Universitat Politècnica de València

DSIC - Departament de Sistemes Informàtics i Computació

Academic Project

MIARFID - Master's in Artificial Intelligence and Pattern Recognition

Optimization of Keyboard Layouts for English

Author:
Jordi Cantavella Ferrero

Academic Year 2025-2026

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

The layout of current keyboards is not optimized for ergonomics and typing efficiency. Current layouts have their roots in traditional typewriting and have not evolved to meet the modern needs of users. Layouts like QWERTY, which emerged in 1873, were designed for mechanical typewriters and are no longer the most efficient for typing on digital devices.

Later studies showed that the QWERTY layout generated inefficient movements, contributing to fatigue and the risk of injury. This led to subsequent layouts such as Dvorak (1936) and Colemak (2006), which aimed to improve efficiency and reduce user fatigue. Seeking to maximize typing speed and minimize physical effort, these layouts are based on ergonomic principles and linguistic analysis.

1.2 Objectives

The main objective of this work is the optimization of keyboard layouts, aiming to generate ergonomic and efficient keyboard arrangements. To achieve this, a genetic algorithm and simulated annealing will be implemented to explore the space of possible layouts and find optimal configurations based on ergonomic and linguistic criteria.

- Implement a genetic algorithm for generating and evaluating different keyboard layouts.

- Implement a simulated annealing algorithm for the same problem.
- Define evaluation metrics that consider distance, hand alternation, and finger effort.
- Analyze the results obtained by the algorithms.
- Compare the generated layouts with existing ones such as *QWERTY*, *Dvorak*, and *Colemak*.

1.3 Scope and Limitations

Given the time and resource constraints, this work will focus on optimizing keyboard layouts for the English language. Aspects related to the physical implementation of keyboards or adaptation to different devices will not be addressed.

Also given the different characteristiscs of each distribution, the optimization will be focused on the most common Latin alphabet layouts, including all the letteres, and the most common punctuation marks.

Chapter 2

Theoretical Framework

2.1 History of Keyboard Layouts

Over time, different types of keyboards have been used, from the first mechanical keyboards to modern digital ones. Among them, the QWERTY, Dvorak, and Colemak keyboards have been particularly prominent.

The **QWERTY** keyboard, designed in 1873 by Christopher Latham Sholes, was created to prevent the keys of early mechanical typewriters from jamming. Its design prioritized separating the most frequently used letters in English to reduce typing speed and prevent the typebars from clashing.

The **Dvorak** keyboard, developed in 1936 by August Dvorak and his brother-in-law William Dealey, was designed to improve efficiency and reduce user fatigue. It demonstrated a reduction of up to 35% in finger movement compared to the QWERTY layout by placing the most used letters on the home row and promoting hand alternation.

The **Colemak** keyboard, introduced in 2006 by Shai Coleman, is an evolution of the QWERTY keyboard that aims to improve efficiency without requiring a radical change in key arrangement. Colemak keeps many keys in their original positions, making the transition easier for users accustomed to QWERTY. It focuses on minimizing finger movement and maximizing typing speed by placing the most common letters in accessible positions.

As shown in Figure 2.1, each of these layouts takes a different approach to ergonomics and efficiency, reflecting the needs and technologies of its time.

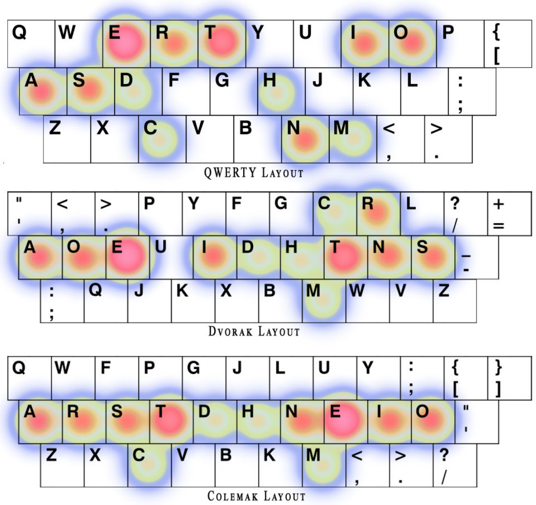


Figure 2.1: Comparison of the QWERTY, Dvorak, and Colemak keyboard layouts.

Representing the different keyboard layouts, Figure 2.1 illustrates how each of these arrangements seeks to optimize the typing experience from different perspectives. Note the concentration of the most used letters on the home row of the Dvorak and Colemak keyboards, in contrast to the more scattered arrangement of the QWERTY keyboard.

2.2 Keyboard Ergonomics

Following the main layouts introduced in section 2.1, keyboard ergonomics is a crucial aspect in the design of keyboard layouts. The hand has a series of anatomical characteristics that influence comfort and efficiency when typing. Among them, we highlight:

- **Finger Strength:** The strength of the fingers varies, with the thumb and middle finger being the strongest, while the pinky is the weakest.

- **Finger Mobility:** Fingers have different ranges of motion, with the thumb and index finger being the most mobile.

- **Muscle Fatigue:** Prolonged use of certain fingers can lead to muscle fatigue, especially if constant effort is required.

- **Hand Posture:** A natural and relaxed hand posture is essential to avoid strain and long-term injuries.

2.3 Linguistic Analysis

Linguistic analysis is fundamental for designing keyboard layouts that optimize typing efficiency. Paying attention to the frequency of letters and their distribution will allow us to design keyboards that minimize finger movement and maximize typing speed.

Figure 2.2 shows an analysis of letter frequency in English.

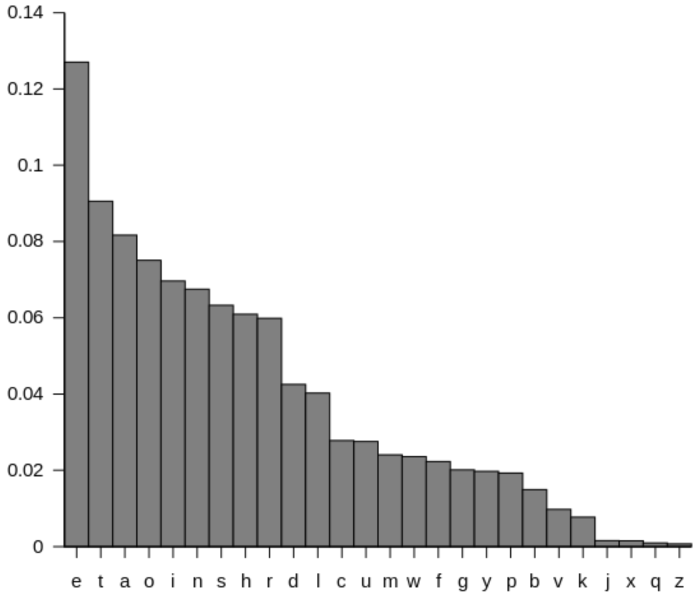


Figure 2.2: Frequency of letters in English.

Figure 2.3 shows the frequency distribution of the 26 most common letters used in Latin-based alphabets, covering all languages that use the Latin alphabet, such as Spanish, English, French, German, and Italian. It can be observed that the frequency of letters varies among different languages, which influences the optimal design of keyboard layouts for each language, even though these languages share a similar alphabet and common origin.

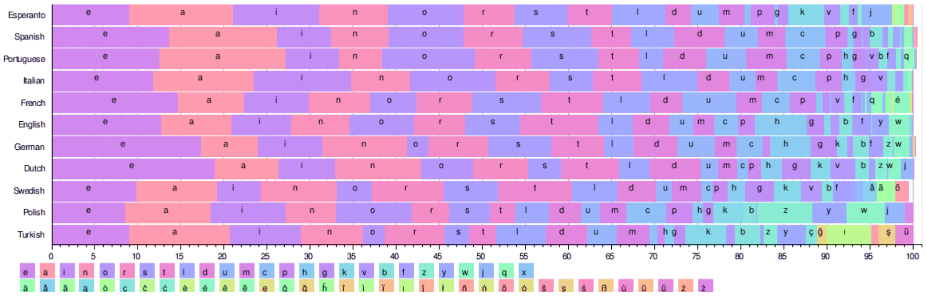


Figure 2.3: Frequency of letters in different languages.

As observed in Figure 2.3, the frequency varies across different languages. It is also important to note that our reference corpus must capture a diverse range of language use, including formal and informal texts, to obtain an accurate representation of the frequency of letters and letter combinations.

2.4 Used datasets

In order to analyse the different types of texts and their characteristics, this works uses different types of books or texts as their testing datasets. For a more classical literature approach, we have used *Moby Dick* by Herman Melville, a classic work of English literature. This novel provides a rich and varied vocabulary, making it suitable for analyzing letter and bigram frequencies in traditional literary texts.

Representing a simpler style, we use *The Wonderful Wizard of Oz* by L. Frank Baum. This book features a more constrained and common vocabulary, simpler sentence structures, and repetitive phrasing typical of children's literature. A layout optimized for this corpus should reflect the patterns of more common, everyday English usage.

For each corpus, the raw text will be pre-processed to normalize the data: all letters will be converted to lowercase, and all punctuation except the used in the algorithms, numbers, and whitespace will be removed. From the resulting continuous stream of characters, we will compute the fitness of each individual in order to determine how good they are as a keyboard layout.

Chapter 3

Genetic Algorithm Methodology

This section describes the methodology used to implement a genetic algorithm for optimizing keyboard layouts.

3.1 Problem Modeling

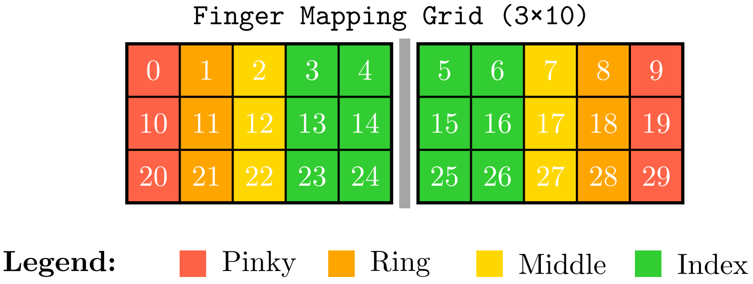
The keyboard layout optimization problem can be modeled as a combinatorial optimization problem where the goal is to find the best arrangement of keys on a keyboard. In our particular case, we will focus on the 26 letters of the English alphabet and a few punctuation marks. Trying to minimize the fitness function later introduced in section **??**. Each keyboard layout can be represented as a permutation of the set of characters to be arranged on the keyboard. Our considered set of letters and symbols includes the 26 letters of the English alphabet, along with the space character, period (.), comma (,), semicolon (:) and apostrophe (').

To model our individuals, we use a list of characters where each position corresponds to a specific key on the keyboard. The keyboard is divided into three rows and ten columns, giving us a total of 30 positions that will be filled with the characters from our set. Each key can only be occupied by one character, and each character can only appear once in the layout, ensuring that each individual represents a valid permutation.

The keyboard layout is represented as a standard 3 X 10 grid structure:

```
0,  1,  2,  3,  4,  5,  6,  7,  8,  9
10, 11, 12, 13, 14, 15, 16, 17, 18, 19
20, 21, 22, 23, 24, 25, 26, 27, 28, 29
```

Each position is assigned to a specific finger based on standard touch typing conventions, with finger assignments considering hand (left/right), finger type (pinky, ring, middle, index), and relative strength values.



Finger Mapping Grid (3×10)

🟥 0	🟧 1	🟨 2	🟩 3	🟩 4	🟩 5	🟩 6	🟨 7	🟧 8	🟥 9
🟥 10	🟧 11	🟨 12	🟩 13	🟩 14	🟩 15	🟩 16	🟨 17	🟧 18	🟥 19
🟥 20	🟧 21	🟨 22	🟩 23	🟩 24	🟩 25	🟩 26	🟨 27	🟧 28	🟥 29

Legend: 🟥 Pinky 🟧 Ring 🟨 Middle 🟩 Index

Figure 3.1: Finger mapping visualization showing key-to-finger assignments for left and right hands in a 3×10 grid.

3.2 Initial Population

The initial population is created by combining well knowon existing layouts with randomly generated layouts, providing the genetetic a good starting point for exploration. The `init_population` function generates a specified number of individuals, each representing a different keyboard layout, using the following approach: When the `known_distributions` parameter is set to True, the algorithm includes four well-established keyboard layouts as part of the initial population:

- `QWERTY`: The most common layout, designed originally for typewriters
- `Dvorak`: Scientifically designed layout focused on efficiency and reduced finger movement
- `QWERTZ`: German variant of QWERTY with Y and Z positions swapped
- `Colemak`: Modern layout designed to improve upon QWERTY while

maintaining some key positions

The remaining individuals in the population are generated randomly to ensure diversity. This individuals are created by shuffling the list of characters to be arranged on the keyboard, ensuring that each layout is a valid permutation of the character set. This hybrid approach of combining known layouts with random enables the genetic algorithm to start with some good solutions while also exploring a wide range of possibilities.

Each individual in the population undergoes validation to ensure it contains exactly 30 characters with no duplicates or missing characters from the defined character set.

3.3 Fitness Function

The fitness function evaluates the quality of a keyboard layout by calculating the total "typing cost" when typing a given text. Lower fitness scores indicate better layouts. The function considers multiple factors that affect typing efficiency and ergonomics.

For each consecutive pair of characters in the input text, the algorithm calculates:

1. `Euclidean distance:` the geometric distance between key positions on the 3×10 grid

2. `Finger penalty:` additional costs based on ergonomic factors

The `finger_penalty` function implements a comprehensive penalty system based on:

- `Same Finger Usage:` heavy penalty (1.0) when consecutive characters require the same finger, with additional penalty (2.0) for weak fingers (pinky and ring finger).

- `Same Hand Usage:` moderate penalty (1.0) when both characters are typed with the same hand, promoting hand alternation.

Also additional penalties for the vertical movements:

- `Adjacents rows (distance 1):` penalty of 0.2, with additional 0.15

for weak fingers

- `Two-row jumps (distance 2):` Tpenalty of 0.8, with additional 0.5 for weak fingers

Regarding the usage of the fingers, the weaker fingers are penalised, based on:

- `Pinky finger usage:` +0.15 penalty
- `Ring finger usage:` +0.1 penalty

Finally an additional penalisation for the movements between the same column. Giving an additional penalty of 0.3 for vertical movement within the same column, with extra penalties for outer columns (0.2 for columns 0 and 9, 0.1 for columns 1 and 8).

The cost for each character pair (c_i, c_{i+1}) is computed as:

$$\text{cost}(c_i, c_{i+1}) = d_{\text{euclidean}}(c_i, c_{i+1}) \times \max(1.0 + P_{\text{finger}}(c_i, c_{i+1}), 0.1) \tag{3.1}$$

where $d_{\text{euclidean}}$ represents the Euclidean distance between key positions and P_{finger} represents the cumulative finger penalty. The total fitness function is then:

$$F(\mathcal{L}) = \sum_{i=1}^{n-1} \text{cost}(c_i, c_{i+1}) \tag{3.2}$$

where $\mathcal{L}$ denotes the keyboard layout, n is the length of the input text, and the objective is to minimize $F(\mathcal{L})$.

3.4 New Population Generation

3.4.1 Selection

The algorithm implements tournament selection as the primary selection mechanism. The `tournament_selection` function works as follows:

1. Groups of k elements are established (where k is the size group). In this particular case having k=5 in order to balance between good and bad fitness individuals.

2. From each group we select the best individual (the one with the lowest fitness function).

3. We perform successive tournaments to select both parments for the crossover operation.

Using the tournament selection over other selection methods we ensure a good balance between selection pressure and diversity maintenance, allowing both high-quality individuals and some lower-quality individuals to participate in reproduction.

3.4.2 Crossover

To generate new individuals from two parents, the algorithm employs a **two-point crossover** operator specifically adapted for permutations. This method ensures that every child produced is a valid keyboard layout, containing all required characters exactly once. The process is executed as follows:

1. `Segment Selection:` two random crossover points are selected along the length of the parent layouts. These points define a central segment.

2. `Direct Inheritance:` the child layout inherits this central segment directly from the first parent and the rest from the second.

3. `Completion:` fill remaining positions using `parent2`, applying the mapping to resolve conflicts and ensure valid permutation, based on a reconstruction function.

The crossover operation is applied with a probability of `crossover_rate` (set to 0.85). If crossover is not performed, one of the parents is passed directly to the next generation, preserving its genetic material without alteration.

A `fix_duplicates` function ensures that any potential duplicates are resolved by replacing them with missing characters, maintaining the permutation property.

3.4.3 Mutation

The mutation operator implements a simple swap mutation strategy:

1. `Mutation detection:` each individual has a probability mutation_rate = 0.15 of being mutated.

2. `Swap Operation:` if mutation occurs, randomly select two positions and swap their characters.

3. `Permutation Preservation:` the swap operation maintains the valid permutation property

This mutation strategy provides local search capability while ensuring that all individuals remain valid keyboard layouts.

3.4.4 Replacement

The algorithm uses an elitist replacement strategy that combines the best individuals from the current generation with newly generated offspring:

1. `Elite preservation:` keep the top 50% of the individuals, particularly elite ones.

2. `Offspring integration:` fill the reamining populaton slots with the offspring from corssover and mutation operations.

This approach ensures that good solutions are not lost while allowing new genetic material to enter the population through reproduction operations. And maintaining a new population the same size as the previous generation.

3.5 Experimental setup

To evaluate the performance of the genetic algorithm and understand the characteristics of optimized keyboard layouts, a series of experiments were designed with varying conditions and corpora.

The baseline configuration used for all experiments is presented in Table 3.1. These parameters were selected based on preliminary experiments and computational constraints.

Table 3.1: Genetic Algorithm Hyperparameters

Parameter	**Value**	**Purpose**
Population Size	10,000	High diversity
Generations	100	Convergence
Elite Size	1,500 (15%)	Preserve best solutions
Tournament Size	5	Balance pressure/diversity
Crossover Rate	0.85	Explore combinations
Mutation Rate	0.15	Maintain diversity
Initial Population	Random	Unbiased search
Random seed	123	In order to recreate the experiments

3.6 Results and Analysis

3.6.1 Moby Dick Results

The genetic algorithm was executed on the Moby Dick book with the Hyperparameters specified in the Table 3.1. This classic literary work presents a rich and varied vocabulary, providing a robust test case for keyboard optimization.

Figure 3.2 illustrates the evolutionary progression of the algorithm over 100 generations. The fitness function decreased from an initial value of **1,375,854.6** to a final optimized value of **799,038.6**, representing a 41.9% improvement through evolution alone.

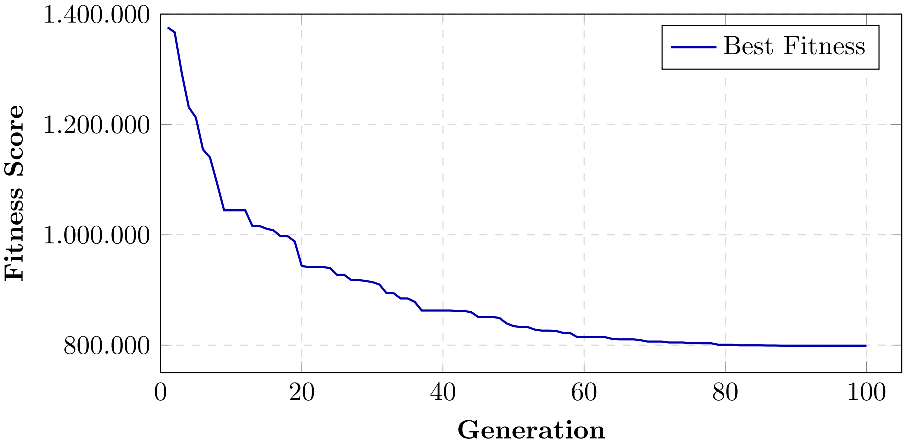


Figure 3.2: Genetic algorithm over 100 generations on Moby Dick corpus

The convergence pattern exhibits three distinct phases, as detailed in Table 3.2. The initial exploration phase (generations 1-20) shows rapid fitness improvement of 31.4%, finding new solutions and improving drastically, during which the algorithm discovers promising keyboard configurations through aggressive exploration of the search space. The exploitation phase (generations 21-60) demonstrates more gradual refinement with 13.6% improvement as the population converges toward local optima. Finally, the convergence phase (generations 61-100) shows minimal change (1.9% improvement), indicating the algorithm has reached a stable, high-quality solution.

Table 3.2: Phhase analysis (Moby dick)

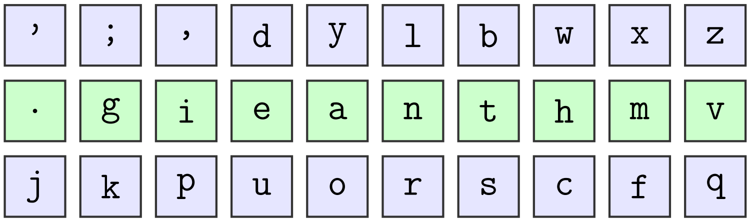
Phase	**Generations**	**Initial Fit.**	**Final Fit.**	**Improvement**
Exploration	1–20	1,375,854.6	943,321.0	31.4%
Exploitation	21–60	943,321.0	814,739.5	13.6%
Convergence	61–100	814,739.5	799,038.6	1.9%
Total	**1–100**	**1,375,854.6**	**799,038.6**	**41.9%**

Table 3.3 presents the fitness evolatuion per decade, showing the descents towards the local optimal solution found.

Table 3.3: Convergence milestones: Best fitness every 10 generations

Generation	**Best Fitness**	**Generation**	**Best Fitness**
1	1,375,854.6	60	814,739.5
10	1,044,380.1	70	806,493.9
20	943,321.0	80	800,763.6
30	914,313.3	90	799,038.6
40	862,777.3	100	799,038.6
50	834,521.5		

The result obtaied from this layout represented in Figure 3.3, shows that the algorithm has distributed characters across the three rows to minimize typing cost according to the fitness function defined in Section 3.3.



’	;	,	d	y	l	b	w	x	z
.	g	i	e	a	n	t	h	m	v
j	k	p	u	o	r	s	c	f	q

Home row (green) contains most frequent letters

Figure 3.3: Optimized keyboard layout evolved by genetic algorithm (Moby Dick book)

Finally, Table 3.4 categorizes the character distribution across rows, revealing the algorithm’s strategy for minimizing finger movement and maximizing typing efficiency. This distribution aligns with distribution shown in Figure 2.2, which represents the distribution of the keys. We can see that the most frequent keys have been placed in the center row or as centered as possible.

Table 3.4: Letter placement analysis of evolved layout

Row	**Letters**	**Notes**
Top	’, ;, ,, d, y, l, b, w, x, z	Punctuation and less frequent consonants
Home	., g, i, e, a, n, t, h, m, v	High-frequency vowels (e, a, i) and consonants (t, n, h)
Bottom	j, k, p, u, o, r, s, c, f, q	Mix of vowels (u, o) and common consonants (r, s)

To evaluate the effectiveness of the genetic algorithm, the evolved layout was compared against four established keyboard designs: QWERTY, QWERTZ, Dvorak, and Colemak. All layouts were evaluated using the same fitness function and Moby Dick book to ensure fair comparison. Table 3.5 presents the fitness scores and percentage improvements of the evolved layout over each known layout.

Table 3.5: Fitness comparison of keyboard layouts on Moby Dick book

Layout	**Fitness Score**	**Improvement of Evolved GA**
QWERTY	1,821,199.8	56.1%
QWERTZ	1,877,982.0	57.5%
Dvorak	1,433,882.9	44.3%
Colemak	1,368,940.0	41.6%
Evolved GA	**799,038.6**	–

3.6.2 Wizard of Oz Results

The genetic algorithm was also applied to The Wizard of Oz book, a children’s literature classic by L. Frank Baum. This text features simpler vocabulary, more repetitive phrasing, and shorter sentence structures compared to Moby Dick.

Figure 3.4 illustrates the evolutionary progression over 100 generations. The fitness decreased from an initial value of **214,378.7** to a final optimized value of **129,234.4**, representing a 39.7% improvement through evolution.

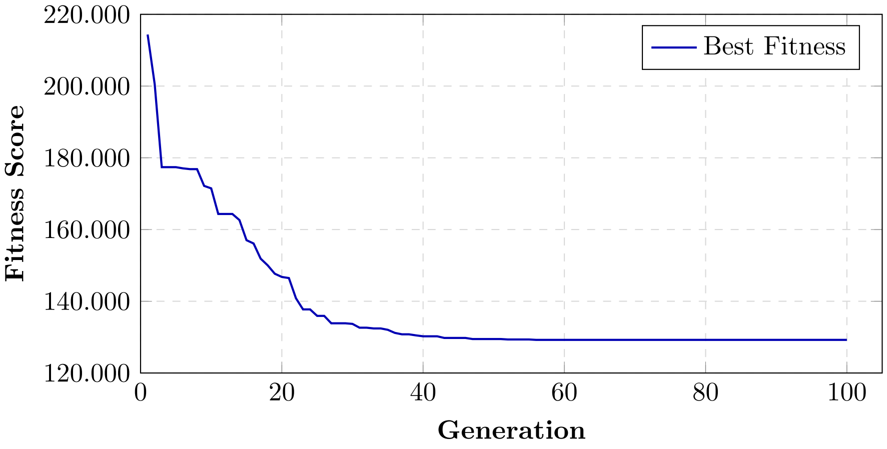


Figure 3.4: Genetic algorithm over 100 generations on Wizard of Oz book

Compared to Moby Dick, the Wizard of Oz optimization exhibits faster convergence. Table 3.6 shows that the algorithm reaches near-optimal solutions by generation 56, after which no further improvements occur for 44 consecutive generations.

Table 3.6: Phase analysis (Wizard of Oz)

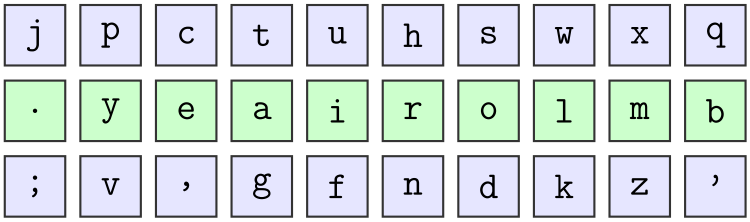
Phase	**Generations**	**Initial Fit.**	**Final Fit.**	**Improvement**
Exploration	1–20	214,378.7	146,779.8	31.5%
Explotation	21–56	146,779.8	129,234.4	12.0%
Convergence	57–100	129,234.4	129,234.4	0.0%
Total	**1–100**	**214,378.7**	**129,234.4**	**39.7%**

This earlier convergence suggest that simpler texts result in a less complex optimization with fewer local optimals. Table 3.7 presents the fitness score at every 10 generations.

Table 3.7: Convergence milestones: Best fitness every 10 generations (Wizard of Oz)

Generation	Best Fitness	Generation	Best Fitness
1	214,378.7	60	129,234.4
10	171,478.2	70	129,234.4
20	146,779.8	80	129,234.4
30	133,691.4	90	129,234.4
40	130,231.1	100	129,234.4
50	129,467.7		

The evolved layout for the Wizard of Oz corpus is presented in Figure 3.5. Notably, this layout differs substantially from the Moby Dick optimization, reflecting the different linguistic characteristics of children's literature. But keeps the same core strategy. It places the vowels at the center row, with frequent consonats, optimizing the high frequency letters, reducing their fitness cost. And placing the less frequent letters and punctuation across the top and bottom rows to reduce finger movement.



j	p	c	t	u	h	s	w	x	q
.	y	e	a	i	r	o	l	m	b
;	v	,	g	f	n	d	k	z	’

Home row (green) optimized for children's literature patterns

Figure 3.5: Optimized keyboard layout evolved by genetic algorithm (Wizard of Oz corpus)

Table 3.8 categorizes the distribution of characters across the three rows. The home row contains the majority of vowels and high-frequency consonants, which minimizes finger movement for common sequences. The top and bottom rows are reserved for punctuation and less frequent letters, reflecting the algorithm's strategic placement for efficient typing.

Table 3.8: Letter placement analysis of evolved layout (Wizard of Oz)

Row	**Characters**	**Characteristics**
Top Row	j, p, c, t, u, h, s, w, x, q	Mix of common (t, h, s) and rare consonants
Home Row	., y, e, a, i, r, o, l, m, b	All five vowels plus frequent consonants
Bottom Row	;, v, ,, g, f, n, d, k, z, ’	Punctuation and moderate-frequency consonants

Finally, Table 3.9 compares the fitness scores of the evolved layout with existing layouts. The evolved layout significantly outperforms QWERTY, QWERTZ, Dvorak, and Colemak, demonstrating the effectiveness of the genetic algorithm in adapting to the unique characteristics of the text.

Table 3.9: Fitness comparison of keyboard layouts on Wizard of Oz corpus

Layout	**Fitness Score**	**Improvement of Evolved GA**
QWERTY	273,549.7	52.8%
QWERTZ	286,167.2	54.8%
Dvorak	230,728.3	44.0%
Colemak	222,474.5	41.9%
Evolved GA	**129,234.4**	–

The improvement percentages are remarkably consistent with the Moby Dick results, with only minor variations (±1-2%). This consistency suggests that our fitness function captures the typing patterns and is capable of representing the improvement in the final individual.

3.6.3 Cross comparison of Layouts

To evaluate the generalization capability of the optimized layouts, a cross-evaluation was performed. The layout evolved with the *Moby Dick* corpus was tested on *The Wizard of Oz* text, and vice versa. The objective is to determine whether a layout optimized for a text with complex linguistic features (rich and varied vocabulary) is more robust than one optimized for a simpler, more repetitive text.

The results of this comparison are presented in Table 3.10.

Table 3.10: Cross-comparison of fitness scores for the evolved layouts.

Test Corpus	**Evolved Layout**	**Fitness Score**	**Notes**
Moby Dick	Moby Dick	**799,038.6**	*Baseline (specialized)*
	Wizard of Oz	848,748.9	Performance of non-specialized layout
Wizard of Oz	Wizard of Oz	**129,234.4**	*Baseline (specialized)*
	Moby Dick	132,224.5	Performance of non-specialized layout

The results in Table 3.10 show us how the layouts and the algorithm behaved:

- `Confirmed Specialization:` Each layout performed best on its own book. This demonstrates effective **exploitation**, where the algorithm perfected the best-found layouts by fine-tuning them to the specific letter patterns of each text.

- `Over-specialization of the Simple-Text Layout:` The *Wizard of Oz* layout struggled on the complex text, with a 6.2% performance drop. The simple book offered a limited field for **exploration**, leading the algorithm to over-specialize on a narrow solution that lacked flexibility.

- `Robustness of the Complex-Text Layout:` Conversely, the *Moby Dick* layout was very robust, dropping only 2.3% on the simpler text. The book's rich language provided a better landscape for **exploration**, allowing the algorithm to discover a more versatile solution before the final **exploitation** phase, resulting in a more robust layout.

Chapter 4

Simulated Annealing Methodology

Chapter 5

Conclusions

5.1 Summary of the developed work

5.2 Achieved objectives

5.3 Future work