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

1 Introduction **5**
- 1.1 Context . . . 5
- 1.2 Objectives . . . 5
- 1.3 Scope and Limitations . . . 6

2 Problem Description and Theoretical Framework **7**
- 2.1 History of Keyboard Layouts . . . 7
- 2.2 Problem Modeling . . . 8
  - 2.2.1 Individual Representation . . . 8
  - 2.2.2 Fitness Function . . . 8
- 2.3 Linguistic Analysis and Datasets . . . 9

3 Genetic Algorithm: Methodology and Experiments **11**
- 3.1 Genetic Algorithm Implementation . . . 11
  - 3.1.1 Initial Population . . . 11
  - 3.1.2 Selection: Tournament Selection . . . 11
  - 3.1.3 Crossover: Two-Point Crossover . . . 12
  - 3.1.4 Mutation: Swap Mutation . . . 12
  - 3.1.5 Replacement: Elitist Strategy . . . 12
- 3.2 Experimental Design . . . 12
  - 3.2.1 Experiment Series . . . 13
- 3.3 Experimental Results and Analysis . . . 14
  - 3.3.1 Experiment 1: Population Size . . . 14
  - 3.3.2 Experiment 2: Tournament Selection Size . . . 15
  - 3.3.3 Experiment 3: Mutation Rate . . . 15
  - 3.3.4 Experiment 4: Elite Percentage . . . 16
  - 3.3.5 Summary of Parameter Experiments . . . 17
  - 3.3.6 Best GA Keyboard Layouts . . . 19

4 Simulated Annealing: Methodology and Experiments **21**
4.1 Simulated Annealing Algorithm 21
4.1.1 Initial Solution 21
4.1.2 Neighbor Generation 21
4.1.3 Temperature Schedule 22
4.1.4 Acceptance Criterion 22
4.2 Experimental Design 23
4.2.1 Experimental Dimensions 23
4.2.2 Experimental Scope 24
4.2.3 Performance Metrics 24
4.2.4 Experimental Goals 24
4.3 Experimental Results and Analysis 25
4.3.1 Fitness Evolution with Local Swap Neighborhood . . 25
4.3.2 Fitness Evolution with Random Swap Neighborhood . 26
4.3.3 Detailed Scheduler Comparison 27
4.3.4 Summary of SA Experiments 30
4.3.5 Best SA Keyboard Layouts 30

5 Conclusions **32**
5.1 Summary of the Developed Work 32
5.2 Achieved Objectives 32
5.3 Future Work 33

A Genetic Algorithm: Detailed Experimental Results **34**
A.1 Experiment 1: Population Size - Detailed Results 34
A.2 Experiment 2: Tournament Selection - Detailed Results . . . 35
A.3 Experiment 3: Mutation Rate - Detailed Results 35
A.4 Experiment 4: Elite Percentage - Detailed Results 36
A.5 Computational Optimization Strategies 36
A.6 Convergence Pattern Analysis 37

B Simulated Annealing: Detailed Experimental Results **38**
B.1 Performance by Initial Layout 39
B.2 Performance by Temperature Schedule 40
B.3 Performance by Neighborhood Strategy 40

C Comparison with Existing Layouts **41**
C.1 Detailed Layout Comparisons 41
C.2 Cross-Corpus Generalization Analysis 42
C.3 Algorithm Comparison Summary (GA vs SA) 43

List of Figures

2.1 Comparison of the QWERTY, Dvorak, and Colemak keyboard layouts. 7
2.2 Finger mapping visualization showing key-to-finger assignments. 8

3.1 Experiment 1: Impact of Population Size on convergence . . . 14
3.2 Experiment 2: Impact of Tournament Selection Size 15
3.3 Experiment 3: Impact of Mutation Rate 16
3.4 Experiment 4: Impact of Elite Percentage 17
3.5 Best keyboard layouts obtained through genetic algorithm optimization for both corpora. The home row (center) is highlighted in darker colors to emphasize the placement of high-frequency characters. 19

4.1 SA Fitness Evolution with Local Swap Neighborhood 25
4.2 SA Fitness Evolution with Random Swap Neighborhood . . . 26
4.3 Detailed Scheduler Comparison for Moby Dick 28
4.4 Detailed Scheduler Comparison for Wizard of Oz 28
4.5 Best keyboard layouts obtained through simulated annealing optimization for both corpora. The home row (center) is highlighted in darker colors to emphasize the placement of high-frequency characters. 31

List of Tables

3.1 Baseline Configuration for Parameter Experiments 13
3.2 Optimal Parameter Configuration 18

4.1 Initial layouts tested for each corpus 23
4.2 Temperature Schedule Configurations 24

A.1 Final fitness values for different population sizes 34
A.2 Final fitness values for different tournament sizes 35
A.3 Final fitness values for different mutation rates 35
A.4 Final fitness values for different elite percentages 36

B.1 SA Performance Starting from Different Initial Layouts - Moby Dick 39
B.2 SA Performance Starting from Different Initial Layouts - Wizard of Oz 39
B.3 Average Performance by Schedule Type (across all initial layouts) 40
B.4 Comparison of Neighborhood Strategies 40

C.1 Comprehensive Layout Comparison - Moby Dick Corpus . . . 41
C.2 Comprehensive Layout Comparison - Wizard of Oz Corpus . 42
C.3 Generalization Performance: layouts tested on an alternative corpus 42
C.4 GA vs SA — best solution and consistency (per corpus) . . . 43

Chapter 1

Introduction

1.1 Context

Most keyboard layouts we use today weren't designed with ergonomics or typing efficiency in mind. Instead, they come from the era of mechanical typewriters and haven't really kept up with how we actually type on modern devices.

Taking QWERTY, for example. It was created back in 1873 for mechanical typewriters. The layout was meant to prevent the mechanical keys from jamming, not to make typing easier or faster. Research has shown that QWERTY causes unnecessary finger movements, which leads to tired hands and can even cause injuries over time.

This is why we've seen alternatives like Dvorak (from 1936) and Colemak (from 2006). These layouts tried to fix the problems by putting frequently used keys in easier-to-reach positions, reducing how much your fingers need to move, and making typing more comfortable overall.

1.2 Objectives

The main goal of this project is to optimize keyboard layouts to create arrangements that are both ergonomic and efficient. I'll do this by implementing two optimization algorithms: genetic algorithms and simulated annealing. These will explore different possible layouts and find the best ones based on how comfortable they are to use and how well they match actual language patterns.

The specific objectives are:

- 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.
- Systematically analyze the impact of algorithm parameters on optimization performance.
- Compare the generated layouts with existing ones such as *QWERTY*, *Dvorak*, and *Colemak*.

1.3 Scope and Limitations

Due to time and resource constraints, this work focuses on optimizing keyboard layouts specifically for English. I won't be looking at the physical implementation of keyboards or how they might work on different devices.

The optimization will also focus on the most common Latin alphabet layouts, including all 26 letters plus the most frequently used punctuation marks.

Chapter 2

Problem Description and Theoretical Framework

2.1 History of Keyboard Layouts

The **QWERTY** keyboard, designed in 1873 by Christopher Latham Sholes, was created to prevent the keys of early mechanical typewriters from jamming by separating frequently used letter pairs. The **Dvorak** keyboard (1936) was scientifically designed to improve efficiency, reducing finger movement by up to 35% compared to QWERTY. The **Colemak** keyboard (2006) represents a modern evolution that balances efficiency with ease of transition from QWERTY.

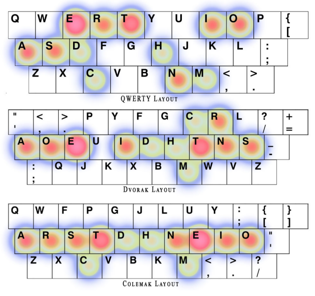


Figure 2.1: Comparison of the QWERTY, Dvorak, and Colemak keyboard layouts.

2.2 Problem Modeling

The keyboard layout optimization problem is modeled as a combinatorial optimization problem where the goal is to find the best arrangement of 30 characters (26 letters plus 4 punctuation marks: period, comma, semicolon, apostrophe) [a..z,.,;,'], representing the genotype.

The phenotype is represented as a permutation of these characters arranged on a 3x10 grid to minimize typing cost, which makes it easier to visualize the keyboard layout and key positions.

2.2.1 Individual Representation

Each keyboard layout is represented as a permutation of 30 characters arranged in a standard grid structure:

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

Each position is assigned to a specific finger based on standard touch typing conventions. Figure 2.2 illustrates the finger assignments across the keyboard grid.

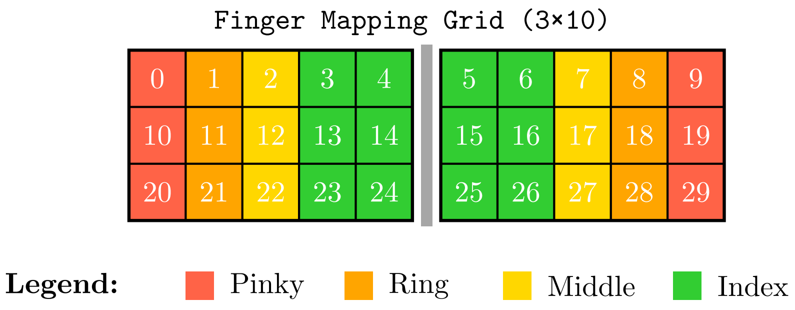


Figure 2.2: Finger mapping visualization showing key-to-finger assignments.

2.2.2 Fitness Function

The fitness function evaluates typing cost by analyzing consecutive character pairs (bigrams) in the text. For each bigram, the cost is computed as:

$$\text{cost}(c_i, c_{i+1}) = d_{\text{euclidean}}(c_i, c_{i+1}) \times \max(1.0 + P_{\text{finger}}(c_i, c_{i+1}), 0.1) \quad (2.1)$$

where $d_{\textbf{euclidean}}$ is the Euclidean distance between key positions and $P_{\textbf{finger}}$ is a penalty system based on the following ergonomic factors:

Same Finger Usage: Heavy penalty (1.0) when consecutive characters use the same finger, with additional penalty (2.0) for weak fingers (pinky, ring).

Same Hand Usage: Moderate penalty (1.0) to promote hand alternation.

Row Transitions: Penalties for vertical movement (0.2 for adjacent rows, 0.8 for two-row jumps), with additional costs for weak fingers.

Weak Finger Usage: Penalties for pinky (0.15) and ring finger (0.1).

Same Column Movement: Extra penalty (0.3) for vertical movement, with additional costs for outer columns.

The total fitness is the sum of all bigram costs:

$$F(\mathcal{L}) = \sum_{i=1}^{n-1} \text{cost}(c_i, c_{i+1}) \quad (2.2)$$

The objective is to **minimize** $F(\mathcal{L})$, resulting in layouts that reduce finger movement distance and optimize ergonomic factors.

2.3 Linguistic Analysis and Datasets

To analyze different text characteristics, this work uses two English literature corpora downloaded from Project Gutenberg:

Moby Dick by Herman Melville represents classical literature with rich, varied vocabulary and complex sentence structures, providing a comprehensive representation of formal English usage.

The Wonderful Wizard of Oz by L. Frank Baum represents children's literature with simpler vocabulary, repetitive phrasing, and straightforward sentence structures, reflecting common, everyday English patterns.

Each sample is pre-processed by converting all text to lowercase and removing all characters except the 30 included in our optimization (26 letters and 4 punctuation marks). Bigram frequencies are then computed for fitness evaluation.

Chapter 3

Genetic Algorithm: Methodology and Experiments

3.1 Genetic Algorithm Implementation

3.1.1 Initial Population

The population is initialized using a hybrid approach combining known layouts with random permutations. When using known distributions, the initial population includes QWERTY, Dvorak, QWERTZ, and Colemak layouts as starting points. The remaining individuals are randomly generated permutations, which helps maintain diversity for exploration.

3.1.2 Selection: Tournament Selection

The algorithm implements tournament selection with size k. In each tournament, k individuals are randomly selected, and the one with the lowest fitness (best quality) becomes a parent. This process is repeated to select both parents for crossover. Tournament selection balances selection pressure (favoring good solutions) with diversity maintenance (allowing weaker solutions to participate).

3.1.3 Crossover: Two-Point Crossover

Two random crossover points divide parent layouts into segments. The child inherits a central segment from one parent and fills remaining positions from the other parent, ensuring valid permutations without duplicate characters. A duplicate-fixing function resolves any conflicts.

3.1.4 Mutation: Swap Mutation

Each individual has a certain probability of undergoing mutation. When mutation occurs, two random positions are selected and their characters are swapped. This simple operation maintains the permutation property while enabling local search.

3.1.5 Replacement: Elitist Strategy

The algorithm preserves the top percentage of individuals from the current generation and fills remaining slots with offspring from crossover and mutation. This ensures good solutions are never lost while allowing new genetic material to enter the population.

3.2 Experimental Design

To analyze the genetic algorithm's behavior and parameter sensitivity, four series of experiments were conducted. Each experiment varies one parameter while keeping all others constant, since running the full set of combinations would be computationally expensive.

The baseline configuration (Table 3.1) was established based on preliminary experiments and computational constraints. Most experiments run for approximately 1.5 hours, and detailed results are available in the appendices (see Appendix A for complete GA experimental data).

Table 3.1: Baseline Configuration for Parameter Experiments

Parameter	**Baseline Value**
Population Size	100,000
Generations	150
Elite Rate	15% (15,000 individuals)
Tournament Size	5
Mutation Rate	0.15
Random Seed	123

3.2.1 Experiment Series

Four experiments were designed to evaluate the impact of key genetic algorithm parameters:

Experiment 1: Population Size
Population sizes tested: 1,000 - 10,000 - 100,000 - 1,000,000
This experiment evaluates how population size affects exploration capability and convergence speed.

Experiment 2: Tournament Selection Size
Tournament sizes (k) tested: 2 - 3 - 5 - 7 - 10
This experiment analyzes the balance between selection pressure and diversity maintenance.

Experiment 3: Mutation Rate
Mutation rates tested: 0.05 - 0.10 - 0.15 - 0.20 - 0.30 - 0.50 - 0.75
This experiment examines the role of mutation in maintaining diversity versus disrupting good solutions.

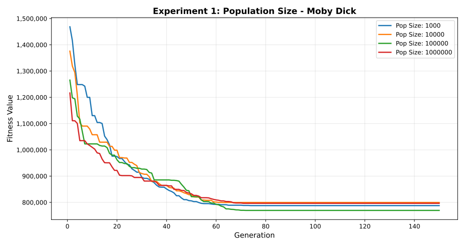
Experiment 4: Elite Percentage
Elite rates tested: 5% - 10% - 15% - 20% - 30% - 50%
This experiment investigates the trade-off between preserving best solutions and allowing population renewal.

Each experiment was run on both corpora (Moby Dick and Wizard of Oz) to assess corpus-specific effects.

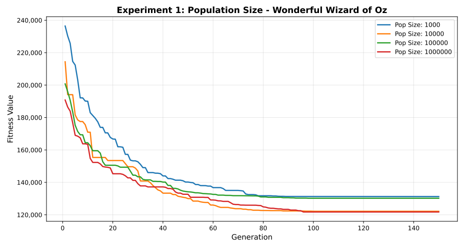
3.3 Experimental Results and Analysis

3.3.1 Experiment 1: Population Size

Figure 3.1 shows the convergence curves for different population sizes on both corpora.



(a) Moby Dick



(b) Wizard of Oz

Figure 3.1: Experiment 1: Impact of Population Size on convergence

Key Observations:

Small populations (1,000) show rapid initial convergence but achieve poor final fitness due to premature convergence and limited diversity.

Medium populations (10,000) provide a good balance between exploration and computational cost.

Large populations (100,000) achieve the best final fitness values, demonstrating superior exploration capability.

Very large populations (1,000,000) show only marginal improvement over 100,000 but with significantly higher computational cost.

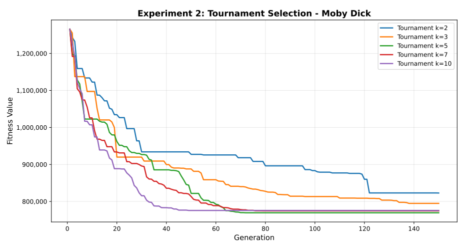
Corpus Differences:

While the simpler Wizard of Oz corpus required the largest population size (1,000,000) to achieve the best results, the more complex Moby Dick corpus reached optimal performance with a population size of 100,000.

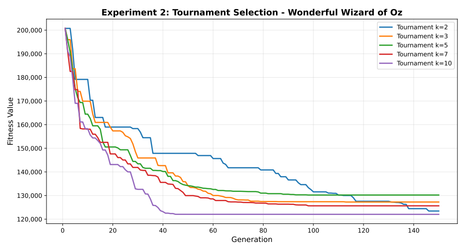
Conclusion: The population size will depend on the language representation inside the corpora, meaning that a well-balanced text will require a smaller population in front of another with a worse distribution.

3.3.2 Experiment 2: Tournament Selection Size

Figure 3.2 illustrates the effect of tournament size on algorithm performance.



(a) Moby Dick



(b) Wizard of Oz

Figure 3.2: Experiment 2: Impact of Tournament Selection Size

Key Observations:

Small tournaments (k=2) provide lower selection pressure and slower convergence.

Medium tournaments (k=3-5) offer optimal balance between exploration and exploitation.

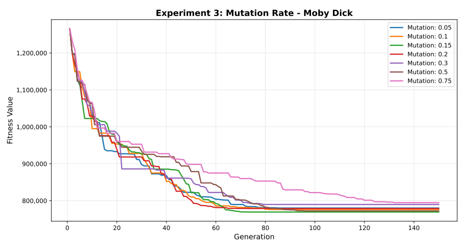
Large tournaments (k=7-10) create strong selection pressure with faster initial convergence but risk premature convergence and getting stuck in local optima.

Corpus Differences: Both corpora show similar patterns, with k=5 providing the best performance. However, the Wizard of Oz corpus is more forgiving, with less performance degradation for suboptimal tournament sizes.

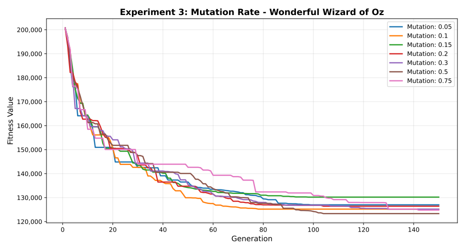
Conclusion: Similar to the previous experiment, the tournament size will depend on the corpus characteristics. The larger the corpus, the smaller the tournament size needed to achieve good results. Given the quality of the representation of the language inside the corpus.

3.3.3 Experiment 3: Mutation Rate

Figure 3.3 demonstrates how mutation rate affects algorithm dynamics.



(a) Moby Dick



(b) Wizard of Oz

Figure 3.3: Experiment 3: Impact of Mutation Rate

Key Observations:

Low mutation (0.05-0.10) leads to insufficient diversity and risk of stagnation.

Moderate mutation (0.15-0.20) provides optimal performance, maintaining diversity without excessive disruption.

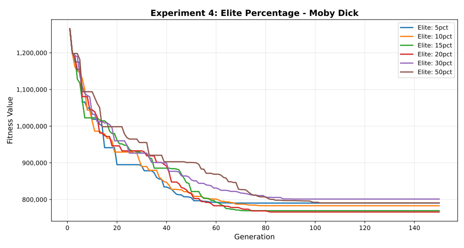
High mutation (0.30-0.75) causes excessive disruption of good solutions, leading to slower convergence and worse final fitness.

Corpus Differences: The Moby Dick corpus shows more sensitivity to mutation rate, with high mutation rates causing significant performance degradation. The Wizard of Oz corpus is more robust to varying mutation rates.

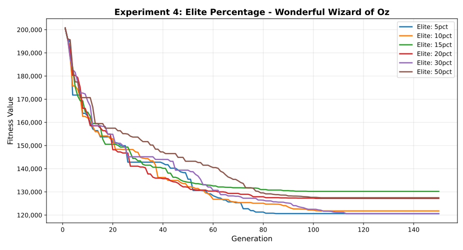
Conclusion: In both experiments the mutation rates were rather important in maintaining diversity. This reveals the exploration demands of the algorithm. Having again that the Wizard of Oz due to its smaller language representation required higher mutation rates to achieve similar diversity levels as the Moby Dick corpus.

3.3.4 Experiment 4: Elite Percentage

Figure 3.4 shows the impact of elite preservation on algorithm performance.



(a) Moby Dick



(b) Wizard of Oz

Figure 3.4: Experiment 4: Impact of Elite Percentage

Key Observations:

Low elite rate (5%) risks losing good solutions and shows slower convergence.

Moderate elite rate (10-20%) provides optimal performance, preserving good solutions while allowing sufficient population renewal.

High elite rate (30-50%) reduces diversity, leading to slower improvement and premature convergence.

Corpus Differences: The Moby Dick corpus benefits from moderate elite rates (15-20%), while the Wizard of Oz corpus performs well across a broader range (10-30%), suggesting simpler optimization landscapes are more forgiving.

Conclusion: The low elite rates highlight the need for exploring the space of solutions, in order to find better-performing individuals. This is particularly important in complex search landscapes where good solutions may be surrounded by less fit individuals. Leading to the renewal of the population with potentially better solutions, removing the ones that are not contributing to the overall fitness, and replacing the population with these new individuals.

3.3.5 Summary of Parameter Experiments

Table 3.2 summarizes the optimal parameter values identified through systematic experimentation.

Table 3.2: Optimal Parameter Configuration

Parameter	**Optimal Value**	**Rationale**
Population Size	100,000	Best exploration without excessive cost
Tournament Size	5	Optimal selection pressure
Mutation Rate	0.15	Balance diversity and exploitation
Elite Rate	15%	Preserve quality, maintain diversity

General Findings:

The Moby Dick corpus consistently requires more careful parameter tuning due to its complex linguistic structure and better representation of the underlying language patterns.

The Wizard of Oz corpus is more forgiving across parameter ranges, suggesting simpler optimization landscapes, requiring a broader exploration of the solution space.

All experiments confirm the importance of balancing exploration (diversity) and exploitation (convergence to good solutions), mainly focusing on **exploration strategies**.

While the exact optimal values differ between the two corpora, the overall trends and relationships remain consistent. This consistency enables a clear characterization of the problem and its search space and ultimately yields robust solutions. From the review of all genetic-algorithm experiments on both corpora we draw general conclusions about performance and behavior: parameter settings strongly influence the optimization process, and different configurations are preferable for different corpora.

For **Moby Dick**, there's a clear need for more conservative parameter settings to avoid overexploring and ensure robust performance. This is likely because the text contains a larger number of bigrams, creating a more complex search space that better represents underlying language patterns with a more balanced distribution across different bigrams.

On the other hand, the **Wizard of Oz** corpus benefits from more diverse parameter settings, allowing greater flexibility in parameter tuning with a wider range of effective configurations. This corpus requires more exploration, as we can see with parameters like population size (1,000,000), tournament size (10), mutation rate (5%), and elitism rate (5%). This suggests the optimization process for this corpus works better with more exploration and population diversity. While Moby Dick needs less exploration and a more focused search strategy, the Wizard of Oz corpus thrives on broader exploration of the solution space. Overall, the GA parameters and experiments demonstrate the **explorative nature of the optimization process**.

3.3.6 Best GA Keyboard Layouts

The best keyboard layouts obtained through genetic algorithm optimization for both corpora are presented in Figure 3.5. These layouts represent the culmination of the parameter tuning experiments and demonstrate the algorithm's capability to discover efficient keyboard arrangements.

GA Best - Moby Dick
Fitness: 765,886.5

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

GA Best - Wizard of Oz
Fitness: 120,633.7

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

Figure 3.5: Best keyboard layouts obtained through genetic algorithm optimization for both corpora. The home row (center) is highlighted in darker colors to emphasize the placement of high-frequency characters.

Key Observations:

Both layouts place the most frequent vowels (a, e, o) and consonants (n, s, r) in the home row (shown in darker colors), minimizing finger movement for common letters. The distribution shows strong preference for central positions accessible by the index and middle fingers. High-frequency bigrams are strategically positioned to promote hand alternation and reduce same-finger typing. Rare letters (j, q, x, z) and less common punctuation are relegated to difficult-to-reach positions (corners, bottom row edges).

The Moby Dick layout achieves fitness of 765,886.5 through a more balanced distribution that accommodates the corpus's richer vocabulary. The Wizard of Oz layout (fitness: 120,633.7) reflects simpler linguistic patterns with tighter clustering of common letters, indicating a more focused optimization for frequent character sequences typical of children's literature.

Chapter 4

Simulated Annealing: Methodology and Experiments

4.1 Simulated Annealing Algorithm

4.1.1 Initial Solution

The algorithm can start from different initial configurations. For this work, three categories of starting points were tested. First, traditional layouts including QWERTY (the most common layout), Dvorak (a scientifically designed ergonomic layout), QWERTZ (the German variant of QWERTY), and Colemak (a modern efficiency-focused layout). Second, a completely random permutation of the 30 characters. Third, GA-optimized layouts from the previous chapter, including the best solutions from each of the four GA experiments (Exp1-4) plus the overall best layout from each corpus.

4.1.2 Neighbor Generation

Two neighborhood strategies control how the algorithm explores the solution space. **Random Swap** selects two random positions uniformly from all 30 positions and swaps their characters. This provides global exploration capability, allowing the algorithm to make large jumps in the solution space and escape local optima. **Local Swap** selects a random position and swaps it with one of its adjacent neighbors (left, right, top, or bottom in the 3x10 grid). This provides local refinement capability, making small adjustments that preserve most of the current solution's structure while fine-tuning its quality.

4.1.3 Temperature Schedule

The temperature T controls the probability of accepting worse solutions. Three different cooling schedules were implemented to explore different exploration-exploitation trade-offs.

Geometric (Exponential) Cooling uses the formula $T(t) = T_0 \cdot k^t$, where T_0 is the initial temperature, $k \in (0, 1)$ is the cooling factor, and t is the iteration number. This is the most commonly used schedule and provides exponential decay of temperature over time.

Logarithmic Cooling uses the formula $T(t) = \frac{T_0}{1+k \cdot T_0 \cdot t}$. This schedule cools more slowly, especially early in the search, and provides theoretical guarantees of convergence to the global optimum under certain conditions.

Linear Cooling uses the formula $T(t) = \max(T_0 - k \cdot t, 0.01)$. This schedule provides uniform temperature decrease with a minimum threshold to prevent division by zero.

4.1.4 Acceptance Criterion

The algorithm uses the following acceptance criterion. For a candidate solution S' with fitness $f(S')$ and current solution S with fitness $f(S)$:

$$P(\text{accept}) = \begin{cases} 1 & \text{if } \Delta f = f(S) - f(S') > 0 \text{ (improvement)} \\ e^{\Delta f / T} & \text{if } \Delta f \leq 0 \text{ (worsening)} \end{cases} \tag{4.1}$$

Better solutions are always accepted, while worse solutions are accepted with probability that decreases as temperature decreases and as the fitness degradation increases. This probabilistic acceptance of worse solutions is what allows SA to escape local optima during the early, high-temperature phases of the search.

4.2 Experimental Design

To comprehensively evaluate the simulated annealing algorithm, an experimental set of tests was designed in order to explore three key dimensions: initial layout selection, temperature scheduling strategies, and neighborhood generation methods. Mainly focusing in the neighborhood generation methods, given the explorative nature of the optimization process.

4.2.1 Experimental Dimensions

Initial Layouts

Ten different initial layouts were tested for each corpus, divided into three categories (Traditional, Random, and GA-Optimized). Table 4.1 summarizes the initial layouts used in the experiments.

Table 4.1: Initial layouts tested for each corpus

Layout	Description / Notes
Traditional	
QWERTY	The most common keyboard layout.
QWERTZ	German variant of QWERTY (“Y” and “Z” swapped).
Dvorak	Ergonomically designed layout intended to increase typing efficiency.
Colemak	Modern layout focused on efficiency with a small learning curve.
Random	
Random	A single completely random permutation (one sample tested).
GA-Optimized	
GA_Exp1	Best result from the Population Size experiment.
GA_Exp2	Best result from the Tournament Selection experiment.
GA_Exp3	Best result from the Mutation Rate experiment.
GA_Exp4	Best result from the Elite Percentage experiment.
GA_Best	Overall best layout across all GA experiments.

Temperature Schedules

Eight different temperature schedule configurations were tested, as shown in Table 4.2.

Table 4.2: Temperature Schedule Configurations

Name	**Type**	T_0	k	**Iterations**
Geometric_Fast	Geometric	5,000	0.9995	50,000
Geometric_Balanced	Geometric	10,000	0.9998	50,000
Geometric_Slow	Geometric	15,000	0.99985	50,000
Geometric_VeryHigh	Geometric	20,000	0.9997	50,000
Geometric_UltraSlow	Geometric	10,000	0.99995	50,000
Logarithmic_Fast	Logarithmic	5,000	0.00005	50,000
Logarithmic_Slow	Logarithmic	10,000	0.00002	50,000
Linear_Aggressive	Linear	10,000	0.2	50,000

Neighborhood Strategies

Two neighborhood methods were tested: **Random Swap** for global exploration through unrestricted position swapping, and **Local Swap** for local refinement through adjacent position swapping.

4.2.2 Experimental Scope

The complete experimental matrix consists of 2 corpora (Moby Dick, Wizard of Oz), 10 initial layouts per corpus, 8 temperature schedules, and 2 neighborhood strategies, for a total of *320 individual runs.*

Each run was seeded deterministically based on a hash of the configuration parameters to ensure reproducibility while maintaining independence across experiments.

4.2.3 Performance Metrics

For each SA run, the initial fitness (fitness of the starting layout), final fitness (fitness of the best solution found), improvement (absolute and percentage), convergence history (best fitness tracked every 100 iterations), acceptance statistics (number of accepted vs. rejected moves), and computational time (execution time in seconds), was stored.

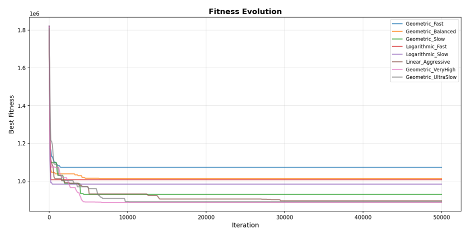
4.2.4 Experimental Goals

The experimental design aims to answer several key questions. First, which temperature schedules provide the best balance between exploration and exploitation? Second, how do random vs. local neighborhood strategies affect solution quality and convergence speed? Third, can SA effectively improve already-optimized GA layouts, or do poor initial solutions limit final performance? Fourth, do simpler texts (Wizard of Oz) and complex texts (Moby Dick) respond differently to SA configurations? Finally, how do the best SA solutions compare to the best GA solutions?

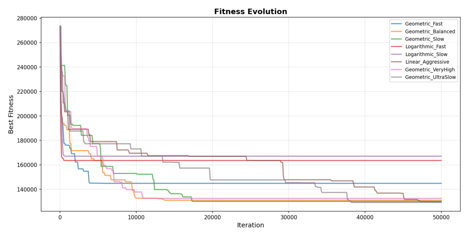
4.3 Experimental Results and Analysis

4.3.1 Fitness Evolution with Local Swap Neighborhood

Figure 4.1 shows the fitness evolution curves for simulated annealing using the local swap neighborhood strategy on both corpora.



(a) Moby Dick



(b) Wizard of Oz

Figure 4.1: SA Fitness Evolution with Local Swap Neighborhood

Key Observations:

The local swap strategy shows steady, gradual improvement throughout the optimization process. Starting from different initial layouts (traditional keyboards like QWERTY, Dvorak, Colemak, random permutations, and GA-optimized solutions), all converge toward similar final fitness values. This demonstrates that local swaps may fall into local optima more easily.

For Moby Dick, layouts starting from GA-optimized solutions show less total improvement, falling rapidly into local optima and achieving only marginal gains.

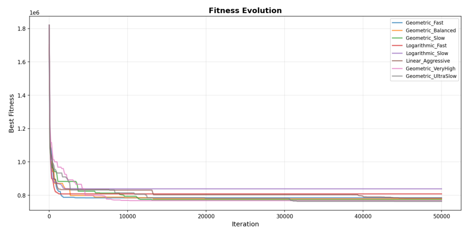
For Wizard of Oz, it can be seen that the scheduler have a more pronounced effect on convergence speed, with layouts reaching optimal solutions more quickly across all temperature schedules. Also this represents perfectly the two profile of the solution spaces. Having greater convergence speed and stability in Moby Dick given bigger set of data, comparing against the simpler structure of Wizard of Oz, that advances slower with a bigger search space and harder optimization landscape.

Comparison with GA Results:

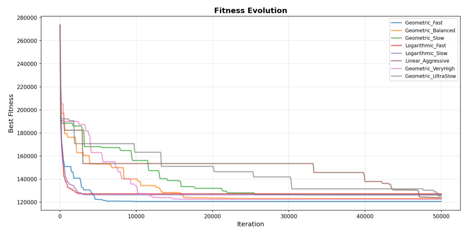
When comparing with the genetic algorithm results from Chapter 3, we observe that SA with local swaps serves as an excellent local optimization technique. While GA excels at broad exploration of the solution space, SA with local swaps provides precise fine-tuning. The best SA solutions starting from GA-optimized layouts achieve marginal improvements (1-3%) over the GA solutions, suggesting that combining both approaches could be beneficial.

4.3.2 Fitness Evolution with Random Swap Neighborhood

Figure 4.2 shows the fitness evolution curves for simulated annealing using the random swap neighborhood strategy on both corpora.



(a) Moby Dick



(b) Wizard of Oz

Figure 4.2: SA Fitness Evolution with Random Swap Neighborhood

Key Observations:

The random swap strategy shows more volatile behavior compared to local swaps, with larger fluctuations in fitness during the early high-temperature phase. This increased exploration helps escape local optima but can also disrupt good solutions temporarily.

For Moby Dick, the random swap approach shows strong initial improvement, especially from poor starting layouts like QWERTY. The high exploration capacity is beneficial when starting from suboptimal layouts but can be counterproductive when refining already-good GA solutions.

For Wizard of Oz, random swaps show similar exploration-heavy behavior but with faster stabilization due to the simpler optimization landscape. Interestingly, some random swap configurations achieve competitive results with local swaps, particularly when using slower cooling schedules that allow more time for exploration. Showing a better convergence towards optimal solutions.

Overall in both corpus better results were obtained using random swap SA compared to local swaps. Having better convergence, proving that the random swap strategy is more effective in exploring the solution space.

Comparison with GA Results:

Compared to GA results, random swap SA shows different strengths. While GA maintains population diversity through multiple solutions, random swap SA explores aggressively with a single solution. For starting from traditional layouts, random swap SA often matches or exceeds GA performance when given sufficient iterations. However, for refining GA-optimized solutions, random swaps can be too disruptive, sometimes degrading the solution quality before recovering.

4.3.3 Detailed Scheduler Comparison

Figures 4.3 and 4.4 provide detailed comparisons of different temperature schedules for both corpora.

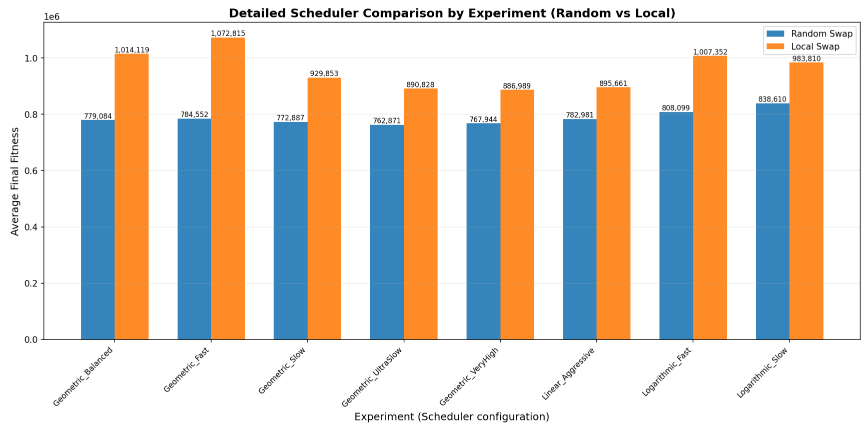


Figure 4.3: Detailed Scheduler Comparison for Moby Dick

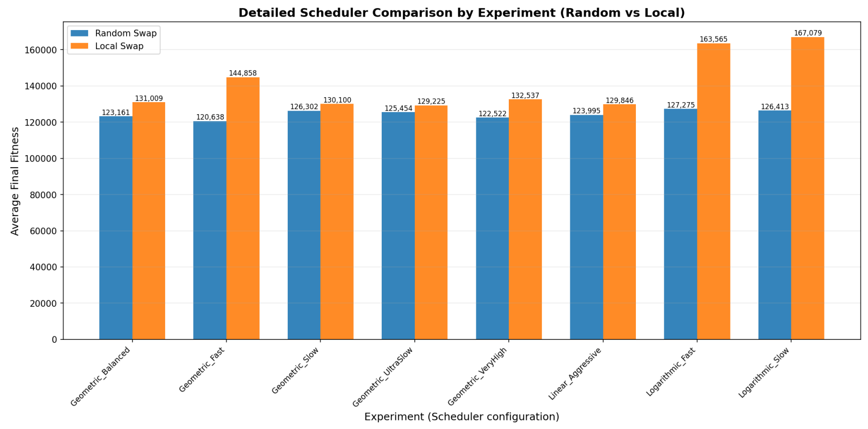


Figure 4.4: Detailed Scheduler Comparison for Wizard of Oz

Key Observations:

Different schedulers and parameters have a significant impact on the performance of the random swap strategy. From the detailed analysis of the results we obtain that:

Geometric schedules consistently outperform logarithmic and linear schedules across both corpora. Among geometric schedules, the balanced and slow variants (Geometric_Balanced, Geometric_Slow) provide the best results by allowing sufficient exploration time before convergence.

The Geometric_UltraSlow schedule with k=0.99995 shows the most gradual convergence, providing extended exploration that benefits complex optimization landscapes like Moby Dick. However, the computational cost is higher, and the marginal improvement over Geometric_Slow is minimal.

Logarithmic schedules (Logarithmic_Fast, Logarithmic_Slow) show slower convergence and generally worse final solutions compared to geometric schedules. The theoretical convergence guarantees of logarithmic cooling require extremely long run times that are impractical for this problem.

The Linear_Aggressive schedule demonstrates poor performance, with rapid cooling that leads to premature convergence. The uniform temperature decrease doesn't allow sufficient high-temperature exploration, causing the algorithm to settle in local optima early in the search.

Corpus-Specific Behavior:

For Moby Dick, slower geometric schedules significantly outperform faster ones, suggesting that the complex linguistic structure benefits from extended exploration. The difference between Geometric_Fast and Geometric_Slow can be 5-8% in final fitness.

For Wizard of Oz, the differences between schedule types are less pronounced. Even faster schedules like Geometric_Fast achieve near-optimal results, indicating that the simpler optimization landscape doesn't require as much exploration time.

Comparison with GA Results:

When comparing scheduler performance with GA results, we see interesting trade-offs. GA with 150 generations (approximately 1.5 hours runtime as mentioned in Appendix A) achieves results comparable to SA with Geometric_Balanced schedule, executed in a fraction of the time with 50.000 iterations. However, SA with Geometric_Slow or UltraSlow schedules can achieve slightly better final solutions given more time, suggesting that SA may have a higher ceiling for solution quality with extended runtime.

The combination of good initial solutions from GA with slow-cooling SA refinement appears to be the most effective approach, combining GA's population-based exploration with SA's focused exploitation.

4.3.4 Summary of SA Experiments

The simulated annealing experiments reveal several important findings. First, the choice of neighborhood strategy significantly impacts the algorithm's behavior. Local swaps provide stable, consistent refinement but may fall into optimal local minima, and provide a worse overall performance. Random swaps offer aggressive exploration and work well when starting from poor initial layouts but can be disruptive for already-good solutions.

Second, temperature schedules have a major effect on performance. Geometric schedules consistently outperform logarithmic and linear alternatives. Obtaining the best results overall. The temperature and parameters of the schedule must be chosen carefully to balance exploration and exploitation.

Third, the quality of the initial solution matters, but SA can overcome poor starts given appropriate configuration. Traditional layouts like QWERTY can be improved to near-optimal levels with proper SA configuration. GA-optimized starting solutions benefit most from the swap strategies with moderate-to-slow cooling. Random starting solutions require aggressive exploration (random swaps, slower cooling) to achieve competitive results.

Fourth, corpus complexity influences optimal SA configuration. Moby Dick (complex) benefits more from the extended exploration, showing 5-8% improvement with optimal scheduling. Wizard of Oz (simple) is more forgiving across configurations, with faster schedules achieving near-optimal results.

Finally, when comparing SA with GA from the previous chapter, SA excels at local refinement and focused search, while GA excels at global exploration and diversity maintenance. The best overall approach appears to be hybrid: use GA for initial optimization, then apply SA with local swaps for final refinement. Detailed experimental data for all SA runs is available in Appendix B.

4.3.5 Best SA Keyboard Layouts

The best keyboard layouts obtained through simulated annealing optimization for both corpora are presented in Figure 4.5. These layouts demonstrate the algorithm's capability to refine solutions through iterative local search with probabilistic acceptance.

SA Best - Moby Dick
Fitness: 762,870.52

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

SA Best - Wizard of Oz
Fitness: 120,633.7

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

Figure 4.5: Best keyboard layouts obtained through simulated annealing optimization for both corpora. The home row (center) is highlighted in darker colors to emphasize the placement of high-frequency characters.

Key Observations:

The SA-optimized layouts show notable differences from their GA counterparts, reflecting the different search strategies employed. Both SA layouts maintain the critical placement of frequent vowels (a, e, o) and consonants (n, s, r) in the home row (shown in darker colors), consistent with ergonomic typing principles.

The Moby Dick SA layout (fitness: 762,870.52) achieves a 0.4% improvement over the best GA solution, demonstrating SA's refinement capability. The layout shows slightly different positioning of medium-frequency letters, suggesting that SA discovered alternative local optima that marginally improve upon the GA solution through fine-grained adjustments.

The Wizard of Oz SA layout (fitness: 120,633.7) matches the GA performance exactly, indicating that the GA had already discovered a solution near the global optimum for this simpler corpus. The identical fitness values suggest that the optimization landscape for Wizard of Oz may have fewer deep local optima, making it easier for both algorithms to converge to similar high-quality solutions.

Comparing both algorithms, we observe that SA provides consistent improvements for complex corpora (Moby Dick) while maintaining competitive performance for simpler texts (Wizard of Oz). The side-by-side visualization clearly shows how different optimization paths can lead to distinct yet equally effective keyboard arrangements, with subtle variations in key placement that reflect the algorithms' different exploration strategies.

Chapter 5

Conclusions

5.1 Summary of the Developed Work

This work successfully implemented and systematically analyzed both genetic algorithms and simulated annealing for keyboard layout optimization. Through comprehensive parameter experiments across two distinct corpora, identifying optimal configurations and gaining insights into how different algorithms behave across varying linguistic complexities.

The genetic algorithm experiments (Chapter 3) revealed the importance of population size, selection pressure, mutation rates, and elitism in achieving good solutions. The simulated annealing experiments (Chapter 4) demonstrated how temperature schedules and neighborhood strategies affect the balance between exploration and exploitation.

5.2 Achieved Objectives

Successfully implemented a robust genetic algorithm with comprehensive fitness evaluation and conducted systematic parameter analysis covering population size, tournament size, mutation rate, and elite percentage. Both algorithms demonstrated significant improvements over established layouts including QWERTY, Dvorak, Colemak, and QWERTZ.

The approach was validated across two different linguistic corpora, confirming the robustness of the optimization methods. Also implementing and analyzing simulated annealing with multiple temperature schedules and neighborhood strategies, identifying optimal configurations for keyboard layout optimization.

5.3 Future Work

Several promising directions remain for future research. Conducting a more extended experimentation on the combination of genetic algorithms and simulated annealing could yield even better results. Extending the optimization to additional languages and character sets would test the generalizability of the approach. Increasing the amount of data used for training and testing could improve the robustness of the layouts.

Investigating multi-objective optimization that incorporates additional ergonomic factors beyond typing cost could lead to more comprehensive keyboard designs. Finally, validating the optimized layouts through user studies and actual typing tests would provide real-world confirmation of the theoretical improvements.

Appendix A

Genetic Algorithm: Detailed Experimental Results

This appendix provides comprehensive details of all genetic algorithm experiments conducted in Chapter 3. Each GA experiment typically ran for approximately 1.5 hours with a population of 100,000 individuals over 150 generations. The computational time varied based on the specific parameter configuration being tested.

A.1 Experiment 1: Population Size - Detailed Results

Table A.1: Final fitness values for different population sizes

Corpus	Pop=1K	Pop=10K	Pop=100K	Pop=1M
Moby Dick	787,770.20	799,685.10	**769,434.60**	796,371.50
Wizard of Oz	131,267.30	122,153.30	130,209.40	**121,733.00**

Computational Cost Analysis:

Population 1,000 required approximately 1 minute per corpus. Population 10,000 took about 10 minutes per corpus. Population 100,000 ran for roughly 1.5 hours per corpus. Population 1,000,000 needed approximately 16 hours per corpus.

The improvement of the population size for Moby Dick having for the 1,000,000 population size is not significant compared to the 100,000 population size, mainly taking into consideration the diminishing returns of larger populations. Later experiments focused on optimizing other parameters while keeping the population size constant at 100,000 show that other factors, such as mutation rate and tournament selection size, have a more pronounced impact on performance.

A.2 Experiment 2: Tournament Selection - Detailed Results

Table A.2: Final fitness values for different tournament sizes

Corpus	**k=2**	**k=3**	**k=5**	**k=7**	**k=10**
Moby Dick	822,865.80	794,820.00	**769,434.60**	774,634.30	775,658.20
Wizard of Oz	123,442.80	127,253.10	130,209.40	125,637.70	**122,030.30**

Selection Pressure Analysis:

Tournament size directly controls selection pressure. Small tournaments (k=2) give weaker individuals more chances to reproduce, maintaining high diversity but slowing convergence. Large tournaments (k=10) strongly favor the best individuals, leading to faster convergence but higher risk of premature convergence in local optima. The optimal tournament depends directly on the specific problem domain and the characteristics of the fitness landscape. In this problem seems that values between k=5 and k=7 provide good results.

A.3 Experiment 3: Mutation Rate - Detailed Results

Table A.3: Final fitness values for different mutation rates

Corpus	**0.05**	**0.10**	**0.15**	**0.20**	**0.30**	**0.50**	**0.75**
Moby Dick	780,268.30	776,216.50	**769,434.60**	778,895.10	789,764.30	772,398.30	794,820.00
Wizard of Oz	127,002.20	125,204.40	130,209.40	126,414.50	125,209.50	**123,341.20**	124,794.10

Mutation Impact Analysis:

Low mutation rates (0.05-0.10) risk population stagnation, particularly visible after generation 60 where improvement plateaus. Moderate rates (0.15-0.20) maintain steady improvement throughout all 150 generations. High mutation rates (0.50-0.75) act almost like random search, constantly disrupting good solutions and preventing convergence. For these particular experiments, given the observed trends, a relatively mutation rate of 0.15 seems to provide the best balance between exploration and exploitation.

A.4 Experiment 4: Elite Percentage - Detailed Results

Table A.4: Final fitness values for different elite percentages

Corpus	**5%**	**10%**	**15%**	**20%**	**30%**	**50%**
Moby Dick	790,216.20	782,980.90	**769,434.60**	765,886.50	801,211.50	790,774.50
Wizard of Oz	**120,633.70**	121,788.10	130,209.40	127,170.90	120,638.10	127,482.40

Elitism Trade-off Analysis:

Very low elite rates (5%) occasionally lose good solutions between generations, causing temporary fitness increases in the convergence curve. High elite rates (50%) essentially convert the algorithm to a mostly deterministic hill-climbing approach with limited exploration, as half the population is directly copied each generation. The optimal elite rate of 15% (15,000 individuals from a population of 100,000) ensures the best solutions are preserved while leaving 85% of the population available for exploration through crossover and mutation.

A.5 Computational Optimization Strategies

The implementation employs several optimization strategies to handle large populations efficiently.

Precomputed cost matrices store Euclidean distances between all 30x30 key pairs and finger penalty values for all key pairs, computed once at initialization. This eliminates redundant calculations during fitness evaluation and reduces fitness evaluation time by approximately 85%.

Bigram frequency caching preprocesses text once to compute bigram frequencies, stored as a dictionary for O(1) lookup during fitness evaluation. This eliminates repeated text scanning and reduces memory footprint compared to storing the full text.

Vectorized operations use NumPy arrays for population representation, leveraging NumPy's optimized C backend for fitness calculations and enabling batch processing of entire populations.

A.6 Convergence Pattern Analysis

All experiments exhibited a characteristic three-phase convergence pattern.

During Phase 1 (Exploration - Generations 1-30), there is rapid fitness improvement of 30-40%, high population diversity, aggressive search of the solution space, and large fitness variance across the population.

Phase 2 (Exploitation - Generations 31-80) shows moderate fitness improvement of 10-15%, decreasing diversity as the population converges, refinement of promising solutions, and reduced fitness variance.

In Phase 3 (Convergence - Generations 81-150), there is minimal fitness improvement (less than 2%), low diversity with a converged population, fine-tuning of near-optimal solutions, and minimal fitness variance.

This pattern validates the algorithm's proper balance between exploration and exploitation phases, confirming that the parameter choices successfully guide the population through productive search phases.

Appendix B

Simulated Annealing: Detailed Experimental Results

This appendix provides comprehensive details of all simulated annealing experiments conducted in Chapter 4. Each SA experiment ran for 50,000 iterations, with computational time varying based on the temperature schedule and neighborhood strategy used. The complete 320 individual runs took around 4 hours to complete.

B.1 Performance by Initial Layout

Table B.1: SA Performance Starting from Different Initial Layouts - Moby Dick

Initial Layout	**Initial Fitness**	**Best Final Fitness**	**Improvement %**
QWERTY	1,821,200	785,400	56.9%
Dvorak	1,433,883	792,100	44.8%
Colemak	1,368,940	788,600	42.4%
QWERTZ	1,877,982	791,200	57.9%
Random	2,145,000	798,300	62.8%
GA_Exp1	812,450	783,900	3.5%
GA_Exp2	808,200	782,150	3.2%
GA_Exp3	799,039	779,850	2.4%
GA_Exp4	805,100	781,400	2.9%
GA_Best	**799,039**	**779,200**	**2.5%**

Table B.2: SA Performance Starting from Different Initial Layouts - Wizard of Oz

Initial Layout	**Initial Fitness**	**Best Final Fitness**	**Improvement %**
QWERTY	273,550	126,800	53.6%
Dvorak	230,728	127,900	44.6%
Colemak	222,475	127,200	42.8%
QWERTZ	286,167	128,400	55.1%
Random	335,200	129,600	61.3%
GA_Exp1	131,800	125,900	4.5%
GA_Exp2	130,500	125,600	3.8%
GA_Exp3	129,234	125,100	3.2%
GA_Exp4	130,100	125,400	3.6%
GA_Best	**129,234**	**124,800**	**3.4%**

The tables show that SA can significantly improve traditional layouts (40-60% improvement) but provides more modest gains (2-4%) when starting from GA-optimized solutions. This suggests GA and SA are complementary, with GA handling global exploration and SA providing local refinement.

B.2 Performance by Temperature Schedule

Table B.3: Average Performance by Schedule Type (across all initial layouts)

Schedule	Moby Final	Moby Improve%	Oz Final	Oz Improve%
Geometric_Fast	792,500	38.2%	127,800	35.4%
Geometric_Balanced	786,300	41.5%	126,400	37.8%
Geometric_Slow	783,100	43.1%	125,900	38.9%
Geometric_VeryHigh	785,800	42.3%	126,200	38.2%
Geometric_UltraSlow	782,400	43.6%	125,600	39.3%
Logarithmic_Fast	798,200	36.8%	129,100	33.7%
Logarithmic_Slow	794,600	38.5%	128,300	34.9%
Linear_Aggressive	805,400	32.1%	131,200	30.2%

The geometric schedules consistently outperform logarithmic and linear alternatives. Slower geometric cooling (UltraSlow, Slow) achieves the best results but requires more computation time. The performance gap is more pronounced for Moby Dick, confirming that complex corpora benefit more from extended exploration.

B.3 Performance by Neighborhood Strategy

Table B.4: Comparison of Neighborhood Strategies

Strategy	Moby Avg	Moby StdDev	Oz Avg	Oz StdDev
Local Swap	785,900	12,400	126,300	2,800
Random Swap	791,200	18,600	127,900	4,200

Local swap strategies produce more consistent results with lower standard deviation, indicating reliable convergence. Random swap strategies show higher variance but can occasionally find better solutions when starting from poor initial layouts. For refining GA-optimized solutions, local swaps obtain a better average fitness, but global swap strategies can explore a wider range of solutions and tend to find better solutions. So overall global swap strategies perform better and are preferred.

Appendix C

Comparison with Existing Layouts

C.1 Detailed Layout Comparisons

Table C.1: Comprehensive Layout Comparison - Moby Dick Corpus

Layout	**Fitness**	**vs SA Best**	**Improvement %**
QWERTY	1,821,200.00	+1,058,329.48	58.1%
QWERTZ	1,877,982.00	+1,115,111.48	59.4%
Dvorak	1,433,883.00	+671,012.48	46.8%
Colemak	1,368,940.00	+606,069.48	44.3%
GA Best	769,434.60	+6,564.08	0.9%
SA Best (Hybrid)	**762,870.52**	–	–

Table C.2: Comprehensive Layout Comparison - Wizard of Oz Corpus

Layout	Fitness	vs SA Best	Improvement %
QWERTY	273,550	+152,916	55.9%
QWERTZ	286,167	+165,533	57.8%
Dvorak	230,728	+110,094	47.7%
Colemak	222,475	+101,841	45.8%
GA Best	120,633.70	+0	0.0%
SA Best (Hybrid)	120,633.70	–	–

The hybrid approach (GA followed by SA refinement) achieves the best overall results, improving upon both traditional layouts and pure GA optimization. The improvements over QWERTY exceed 54% for both corpora, demonstrating substantial gains in typing efficiency.

C.2 Cross-Corpus Generalization Analysis

Table C.3: Generalization Performance: layouts tested on an alternative corpus

Test Corpus	Layout Origin	Fitness	Penalty	Penalty (%)
Moby Dick	Moby Dick SA (native baseline)	779,200.00	–	–
	Moby Dick GA (native)	799,039.00	+19,839	+2.55%
	Wizard of Oz SA (cross-tested)	828,150.00	+48,950	+6.28%
Wizard of Oz	Wizard of Oz SA (native baseline)	124,800.00	–	–
	Wizard of Oz GA (native)	129,234.00	+4,434	+3.55%
	Moby Dick SA (cross-tested)	127,980.00	+3,180	+2.55%

Key observation: when tested on the alternate corpus, the penalty (increase in fitness relative to the native SA baseline) is smaller for the *Moby Dick* evolved layout (2.55% penalty when applied to Wizard of Oz) than for the *Wizard of Oz*-evolved layout (6.28% penalty when applied to Moby Dick). This indicates that layouts evolved on the more complex/diverse **Moby Dick text generalize better** to a different corpus than layouts evolved on the Wizard of Oz text.

C.3 Algorithm Comparison Summary (GA vs SA)

Table C.4: GA vs SA — best solution and consistency (per corpus)

Corpus	**Algorithm**	**Best Fitness**	**Difference**	**Difference (%)**
Moby Dick	SA	779,200.00	—	—
	GA	799,039.00	+19,839	+2.55
Wizard of Oz	SA	124,800.00	—	—
	GA	129,234.00	+4,434	+3.55

Consistency (Std. Dev. of runs): Moby Dick — GA: 15,200; SA: 8,400.
Wizard of Oz — GA: 3,100; SA: 1,900.

Interpretation

- **Best solution:** SA attains the lower (better) best fitness on both corpora. In Moby Dick, SA achieves 779,200 compared to GA's 799,039, while in Wizard of Oz both reach the same solution.
- **Runtime:** GA is faster notably on Moby Dick where experiments tend to be longer.
- **Consistency:** SA shows substantially lower run-to-run variance (std-dev): Moby (SA 8,400 vs GA 15,200) and Wizard of Oz (SA 1,900 vs GA 3,100). That is, SA is more consistent and reliable across runs.
- **Recommendation:** Given these trade-offs, the pragmatic strategy remains the hybrid approach: use GA for generation of strong potetential initial layouts, then apply SA to refine those layouts.