Unfortunately, combination enumeration would not have been possible, as there are a minimum 30! possible combinations of keyboard layouts for the main part of the keyboard alone. Random search was also ruled out, with it's near infinite runtime and complete lack of any type of history or knowledge.

The only logical choice to solve this problem is by using an evolutionary algorithm. This is because near optimal layouts should be within only a few key swaps of being completely optimal, so it's much more likely that a child of the near-optimal layout be the desired optimal layout than if the search space were navigated randomly. This method would also be quite a bit more memory and potentially time efficient than enumerating and evaluating every possible combination.

3 An Evolutionary Approach

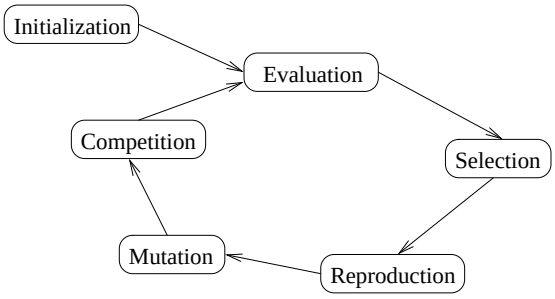
Evolutionary Algorithms are guided random search techniques inspired by modern evolutionary theory. They involve a population of candidate solutions to a problem, called individuals, which live, reproduce, and die based on their fitness relative to the rest of the population [4]. The general evolutionary algorithm consists of the following phases: initialization, evaluation, selection, reproduction, mutation, and competition (Figure 3). The last five stages of this process are repeated until the ending conditions are met [4].

3.1 The Individual

One critical piece of information for every EA is the design of the individual. In most implementation of an EA, the individual is a candidate solution to some problem the EA is trying to solve [4]. The solutions are then recombine, mutated and evolved to form new and hopefully better solutions.

The individual in this project is a keyboard layout. The keyboard layout is

Figure 3: The Evolutionary Cycle



represented by an array of 30 elements, each holding a unique key location on the keyboard. The indexes in the array correspond to the letter offset of the letter in the English alphabet. This is representation was used for a quicker key lookup for the fitness evaluations.

3.2 The EA's Design

Although the basic mechanisms remain the same for every EA, the underlying implementations may vary greatly depending on the design decisions and the specific characteristics of the problem [4]. Specifically, the selection, reproduction, and evaluation methods can change drastically between implementations. One must be extremely careful when selecting these methods, because they tend to have a great influence on time to convergence and chance of premature convergence.

The methods employed in the design of the EA were chosen very carefully based on how the individuals were evaluated. The basic idea for evaluation is to randomly select words from a source and have each keyboard virtually type the selected words. Those keyboards that type the words the fastest overall should perform better under normal circumstances than those that don't. This idea will cause a number of design issues, which will be discussed in the following subsections.

3.2.1 Two Population Pools

In order to reduce the necessary computation time, a two pooled approach was used. For a one pooled approach, each individual would have to be evaluated for every possible word for it's initial evaluation. For relatively large populations sizes, this is not computationally feasible. Instead, only a small subset of the words are used for evaluation each generation and a maturity requirement was placed on the individuals. The new members of the population are placed into a separate "child pool" until they reach a certain age when they are taken from the child pool and placed into the parent pool. While in the child pool, individuals are not allowed to reproduce or be killed.

3.2.2 Competition

The competition strategy used was strictly elitist. The adult population size was fixed, and at each generation the population was allowed to create a fixed number of children. After the children were created for the generation those old enough from the child pool were promoted to the adult pool. The adult pool was then sorted, and resized to the popultion limit so that the weakest members of the adult population were killed. There were no restrictions placed on the child population.

3.2.3 Selection

A Linear Ranking Method was used for the reproductive selection scheme. Rank-based selection schemes assign probability based on the rank of an individual in a given population. For Linear Rank-based selection, there is a linear correspondence between the probability and the rank of the selection [4]. For this EA, the probability of reproduction for any specific individual is given by Equation 1. In this formula, β_{rank} and α_{rank} correspond to the expected number of offspring for the most and least fit individuals respectively.

$$Pr_{lin\_rank}(i) = \frac{\alpha_{rank} + [rank(i)/(\mu - 1)](\beta_{rank} - \alpha_{rank})}{\mu} \tag{1}$$

3.2.4 Reproduction

Reproduction is the method evolutionary algorithms use to generate new potential solutions. This process is generally broken down into two parts: crossover and mutation [4]. Depending on the problem, one or both of these methods maybe useful in the creation of new individuals. In this case, no suitable crossover method could be found, so mutation became the sole genetic operator.

Mutation was handled by swapping locations of letters on the keyboard. The mutation operator would choose a random number (the number being based on the mutation rate) of key locations on the keyboard. The operator would then randomly select another location from the set of all possible locations, and the values at those locations would be swapped. To reduce the probability of parent duplication, the mutation operator was required to swap at least one pair of keys.

Since the initial population was bound to be much worse than the later ones, the earlier generations would be able to accept much more mutation than the later ones. To take advantage of this, two methods were devised to vary the mutation rate throughout the course of the execution. One method borrowed ideas from Simulated Annealing and would gradually reduce the mutation rate, or "heat", of the system as time increased. Although this method would work logically, determining an acceptable falloff rate was found to be fairly tricky. Instead, a method similar to Rechenberg's "1/5 Success Rule" was developed which allowed the mutation rate to vary according to the rate at which new

individuals were accepted into the population. This allowed the mutation rate to increase or decrease, depending on how well the current generation mutated, completely removing any major design decisions regarding mutation rate falloff.

In the end, the mutation rate was allowed to vary within the interval [0.1, 0.9], depending on the acceptance rate of new individuals into the population. If the acceptance rate fell below a specific threshold, then the mutation rate was decreased, otherwise it was increased.

3.2.5 Evaluation

Instead of attempting to score the individuals based on an estimate of how well they would perform, the individuals were evaluated based on simulated "real word" typing excersices. Each individual "types" a selection of words according to a Virtual Typing Algorithm (Algorithm 1) derived for this experiment. The general idea behind this algorithm is the quality of the keyboard is dependant only on pairing of individual letters, so the individual should be rewarded if it pairs commonly seen paired letters in specific ways. The reward function is described further in Section 3.2.5.

In order to have an adequate understanding of how well a specific keyboard layout will perform, the keyboard must "type" a large selection of words. This is not time effective, since a large amount of time would be wasted on evaluating unfit candidates. The two population pools were used to fix this problem, which would speed up the execution time of each generation and hopefully reduce the amount of time wasted on unfit individuals.

Algorithm 1 Virtual Typing

procedure Virtual_Type(keyboard, word)
  $total_score \leftarrow 0$
  for $i \leftarrow 0$ *to* $length(word) - 1$ **do**
    $current\_score \leftarrow DetermineReward(word[i], word[i+1])$
    $total\_score \leftarrow total\_score + current\_score$
  end for
  $total\_score \leftarrow total\_score/(length(word) - 1)$

3.2.6 Evaluation Criteria

Invidiuals will be rewarded and penalized according to various rules that should result in faster typing speeds. Generally, when typing most words on a fast keyboard a person should alternate hands for each letter, fingers if two letters are on the same hand, not have to move the pinky from the home row, and the most used letters should be placed on the most powerful fingers (mainly the first and middle fingers). These factors were taken into account when coming up with an evaluation criteria. An individual was awarded a certain number of points for each "good" feature found while typing two consecutive letters, and penalized another amount of points for each "bad" feature found in the

Table 1: Evaluation Criteria

Event	Points Awarded
Using the Same Finger	-2
Using the Same Hand, Different Fingers	1
Different Hands	3
Using a Key on Home Row	2
Using the Pinky Not on Home Row	-1.5
Using the First Finger	.75
Using the Middle Finger	.5
Using the Ring Finger	-.1
Using the Pinky Finger	-.2

keyboard layout. Table 1 contains a list of all of the criteria checked for in while each word was being typed. Points only stacked for mutually exclusive conditions (for instance, different hands did not stack with different fingers, but different hands did stack with using the first finger).

Once the sum of these points was found, the scores were then normalized over the length of the word and then averaged into the total fitness of the individual.

4 Experiment Specifics

The specific parameters used for the experiment are listed in Table 2. The Fit/Unfit Reproduction Ratio were used in the selection equation (they were factors in determining α_{rank} and β_{rank}), and the new child ratio was the value multiplied by the total population to determine the maximum number of children to create in one generation. The "Desired Child Acceptance Rate" attribute is the percentage of the total population that should be replaced by children each generation.

A strictly random seeding was used to populate the initial population. There were no predefined layouts, nor were they any hardcoded "hints" to help the EA alone. The initial population was randomly generated, evaluated for five generations, moved to the adult pool and the EA proceeded to reproduce, mutate, and evolve.

The top 1000 used words in the English language were used to evaluate the individuals. This should optimize the keyboard for the most common used words, which would result in faster typing overall.

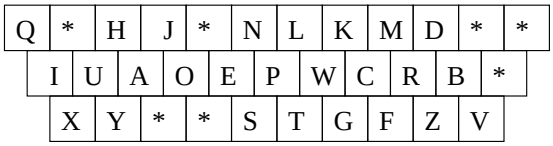
5 Results

Figure 4 is the most optimal keyboard layout generated by the EA. This individual appeared after 305,780 generations and remained the top individual for another 30,000 generations after that. Table 3 lists the fitness values of

Table 2: Experiment Parameters

Parameter Name	Value
Population Size	2000
Initial Key Mutation Rate	.25
Minimum Age	5 Generations
Fit Reproduction Ratio (of population size)	1.0
Unfit Reproduction Ratio (of population size)	0.01
New Child Ratio (of population size)	1.5
Desired Child Acceptance Rate	0.2

Figure 4: Walker's Evolved Keyboard Layout



Walker's Evolved Keyboard Layout and compares them to the fitness values for the QWERTY and Dvorak Keyboard Layouts.

Although the layout does not quite beat the Dvorak Keyboard layout, it is a significant improvement over the standard QWERTY Keyboard. It is also interesting to observe that although the Walker and Dvorak Keyboard layouts look significantly different, they do share a number of common traits. For instance, both keyboard layouts place all of the vowels on the same hand along home row. This is most likely due to the frequency of vowels when compared to the infrequency of repeated vowels. It is also interesting to note the location of non-alphabet symbols (denoted by the asterisks) in the evolved keyboard, placing them so that the stronger fingers have access to the more frequently used characters, but don't have to be held responsible for typing the maximum number of letters allowed.

6 Conclusions

An Evolutionary Algorithm was designed and developed to generate near optimal keyboard layouts. The algorithm was successful in creating a significantly

Table 3: Experiment Results

Keyboard Layout	Fitness Evaluation
QWERTY Keyboard	2.4910
Dvorak Keyboard	3.3659
Walker Keyboard	3.2667

more efficient keyboard layout than the standard QWERTY layout, which also came very close to beating the Dvorak Keyboard.

While the implementation was English-specific, the design was generic enough so that it could easily be expanded to different alphabets with little more than a words list and a modification of the individual. The individual could also be modified to handle keyboard optimization for non-standard typists, who cannot type using the "correct" method for one reason or another. Eventually, this code could be adapted to take into account a single person's typing style (including finger strengths and weaknesses) and a personalized keyboard could be evolved to take advantage of their style.

For now, the Dvorak Keyboard remains one of the most efficient known keyboard layouts, even if it isn't the most widely used.

References

[1] Daniel Gopher and David Raij. Typing with a two-hand chord keyboard: Will the qwerty become obsolete? *IEE Transactions on Systems, Man, and Cybernetics*, 18:117–122, July 1998.

[2] Jan Noyes. Qwerty - the immortal keyboard. *Computing & Control Engineering Journal*, pages 117–122, 1998.

[3] B. J. Oommen and J. R. Zgierski. Keyboard optimization using genetic techniques. 1991.

[4] D B Fogel T Back and T Michalewicz. *Evolutionary Computation 1 Basic Algorithms and Operators*. Institute of Physics Publishing, Philadelphia, PA, USA, 2000.