ITP4514 Artificial Intelligence and Machine Learning

Project

Name: Tse Man Fung

Student ID: 230026964

Class: SE-2A

# Table of Contents

[1. Table of Contents 2](#_Toc182612669)

[2. A1 4](#_Toc182612670)

[3. A2 6](#_Toc182612671)

[4. Section B 7](#_Toc182612672)

[4.1. Introduction 7](#_Toc182612673)

[4.2. Dataset 7](#_Toc182612674)

[4.2.1. Novel 7](#_Toc182612675)

[4.2.2. Cangjie file 8](#_Toc182612676)

[4.3. Define Keyboard 9](#_Toc182612677)

[4.3.1. Keyboard XY Value Calculation Table 11](#_Toc182612678)

[4.4. Train method 12](#_Toc182612679)

[4.4.1. Initialization 12](#_Toc182612680)

[4.4.2. Evaluation 12](#_Toc182612681)

[4.4.3. Selection 12](#_Toc182612682)

[4.4.4. Crossover and Mutation 13](#_Toc182612683)

[4.4.5. Iteration 13](#_Toc182612684)

[4.4.6. Results 14](#_Toc182612685)

[4.5. Coding 15](#_Toc182612686)

[4.5.1. Genetic Algorithm Implementation 15](#_Toc182612687)

[4.5.2. Keyboard Class Functionality 15](#_Toc182612688)

[4.5.3. Data Handling 15](#_Toc182612689)

[4.5.4. User Interaction 16](#_Toc182612690)

[4.5.5. Visualization 16](#_Toc182612691)

[4.6. Result 17](#_Toc182612692)

[4.6.1. Parameter 17](#_Toc182612693)

[4.6.2. Label 19](#_Toc182612694)

[4.6.3. Chart 20](#_Toc182612695)

[4.6.4. Result 1 21](#_Toc182612696)

[4.6.5. Result 2 22](#_Toc182612697)

[4.6.6. Result 3 23](#_Toc182612698)

[4.6.7. Result 4 24](#_Toc182612699)

[4.6.8. Result 5 25](#_Toc182612700)

[4.7. Conclusion 26](#_Toc182612701)

[5. Reference 27](#_Toc182612702)

# A1

I choose Breadth-First Search and A\* Search

|  |  |
| --- | --- |
| uninformed search algorithm | Breadth-First Search |
| informed search algorithm | A\* Search |

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Compare the results of the two algorithms and analyze factors like time complexity and path optimality.

**Time Complexity**

Breadth-First Search (BFS) has a time complexity of O(b^d), where **b** represents the branching factor, or the average number of children per node, and **d** is the depth of the shallowest solution. BFS explores all nodes at the current depth before proceeding to deeper levels, which can result in significant time consumption, especially in graphs that are wide or have considerable depth. In contrast, A\* Search also has a time complexity of O(b^d), but it is typically more efficient in practice. This efficiency arises from its use of a heuristic that guides the search process. The effectiveness of A\* largely depends on the quality of the heuristic; when designed well—meaning it is admissible and consistent—A\* can significantly reduce the number of nodes explored compared to BFS.

**Path Optimality**

When it comes to path optimality, BFS guarantees the shortest path in terms of the number of edges in unweighted graphs or when all edges have the same weight. However, if the edges have different weights, BFS does not ensure that the shortest path is found. On the other hand, A\* guarantees an optimal path if the heuristic used is admissible (never overestimates the true cost) and consistent (the estimated cost is always less than or equal to the cost from any neighboring node plus the step cost). A\* is specifically designed to intelligently find the least-cost path by taking into account both the cost to reach a node and the estimated cost to reach the goal.

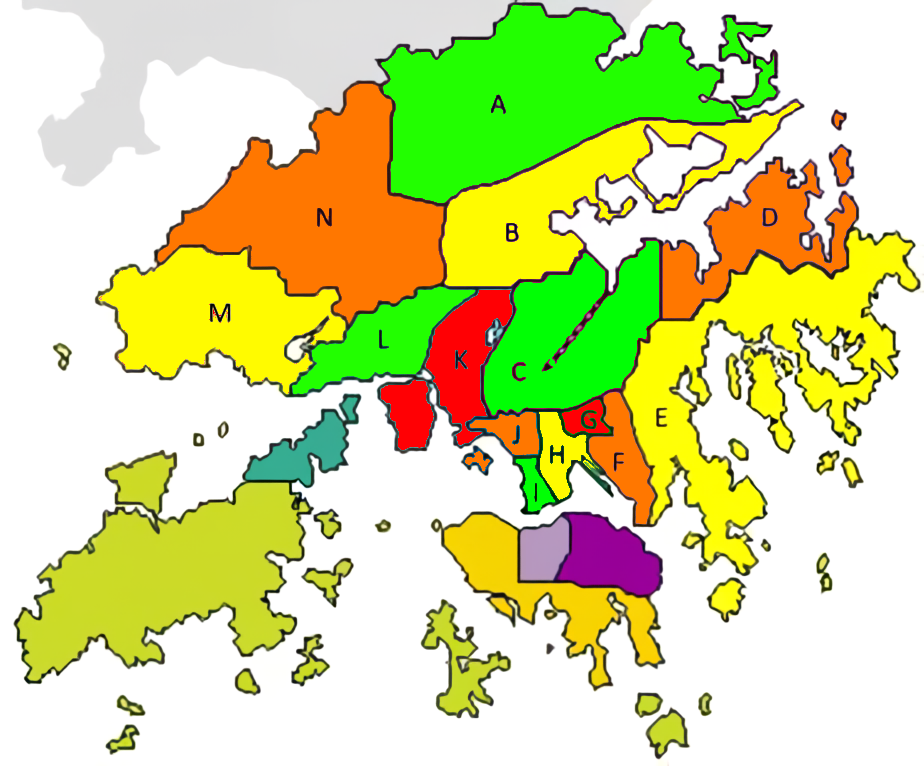
# A2

一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Output:

Found 24576 solutions with 4 colors:

[{'C': 'Green', 'B': 'Yellow', 'H': 'Yellow', 'F': 'Orange', 'J': 'Orange', 'K': 'Red', 'L': 'Green', 'N': 'Orange', 'G': 'Red', 'E': 'Yellow', 'A': 'Green', 'D': 'Orange', 'I': 'Green', 'M': 'Yellow'}, ……

It needs at least 4 colors to fill the map and there is 24576 solutions to fill the map with 4 colors.

# Section B

Report

Topic: Use genetic algorithm to find out the best keyboard for Cangjie Input Method

## Introduction

In the realm of technology and human-computer interaction, the design of efficient input devices like keyboards plays a crucial role in enhancing user experience and productivity. The Cangjie input method, named after the legendary figure CangJie, is a widely used system for typing Chinese characters. This project aims to explore the optimization of keyboard layouts specifically tailored for the Cangjie input method using genetic algorithms.

Genetic algorithms (GAs) are inspired by the principles of natural selection and evolution. They offer a robust framework for solving complex optimization problems by iteratively improving candidate solutions based on defined criteria. By applying GAs to keyboard design, we can systematically evaluate various configurations to determine the most effective layout that enhances typing efficiency and minimizes user fatigue.

This report outlines the methodology, implementation, and results of using genetic algorithms to identify the best keyboard design for the Cangjie input method.

## Dataset

### Novel

To facilitate the creation of a comprehensive dataset, I developed a web scraper (getArticle.py) designed to systematically extract continuous and coherent sentences from various novel websites. The primary objective of this program is to crawl multiple online literary resources (unknown, 2024), collecting a substantial volume of text data. The extracted sentences are subsequently saved as text files, contributing to the development of a robust corpus suitable for further linguistic analysis and research applications. This automated approach ensures efficiency and scalability in gathering diverse textual samples for dataset construction.



### Cangjie file

Due to the necessity of consolidating Cangjie files obtained from multiple sources, I developed a script called makeDataset.py to standardize the data into a unified format. This script seamlessly integrates the information into a single JSON file, unicode2cangjie.json. The resulting dataset maps Chinese characters to their respective Unicode values and corresponding Cangjie keys. For example, the JSON structure includes entries such as {"3007":["W"],"4e00":["M"],"4e59":["NU"],"4e01":["MN"],"4e03":["JV"],"4e43":["NHS"],…}, where each Chinese word is linked to its Unicode representation and the associated Cangjie input keys. This structured approach facilitates easier access and analysis of the data for subsequent processing and applications.

## Define Keyboard

The Keyboard class is designed to model a keyboard layout optimized for typing efficiency. It incorporates several key features:

1. **Finger Grouping**: The keyboard is organized based on finger assignments, where each key corresponds to a specific finger.   
     
   The mapping is as follows:
   * **Left Hand**:
     + 0 → Left little finger
     + 1 → Left ring finger
     + 2 → Left middle finger
     + 3 → Left index finger
   * **Right Hand**:
     + 5 → Right index finger
     + 6 → Right middle finger
     + 7 → Right ring finger
     + 8 → Right little finger
2. **Initialization**: When creating an instance of the Keyboard class, users can specify custom finger groupings and keyboard layouts. If none are provided, default settings are used. The default finger group is organized to optimize the distribution of keys across both hands, while the default keyboard layout maps the letters from 'A' to 'Z' to specific positions.
3. **Distance Calculation**: The class includes a method to calculate the distance between keys based on their positions. This is crucial for understanding the ergonomic aspects of typing, as it helps minimize finger movement and reduce fatigue. The distance is computed using a 2D coordinate system, allowing an evaluation of the efficiency of finger movement between keys.
4. **Key Coordinates**: The layout defines specific coordinates for each key. For example, the key 'Q' is positioned at (0, 0) on the keyboard (first row). The key 'W', located immediately to the right of 'Q', is positioned at (4, 0).
5. In the first row, all keys can be represented as coordinates of the form (X,0), where X varies based on the key's position.
6. In the second row, all keys directly below 'Q' will have their X-coordinate defined as (X+1,4), where X is the X-coordinate of the corresponding key in the first row. This means, for example, if 'A' is directly below 'Q', its position will be adjusted to account for the increase in horizontal distance.
7. In the third row, the keys will have their X-coordinates defined as (X+3,8). This reflects a further increase in horizontal distance, ensuring each key's position is accurately represented based on its relative location to the keys above it.

### Keyboard XY Value Calculation Table

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Size | |  | Lower-Upper (Offset) | |  | Lower-Upper (Scale) | |
|  | 2: |  |  | X | Y |  |  |  |  |  |  |
| Upper Left | 106 | 0 |  | 53 | 53 |  | / | / |  | / | / |
| Lower Right | 159 | 53 |  |  |  |  | / | / |  | / | / |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | W: |  |  | X | Y |  |  |  |  |  |  |
| Upper Left | 133 | 53 |  | 53 | 53 |  | 27 | 53 |  | 50.94% | 100.00% |
| Lower Right | 186 | 106 |  |  |  |  | 27 | 53 |  | 50.94% | 100.00% |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | S: |  |  | X | Y |  |  |  |  |  |  |
| Upper Left | 146 | 106 |  | 53 | 54 |  | 13 | 53 |  | 24.53% | 98.15% |
| Lower Right | 199 | 160 |  |  |  |  | 13 | 54 |  | 24.53% | 100.00% |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | X: |  |  | X | Y |  |  |  |  |  |  |
| Upper Left | 174 | 160 |  | 52 | 53 |  | 28 | 54 |  | 53.85% | 101.89% |
| Lower Right | 226 | 213 |  |  |  |  | 27 | 53 |  | 51.92% | 100.00% |

## Train method

The **train method** is a crucial component of the genetic algorithm implemented within the GA class. Its primary purpose is to evolve keyboard layouts over multiple generations to optimize typing efficiency.

Here’s an overview of how it works:

### **Initialization**

The method begins by initializing a population of keyboard layouts. This population can be randomly generated or based on a predefined keyboard layout.

### **Evaluation**

Each keyboard in the population is evaluated for its typing efficiency by calculating the total distance that fingers travel when typing a given string. This is done using the check\_total\_distance method, which accounts for the finger movements required for each character in the string.

### **Selection**

After evaluating all keyboards, the method selects the top-performing layouts based on their total distance scores. The best layouts are retained for the next generation.

Here is the formula derivation

1. Definitions:
   * Let **P** be the population\_size.
   * Let **m** be the mutation\_rate.
   * Let **N** be the number of parents selected for the next generation.
2. Parent Size Calculation:  
   The parent size is calculated as:

|  |
| --- |
| parent\_size=√**P** |

1. **Random Size Calculation**:  
   The number of keyboards to be randomly generated (mutated) in the next generation is:

|  |
| --- |
| random\_size = floor(parent\_size \* **m**) |

1. **Next Generation Composition**:  
   The total number of keyboards in the next generation (N) consists of the top-performing keyboards from the current population and the new randomly generated keyboards:

|  |
| --- |
| N = Top\_keyboard = population[:int(parent\_size)−random\_size] + random\_keyboard(random\_size) |

### **Crossover and Mutation**

The new generation of keyboards is created through crossover, where features from top-performing keyboards are combined. Additionally, random mutations are introduced to maintain genetic diversity, which helps prevent premature convergence on suboptimal solutions.

### **Iteration**

The process is repeated for a specified number of generations. Over time, the population evolves, ideally resulting in layouts that minimize finger movement and improve typing speed.

### **Results**

At the end of the training process, the best keyboard layout is saved, and performance metrics are logged for analysis.

## Coding

The implementation of the genetic algorithm and the associated methods within the GA and Keyboard classes.

### **Genetic Algorithm Implementation**

The GA class contains methods for initializing populations, evaluating layouts, performing crossover, and applying mutations. These methods leverage Python's capabilities, such as multiprocessing, to efficiently handle large populations and computations.

### **Keyboard Class Functionality**

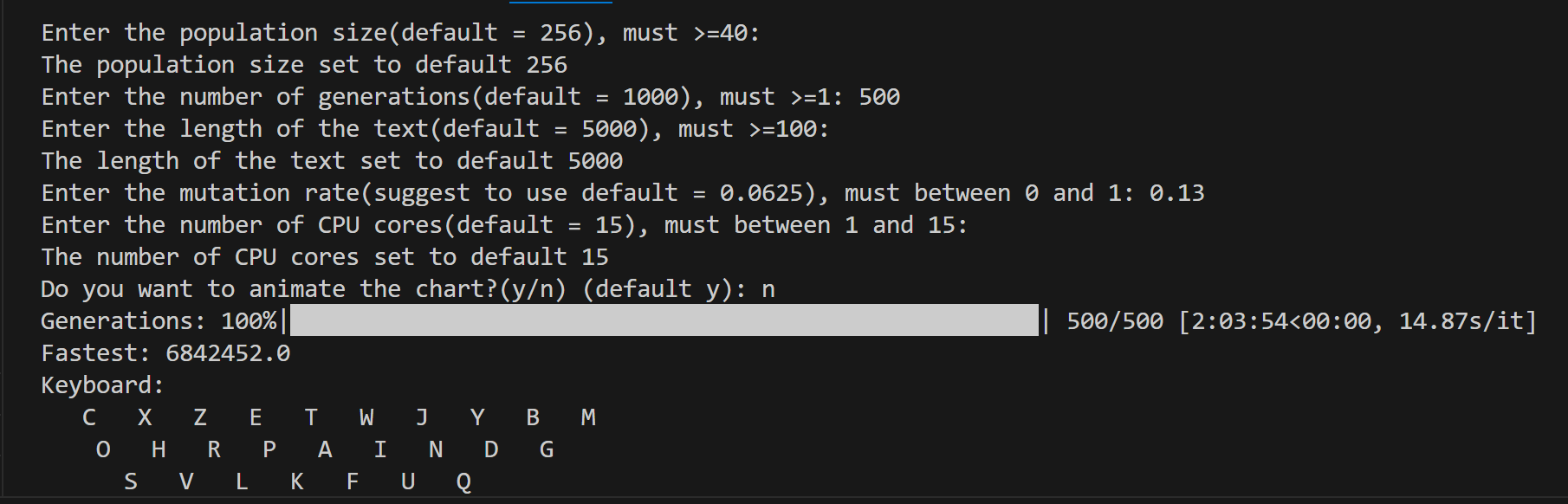
The Keyboard class encapsulates the representation of a keyboard layout. It defines methods for setting up new layouts, calculating distances between keys, and managing finger assignments. This class is essential for evaluating the effectiveness of different layouts.

### **Data Handling**

The code includes methods for reading from and writing to JSON files, enabling the saving and loading of keyboard configurations. This functionality allows for experimentation with different layouts and the preservation of results across sessions.

### **User Interaction**

The main script facilitates user interaction for configuring parameters like population size, number of generations, and mutation rates. This allows users to customize the training process according to their preferences.



### **Visualization**

The implementation also incorporates data visualization techniques using libraries like Matplotlib to display the performance of different keyboard layouts over generations, helping users understand the optimization process visually.

## Result

### Parameter

|  |  |
| --- | --- |
| Parameter | Description |
| Population size | The population size refers to the total number of keyboard layouts generated and evaluated in each iteration of the genetic algorithm. A larger population size allows for greater diversity in keyboard configurations, which can enhance the algorithm's ability to explore various solutions and potentially find more optimal layouts. |
| Generation | The generation indicates the number of iterations the genetic algorithm undergoes to evolve the population. Each generation involves evaluating the current population, selecting the best-performing layouts, and creating a new population through crossover and mutation. More generations typically lead to better optimization as the algorithm refines the keyboard layouts over time. |
| Number of text line | Number of Text Lines  The number of text lines specifies how many lines of text are used for evaluating the keyboard layouts. This parameter is crucial for testing the performance of the layouts, as it simulates real-world typing scenarios. More text lines can provide a more comprehensive assessment of typing efficiency and the effectiveness of the layout. |
| Random text | Random text refers to whether the text used for evaluation is generated randomly or if it follows a specific pattern or set of words. Using random text can help assess the keyboard layout's versatility and performance across varied typing scenarios, making the evaluation more realistic. |
| Mutation rate | The mutation rate is the percentage of the population that undergoes random changes during the evolution process. This parameter introduces variability and helps maintain genetic diversity within the population. A balanced mutation rate is essential; too high can lead to chaotic changes, while too low may result in stagnation and prevent the discovery of new, optimal layouts. |

### Label

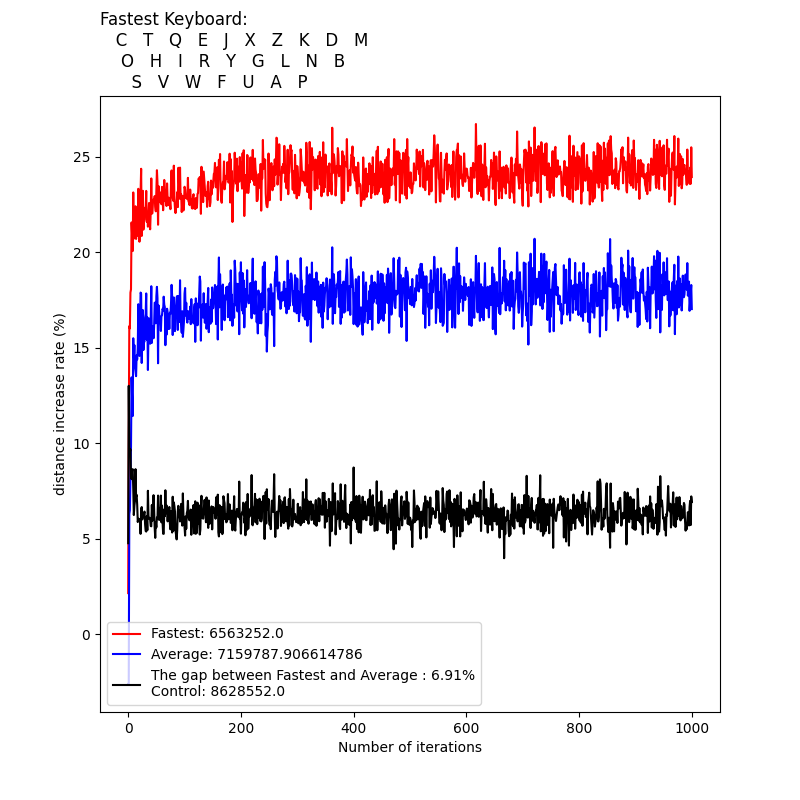
|  |  |
| --- | --- |
| Label | Description |
| Distance of fastest keyboard | The distance of the fastest keyboard measures the total finger movement required to type a given text using the most optimized keyboard layout identified by the genetic algorithm. This distance reflects the efficiency of the layout, with lower values indicating reduced finger travel and improved typing speed. |
| Distance of average keyboard | The distance of the average keyboard quantifies the total finger movement needed to type the same text using the average-performing keyboard layout from the population. This value serves as a benchmark for comparison, highlighting the typical efficiency of keyboards that are not specifically optimized. |
| Gap between Distance of fastest keyboard and Distance of average keyboard | The gap between distances indicates the stability of the entire population. A gap of 0% signifies that all keyboards in the population are completely identical. |
| Distance of control keyboard (QWERTY) | The distance of the control keyboard (QWERTY) denotes the total finger movement required to type the same text using the standard QWERTY layout. This serves as a reference point for evaluating the performance of the optimized keyboards, allowing comparisons to traditional typing efficiency. |

### Chart

|  |  |
| --- | --- |
| Chart Elements | Description |
| X-Axis | Represents the Number of Iterations, indicating the progression of generations in the genetic algorithm. |
| Y-Axis | Shows the Distance Increase Rate (%). This percentage is used to normalize the data, making it easier to observe trends despite variations in text length caused by the presence of random text. |
| The top of the chart | At the top of the chart, the fastest keyboard from the previous generation is displayed, providing a quick reference for the best-performing layout during that iteration. |

### Result 1

|  |  |
| --- | --- |
| Population size | 256 |
| Generation | 1000 |
| Number of text line | 5000 |
| Random text | Yes |
| Mutation rate | 6.25% |



### Result 2

|  |  |
| --- | --- |
| Population size | 256 |
| Generation | 500 |
| Number of text line | 5000 |
| Random text | Yes |
| Mutation rate | 13% |

一張含有 文字, 螢幕擷取畫面, 字型, 繪圖 的圖片

自動產生的描述

### Result 3

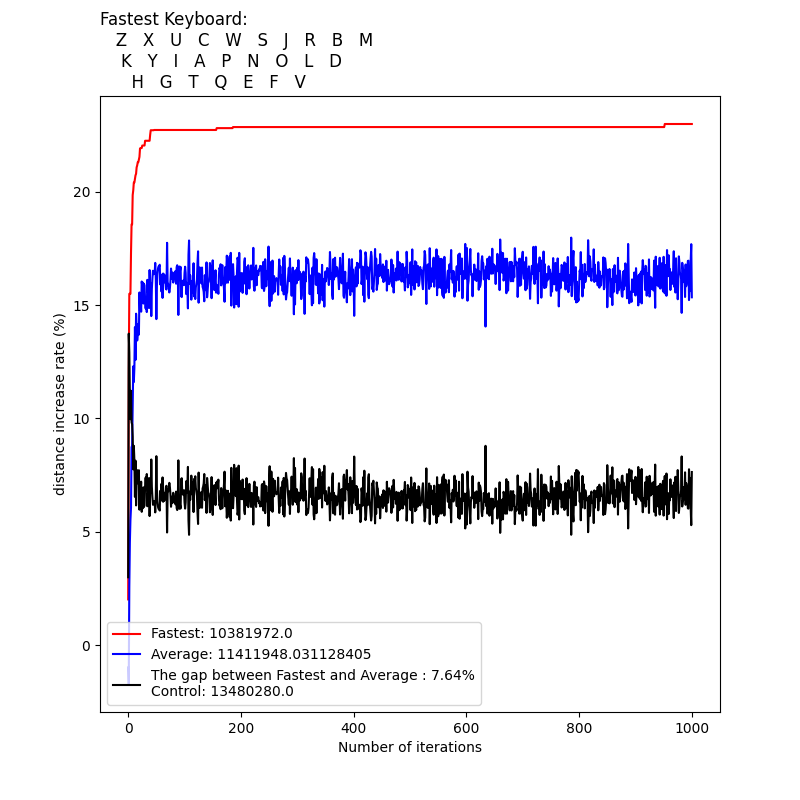
|  |  |
| --- | --- |
| Population size | 100 |
| Generation | 100 |
| Number of text line | 1000 |
| Random text | Yes |
| Mutation rate | 10% |

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

自動產生的描述

### Result 4

|  |  |
| --- | --- |
| Population size | 256 |
| Generation | 1000 |
| Number of text line | 5000 |
| Random text | No |
| Mutation rate | 6.25% |



### Result 5

|  |  |
| --- | --- |
| Population size | 100 |
| Generation | 100 |
| Number of text line | 1000 |
| Random text | Yes |
| Mutation rate | 0% |

一張含有 文字, 螢幕擷取畫面, 繪圖, 圖表 的圖片

自動產生的描述

## Conclusion

From the results 1-3 above, we can identify some characteristics of the best keyboard for the Cangjie Input Method:

一張含有 字型, 文字, 螢幕擷取畫面, 數字 的圖片

自動產生的描述

* **Green** indicates patterns, such as X and Z being connected in the edge rows (1st and 3rd rows).
* **Yellow** indicates keys that will appear in the edge rows.
* **Blue** indicates keys that will appear in the middle row (2nd row).
* **Purple** indicates keys that must occupy this position.

|  |  |
| --- | --- |
| **Green** | A set of keys with clear relationships or combinations. |
| **Yellow** | Placing these keys in the edge rows reduces their impact on frequently input keys, as they are less often used. |
| **Blue** | Keys that are input more frequently are positioned in the middle row, where they are closer together both above and below, resulting in shorter distances to the previous and next keys. |
| **Purple** | The case of the key 'M' is unique; it is thought that 'M' is input most often, because the finger for that position is deemed not needing to move. |

# Reference

HKSAR Policy OfficeDigital. (2024年7月25日). Hong Kong Supplementary Character Set related information | DATA.GOV.HK. 擷取自 DATA.GOV.HK: https://data.gov.hk/en-data/dataset/hk-dpo-dpo\_hp-hong-kong-supplementary-character-set-related-information

TW of Digital AffairsMinistry. (2015年2月1日). CNS11643中文標準交換碼全字庫(簡稱全字庫) ｜ 政府資料開放平臺. 擷取自 data.gov.tw: https://data.gov.tw/dataset/5961

TW of Digital AffairsMinistry. (2015年2月1日). 技術支援 \\ 全字庫授權 - 全字庫 CNS11643 (2024). 擷取自 www.cns11643.gov.tw: https://www.cns11643.gov.tw/pageView.jsp?ID=59

unknown. (2024年11月14日). 嗶哩輕小說\_日本動漫輕小說在線閱讀. 擷取自 tw.linovelib.com: https://tw.linovelib.com/