НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:
Завідувач кафедри
_____ О.І. Чумаченко
“___”_____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи та методи штучного інтелекту»
спеціальності 122 «Комп’ютерні науки»
на тему: «Створення ефективної україномовної розкладки клавіатури з використанням генетичного алгоритму»

Виконав:
студент IV курсу, групи КІ-92
Бурков Антон Олексійович _____

Керівник:
проф., д.т.н., Чумаченко Олена Іллівна _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Надія Василівна _____

Консультант з нормоконтролю:
старший інженер, Гончарук Максим Миколайович _____

Рецензент:
старший викладач, к.т.н, Гордієнко Олександр Миколайович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» Інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший(бакалаврський)

Спеціальність – 122 «Комп’ютерні науки»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Чумаченко
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Буркову Антону Олексійовичу

1. Тема роботи «Створення ефективної україномовної розкладки клавіатури з використанням генетичного алгоритму», керівник роботи Чумаченко Олена Іллівна, професор кафедри ШІ, затверджені наказом по університету від «30» травня 2023 р. № 2065-с

2. Термін подання студентом роботи 12.06.2023

3. Вихідні дані до роботи: сучасні статті у публіцистичному стилі на різні тематики.

4. Зміст роботи: Аналіз предметної області дослідження та даних, побудова моделі генетичного алгоритму для вирішення заданої задачі, аналіз отриманих результатів.

5. Перелік ілюстративного матеріалу: блок-схема алгоритма, розкладки клавіатур, графіки з аналізом результатів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, канд.економ.наук, доцент		

7. Дата видачі завдання: 21 лютого 2023 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Підготовка даних до роботи	12.03.2023	Виконано
2.	Вивчення літератури за темою роботи	20.04.2023	Виконано
3.	Підготовка першого розділу	07.05.2023	Виконано
4.	Підготовка другого розділу	18.05.2023	Виконано
5.	Розробка генетичного алгоритму	21.05.2023	Виконано
6.	Підготовка третього розділу	24.05.2023	Виконано
8.	Оформлення розділів відповідно до нормоконтролю	26.05.2023	Виконано
9.	Оформлення дипломної роботи	28.05.2023	Виконано
10.	Підготовка презентації доповіді	30.05.2023	Виконано

Студент Антон БУРКОВ

Керівник Олена ЧУМАЧЕНКО

РЕФЕРАТ

Дипломна робота: 105 с., 6 табл., 35 рис., 2 додатки, 20 джерел.

ГЕНЕТИЧНИЙ АЛГОРИТМ, ОПТИМІЗАЦІЯ, ЕВОЛЮЦІЙНІ АЛГОРИТМИ, РОЗКЛАДКА КЛАВІАТУРИ

Об'єкт дослідження – україномовні розкладки клавіатури.

Предмет дослідження – оптимізація загальної дистанції, яку проходять пальці під час друку на україномовній розкладці клавіатури.

Мета роботи – розробка моделі генетичного алгоритму для створення ефективної розкладки клавіатури.

Актуальність роботи пов'язана з неефективністю існуючих україномовних розкладок клавіатури.

У дослідженні проводиться аналіз існуючих клавіатурних розкладок для української мови, з урахуванням частотного аналізу використання букв в українській мові. На основі зібраних даних та урахування загальної дистанції, яку проходять пальці під час друку, розробляється математична модель генетичного алгоритму для оптимізації розкладки.

Для оцінки ефективності створеної з використанням генетичного алгоритму розкладки проводяться експерименти та порівняння з існуючими розкладками. Результати експериментів демонструють переваги розробленої україномовної розкладки клавіатури.

ABSTRACT

Thesis: 105 pages, 6 tables, 35 figures, 2 appendices, 20 sources.

GENETIC ALGORITHM, OPTIMIZATION, EVOLUTIONARY ALGORITHMS, KEYBOARD LAYOUT

The object of research is Ukrainian keyboard layouts.

The subject of research is the optimization of the total finger movement distance during typing on the Ukrainian keyboard layout.

The aim of the work is to develop a model of a genetic algorithm for creating an efficient keyboard layout.

The relevance of this work is associated with the inefficiency of existing Ukrainian keyboard layouts.

The research involves an analysis of existing keyboard layouts for the Ukrainian language, taking into account the frequency analysis of letter usage in the Ukrainian language. Based on the collected data and considering the total finger distance during typing, a mathematical model of a genetic algorithm is developed for layout optimization.

Experiments and comparisons with existing layouts are conducted to evaluate the effectiveness of the created keyboard layout using the genetic algorithm. The results of the experiments demonstrate the advantages of the developed Ukrainian keyboard layout.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ГА – генетичний алгоритм.

ПП – програмний продукт.

ВСТУП

Клавіатура є невід'ємною частиною життя сучасної людини у роботі, бо є однією з основних пристроїв вводу комп’ютера. Кожен день мільйони україномовних людей у всьому світі використовують клавіатуру у роботі, навчанні та спілкуванні. Тому зі збільшенням кількості україномовних користувачів комп’ютерів зростає і важливість продуктивності та комфорту роботи з клавіатурою при вводі документів, повідомлень та інших текстових даних українською мовою, а відповідно і ефективність україномовної розкладки.

Актуальність створення більш ефективної україномовної розкладки клавіатури виникає з необхідності пристосування клавіатури до вводу тексту українською мовою. На жаль, існуюча українська розкладка клавіатури не була створена спеціально для роботи з українською мовою, а є лише адаптацією найбільш популярної російськомовної розкладки, що знижує її ефективність та зручність використання. Тому, метою даної дипломної роботи є створення більш ефективної української розкладки клавіатури, що буде пристосована під вимоги сучасної української мови.

Для досягнення поставленої мети в даній дипломній роботі використовується генетичний алгоритм, який є потужним інструментом для пошуку оптимального рішення в складних задачах. Застосування генетичного алгоритму для створення більш ефективної української розкладки клавіатури є підходом, який допоможе з усіх можливих варіацій розкладок знайти ту, яка мінімізує переміщення пальців при наборі текстових даних українською мовою, а тобто і пришвидшити комфорт та швидкість набору.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Розкладки клавіатур

Початком створення сучасних розкладок клавіатур можна вважати створення однієї з найбільш популярних розкладок у світі "QWERTY". Ця розкладка була створена у 1874 році американським винахідником Крістофером Шоулзом для друкарської машинки Ремінгтон 1, а у 1882 році була випущена друкарська машинка Ремінгтон 2, у якій вже була представлена оновлена розкладка "QWERTY" з розміщенням клавіш латинських букв, яке використовується і сьогодні [1]. Розкладка була адаптована для швидкого набору телеграм, розкодованих з використанням азбуки Морзе [1].

Відслідкувати історію походження розкладок клавіатур з кирилицею досить важко, але сучасна найпопулярніша розкладка з кирилицею "ЙЦУКЕН" була затверджена ГОСТом СРСР 6431-75 у 1975 році для обладнання клавіатур для пишучих машин [2]. Поскільки цей документ поширювався на всі республіки СРСР, "ЙЦУКЕН" одразу поширився і на територію України, але ця розкладка у свої початковій своїй формі представляє усі літери російського алфавіту. Зараз існує "ЙЦУКЕН" для кожної мови, що використовує кириличний алфавіт, такі мови як білоруська, казахська, таджицька, та інші [3].

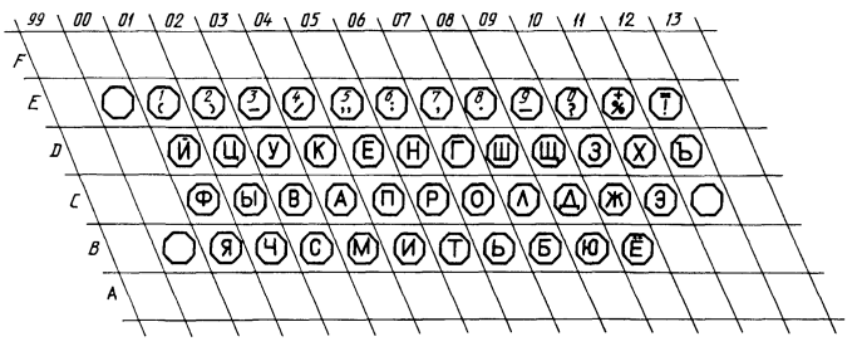


Рисунок 1.1 — Схема розкладки "ЙЦУКЕН" за ГОСТом СРСР 6431-75 [2]

Звісно "ЙЦУКЕН" отримав свою адаптацію і на українській мові. У 1996 році українська версія розкладки була затверджена у ДСТУ 3470-96. І зараз ця версія використовується на більшості девайсах україномовних користувачів.

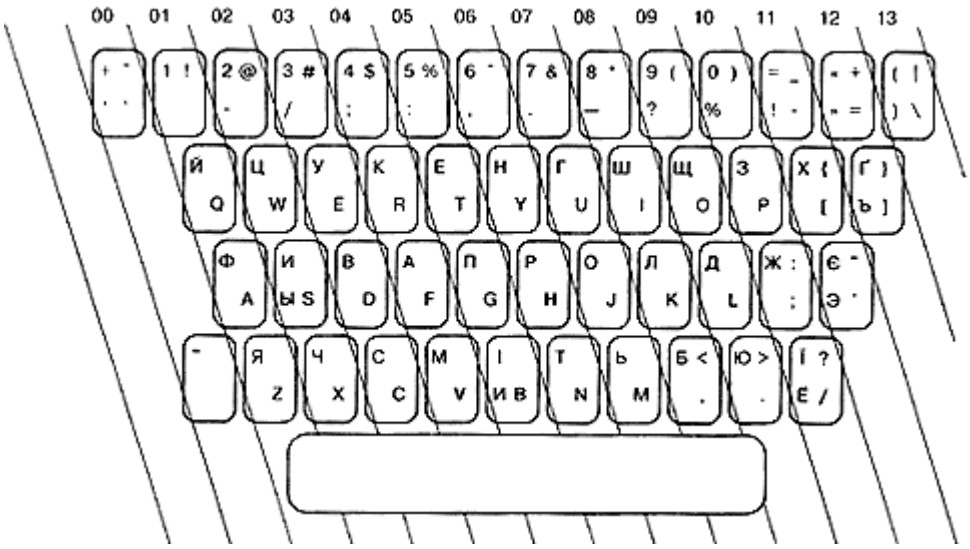


Рисунок 1.2 — Схема сучасної україномовної розкладки "ЙЦУКЕН" [3]

Відповідно до історій розкладок "QWERTY" та "ЙЦУКЕН" можемо зробити висновок, що не всі найпопулярніші розкладки клавіатур є ефективні. "QWERTY" була розроблена з врахунком зручності розшифрування азбуки Морзе, що є непотрібним для потреб сучаної людини. "ЙЦУКЕН" був створенний для російської мови, а отже українська версія цієї розкладки не може бути ефективною, поскільки не враховує різницю мов і відповідно частоту використання літер та їх комбінацій.

Крім того існують і альтернативні розкладки української мови, але більшість з них, такі як розкладки "ol03a" та "ol03g" [5], представляють з себе модіфікований "ЙЦУКЕН", де передставлені функціональні клавіші, які не цікавлять нас в контексті цієї роботи. Також існує україномовна

розкладка клавіатури "Шарапівка", вона була створена з розміщенням найуживаніших літери в українській мові у середньому ряду для оптимізації при сліпому методі друку [6].

Рисунок 1.3 — Схема розкладки "Шарапівка" [6]

Альтернативна розкладка "Шарапівка" враховує уживаність літер в українській мові, але не враховує найуживанішіх комбінацій букв у словах та розроблена тільки для сліпого методу друку. Тому створення більш ефективної україномовної розкладки клавіатури для різних методів друку є актуальною задачею.

1.2 Сучасні клавіатури та їх розмірність

Хоч і розкладки клавіатур і визначають розміщення клавіш на клавіатурі, але вигляд, форма клавіатури дуже відрізняються на різних девайсах.

Клавіатури можна класифікувати за різними характеристиками, включаючи тип взаємодії (цифрова чи фізична), спосіб підключення, розмір і функції. Зупинимося на двох основних типах клавіатур: фізичній і віртуальній.

Фізична клавіатура - це окремий пристрій від екрану, який використовується для вводу тексту або команд на пристрої. Це може бути клавіатура, підключена до комп'ютера через USB або бездротовий інтерфейс, або вбудована клавіатура на ноутбуці або іншому пристрої.

Фізичні клавіатури мають реальні фізичні клавіші, які натискаються для введення символів і команд. Ці клавіатури можуть мати різні розміри, розкладки та функціональні можливості в залежності від моделі та призначення.

Стандартною клавіатурою для настільних комп'ютерів вважається повнорозмірна. Вона включає в себе цифрову клавіатуру, зазвичай у крайньому правому боці дошки, групу стрілок ліворуч від неї та клавіші-модифікатори над клавішами зі стрілками [7].

Стандартною формою клавіші є об'ємна трапеція, а стандартним розміром клавіші є 18 міліметрів у довжину і ширину у верхньому квадраті та 14 міліметрів у нижньому [8].

Відстань між клавіш також є важливим фактором і стандартно складає близько 5.05 міліметрів від верхніх квадратів [8]. Ця відстань вимірюється між кутів верхніх квадратів сусідніх клавіш і допомагає забезпечити комфортну просторову організацію клавіатури. Але в подальшому будемо враховувати відстань між нижніми квадратами, що стандартно дорівнює 1.05 міліметрів.

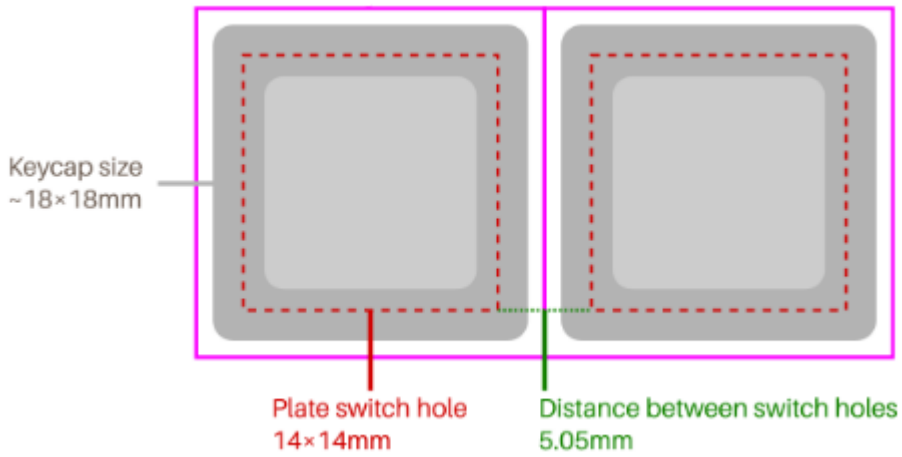


Рисунок 1.4 — Схема відстані між клавішами та розмірів клавіши [8]

Також для подальшого вимірювання дистанцій між несусідніми клавішами нам знадобиться довжина лівих клавіш Tab, Caps Lock та Shift

для вирахування відступів рядків буквених клавіш від лівого початку клавіатури. Поскільки немає стандартних розмірів цих клавіш, візьмемо за основу розміри цих клавіш на клавіатурі Razer Ornata V2, оскільки на ній буквенні клавіші відповідають стандартним розмірам. Звідси маємо довжину лівих клавіш Tab, Caps Lock та Shift у 2.6, 3.2, та 4 міліметра відповідно.

Варто відзначити, що на різних фізичних клавіатурах реальні розміри клавіш та відстань між ними можуть варіюватися в залежності від виробника та конкретної моделі клавіатури.

Віртуальна клавіатура, за визначенням , яке наведено в статті [9], характеризується відсутністю фізичного представлення. Введення тексту здійснюється за допомогою віртуальних дотикових зон, які можуть бути реалізовані за допомогою сенсорів, методів відслідковування руху пальців або сенсорних панелей [9]. Віртуальна клавіатура відображається на екрані такого пристрою, як смартфон, планшет або монітор комп’ютера. Віртуальна клавіатура може мати стандартну розкладку будь-якої мови або вона може бути спеціально розроблена для певних функцій, наприклад цифрова клавіатура чи графічна клавіатура для введення емодзі [10].

Для віртуальних клавіатур на телефонах та планшетах немає конкретного універсального стандарту, оскільки розміри та дизайн клавіатур можуть відрізнятися в залежності виробника, моделі, оперативної системи та її версії. Кожен виробник може мати свої власні варіанти віртуальних клавіатур, які відповідають їхнім дизайнерським вимогам та функціональності пристрою. Крім того, навіть на одному девайсі клавіатура може мати мати різні розміри, залежно від орієнтації пристрою в даний момент. На рисунках 1.5 та 1.6 зображено віртуальну клавіатуру на пристрої OnePlus Nord N10 5G з оперативною системою Android 11 у вертикальній та горизонтальній орієнтаціях відповідно.

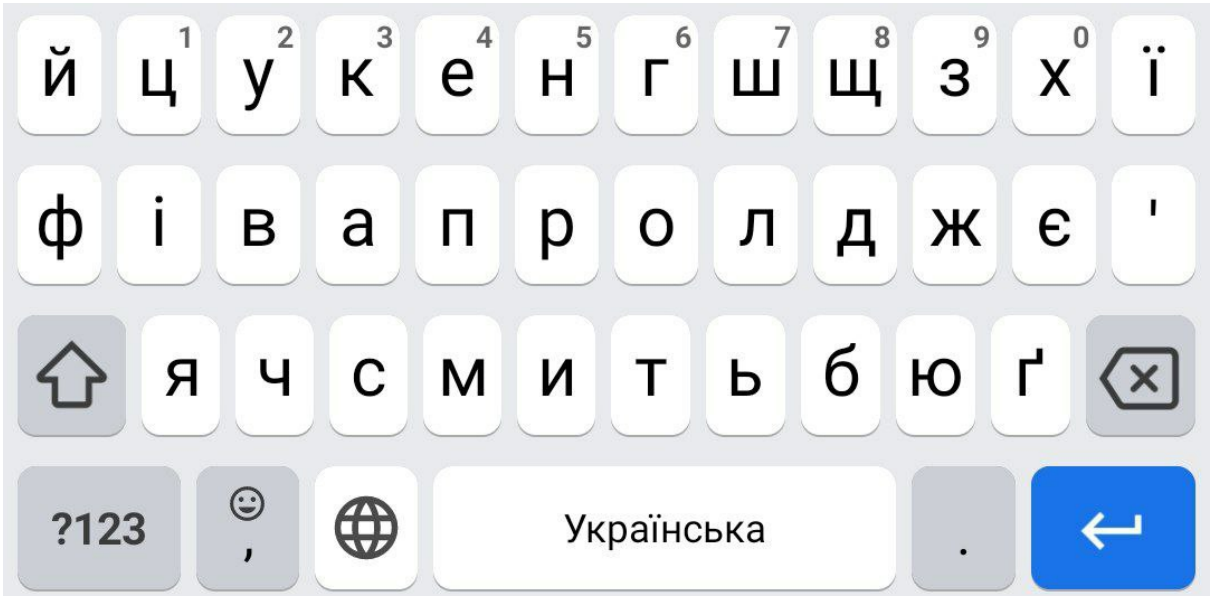


Рисунок 1.5

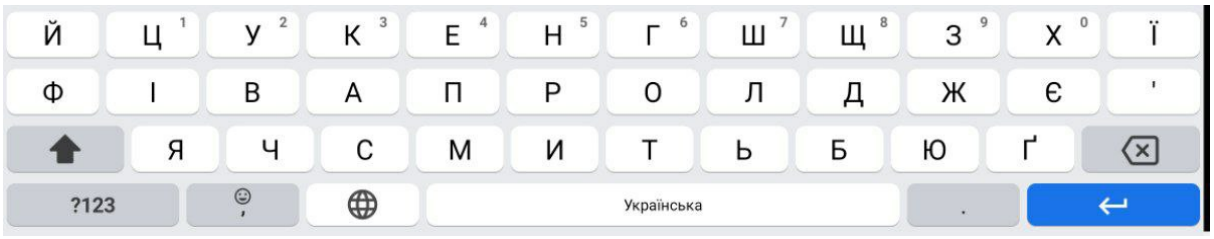


Рисунок 1.6

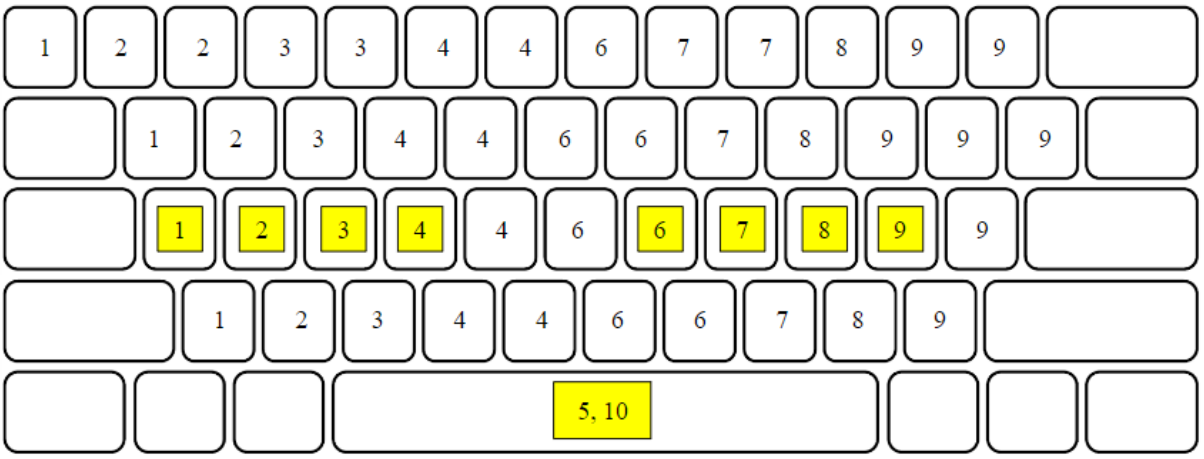
У операційній системі Android, яка є одним з лідерів на ринку мобільних пристроїв та використовується на пристроях різних виробників, віртуальна клавіатура може бути налаштована за допомогою чотирьох основних параметрів, які визначені у класі Keyboard згідно з документацією операційної системи [11]. Ці параметри включають вертикальний відступ, горизонтальний відступ, висоту клавіші та ширину клавіші. Значення за замовчуванням у цих параметрів відрізняються в залежності від пристрою та версії операційної системи, тому за основу візьмемо розміри клавіш на пристрої OnePlus Nord N10 5G з операційною системою Android 11 у вертикальній орієнтації. За допомогою зображення клавіатури можна визначити, що ширина та висота клавіш дорівнюють 42

та 60 абсолютним одиницям, а горизонтальна та вертикальна відстань між клавішами дорівнює 8 та 15 абсолютним одиницям. Ширина лівої клавіші Shift дорівнює 54 абсолютних одиниць.

1.3 Методи набору на клавіатурі

На ефективність та швидкість набору на клавіатурі впливає не тільки розміщення клавіш, але і розміщення пальців на клавіатурі та їх переміщення під час друку. Звісно на саму методику впливає і сама клавіатура, поскільки тоді як на фізичній клавіатурі можливо використовувати всі десять пальців, то на девайсах з віртуальною сенсорною клавіатурою, таких як смартфони, зазвичай це не можливо, поскільки користувач також використовує пальці щоб тримати девайс у руках. У цій роботі розглянемо найпопулярніші методи друку для фізичних та сенсорних клавіатур.

Найпопулярнішим та одним з найефективніших методом друку на фізичних клавіатурах є сліпий метод друку або ж десятипальцевий метод друку. Цей метод став популярним після перемоги Френка Едварда МакҐурріна 25-ого липня 1888 року на змаганнях з швидкості набору тексту на клавіатурі, де Френк використав сліпий метод друку [12]. Ідея методу полягає у використанні всіх десяти пальців при наборі, де вказівний, середній, безіменний, мізинець правої та лівої руки у початковому стані лежать у середньому ряду клавіатури, а великі пальці на клавіші пробілу. На рисунку 1.7 зображена схема розташування пальців на буквено-цифрових клавішах клавіатури при друку за сліпим методом. Виділені на рисунку початкові позиції пальців, які використовуються для початку набору тексту. При друку будь-яким пальцем, він одразу повертається на свою початкову позицію, якщо не відповідає за наступну букву у слові.



Ліва рука:
1 — мізинець
2 — підмізинний палець
3 — середній палець
4 — вказівний палець
5 — великий палець

Права рука:
6 — мізинець
7 — підмізинний палець
8 — середній палець
9 — вказівний палець
10 — великий палець

Рисунок 1.7

Але сліпий метод друку не є оптимальним для сенсорних пристроїв, оскільки сумарна ширина пальців зачасту більша, ніж ширина екранів смартфонів. Крім того, зазвичай люди набирають текст на смартфонах, тримаючи їх у руках, тому пальці також задіяні у процесі утримання пристрою і не можуть бути використані для друку на клавіатурі. Тому для таких випадків існує метод набору на сенсорній клавіатурі за допомогою двох великих пальців.

Для формалізації цього методу набору за допомогою двох великих пальців поділимо сенсорну клавіатуру посередині по вертикалі на дві частини, в яких відповідно розміщуються два великі пальці. На рисунку 1.8 зображено схему розміщення пальців при друку на сенсорній клавіатурі за допомогою двох великих пальців.

Рисунок 1.8

Також для вводу даних на сенсорних девайсах використовується лише один палець. Звісно, у такому випадку всі клавіші клавіатури відносяться до цього пальця, а початкова позиція є клавіша, що знаходиться по середині середнього рядка клавіатури. На рисунку 1.9 зображено схему розміщення пальців при друку на сенсорній клавіатурі за допомогою одного пальця.

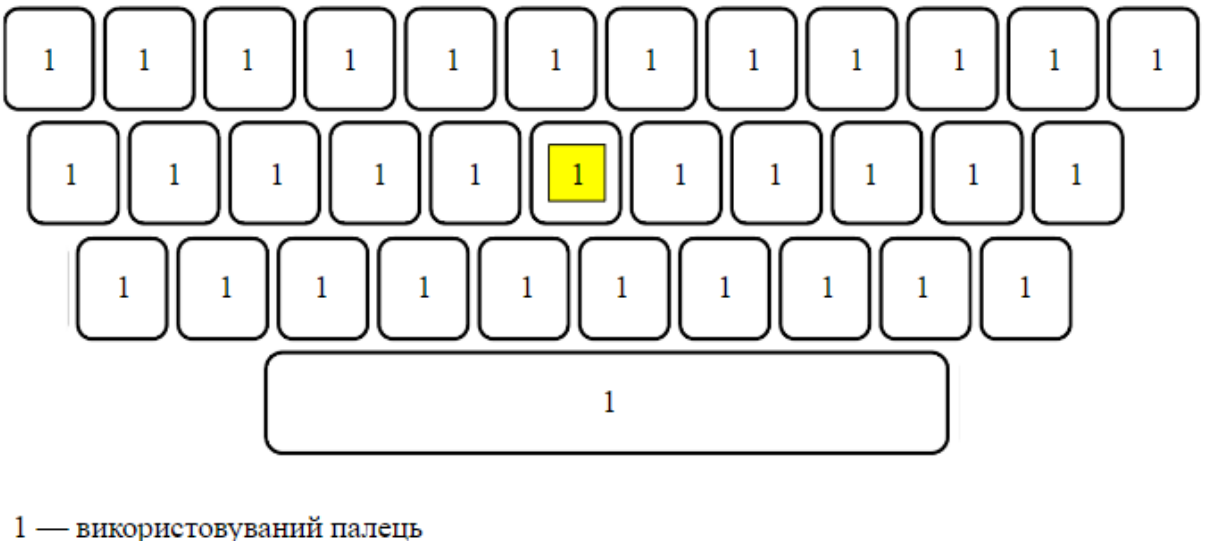


Рисунок 1.9

Зрештою, має три методи друку, один для фізичної клавіатури – сліпий метода набору, та два методи для сенсорної клавіатури – метод друку двома пальцями та одним пальцем.

1.4 Аналіз сучаної української мови

Аналіз сучасної української мови відіграє важливу роль у подальшому розміщенні букв на розкладці клавіатури. У цьому контексті, одним із ключових аспектів дослідження є вивчення частоти використання букв української мови. Для цього у цьому підрозділі проаналізуємо слова української мови та сучасні україномовні тексти. Це дозволить нам отримати більш повну картину щодо вживання букв у сучасній українській мові та використовувати ці дані для подальшого аналізу та розробки клавіатурної розкладки.

У одному з найбільших сучасних словників української мови [13] налічується 164837 слів. На рисунку 1.10 таблицю, в якій вказено скільки слів у словнику починаються з певної букви, впорядковані за спаданням кількості слів. На рисунку 1.11 зображено гістораму на основі даних з таблиці, де букви відображені в алфавітному порядку.. Як видно з цієї гістограми, з деяких букв починається зовсім незначна кількість слів, наприклад, з наступних букв починається лише 0.3% слів у словнику: ю, ґ, є, и, ї, й, ь. Найчастіше слова починаються з букв: п, в, с, н, р, о, к, б, т, м, г. Ці букви можуть розглядатися для розміщення на початкових позиціях у клавіатурі, але такий аналіз не є точним, поскільки не всі слова однаково часто вживаються, а частота вживаності букви немає прямої кореляції з частотою вживаності на початку слова.

П	34303	У	4989	Е	1421
В	16192	А	4233	Я	672
С	13930	Ш	3376	Щ	551
Н	12219	З	3246	Ж	298
Р	10273	Л	3245	Ю	187
О	8883	Х	2873	Ґ	168
К	8471	Ф	2737	Є	89
Б	6696	Ч	2706	Й	16
Т	6492	Д	2357	И	10
М	6040	Ц	1513	Ї	10
Г	5146	І	1430	Ь	1

Рисунок 1.10

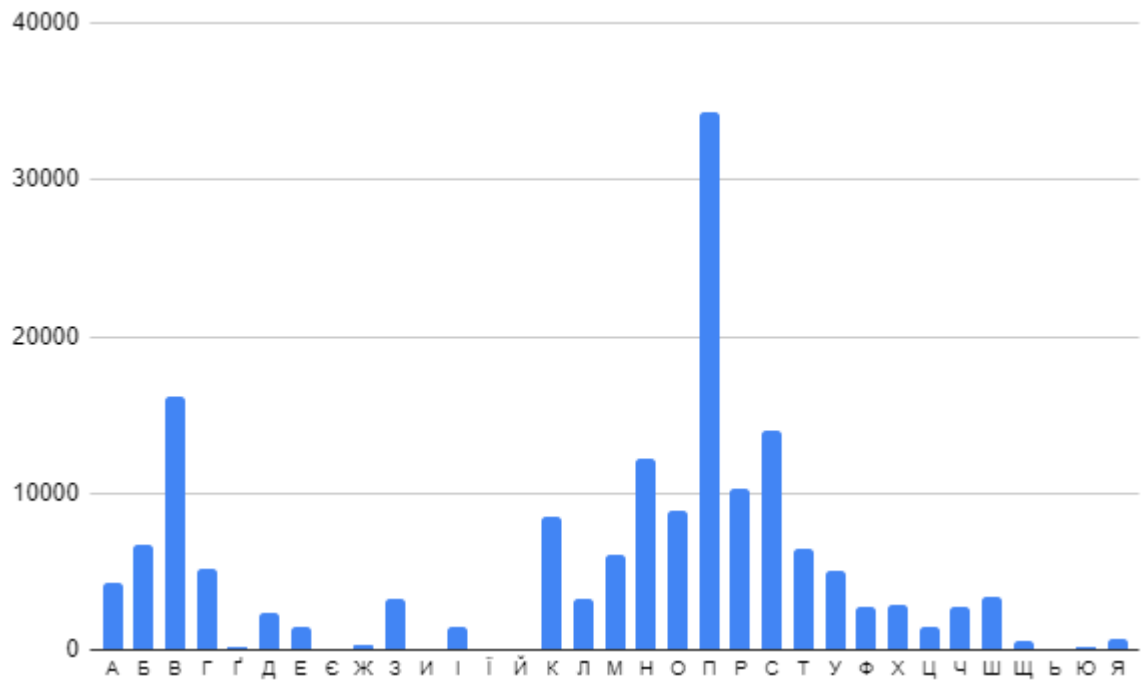


Рисунок 1.11

Більш точним показником у цьому випадку буде частота використання букв в україномовних текстах в цілому. Візьмемо дані зі статті щодо частотного аналізу використання букв української мови [14], поскільки у цьому досліджені було проаналізовано сучасні технічні тексти, художню літературу та гуманітарні тексти. Маємо дані зі статті [14], що представлена на рисунку 1.12 у вигляді таблиці частости використання

букв української мови, впорядкованих за спаданням частоти, частоти та на рисунку 1.13 у вигляді гістограми даних з таблиці, де букви відображені в алфавітному порядку. Найчастіше використовуються букви: о, а, н, и, і, в, т, е, р, с, л.

О	0,0942	р	0,0448	я	0,0248	ж	0,0093
А	0,0807	с	0,0424	з	0,0232	ю	0,0093
Н	0,0681	л	0,0369	б	0,0177	ц	0,0083
И	0,0626	к	0,0354	ь	0,0177	ш	0,0076
І	0,0575	д	0,0338	г	0,0155	ї	0,0065
В	0,0535	у	0,0336	ч	0,0141	є	0,0061
Т	0,0535	м	0,0303	й	0,0138	щ	0,0056
Е	0,0495	п	0,0290	х	0,0119	ф	0,0028

Рис 1.12

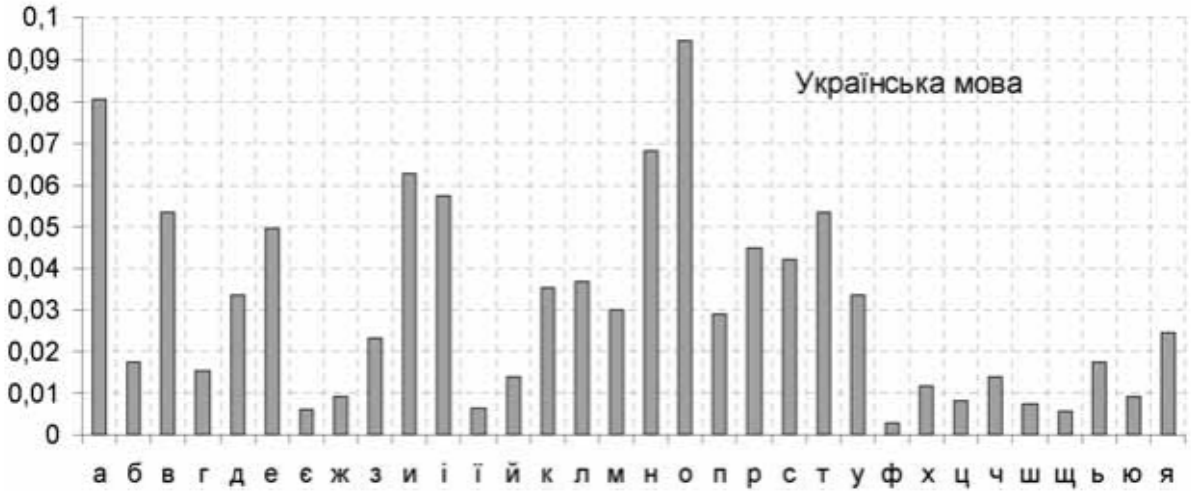


Рис 1.13

Букви н, в, т, р та с зустрічаються як і в списку букв, з яких найбільша кількість слів починається у словнику, так і в списку букв, що використовуються найчастіше у україномовних текстах. З обох списків видно, що хоч не так багато слів починаються з голосних букв, але частота використання деяких з них є найбільшою.

З даних цього підрозділу утворимо список букв, що будемо розміщати на початкових позиціях на розкладках клавіатур. Більш показовами для мети цієї роботи є дані з частотного аналізу частотного аналізу використання букв української мови, тому за основу візьмемо їх. Але враховуючи те, що при початку друку слова усі пальці знаходяться на початкових позиціях, візьмемо до уваги, що частота використання букв л і к відрізняється лише у 1.04 рази, в той час як на букв к починається у більше ніж 2.6 разів кількість слів. Тому маємо наступним список: о, а, н, и, і, в, т, е, р, с, к.

1.5 Висновок до розділу 1

У розділі 1 розглянуто існуючі розкладки клавіатур та їх історію. Встановлено, що деякі з популярних розкладок не є ефективними через їх застарілість або невідповідність до специфіки мови, на яку вони призначені. Були також розглянуті альтернативні розкладки, проте вони також не повністю враховують всю специфіку мов та різних методів набору.

Крім того, було розглянуто сучасні клавіатури з урахуванням фізичних та віртуальних варіантів. Були визначені параметри розміру для обох видів клавіатури, які в подальшому будуть впливати на оцінку її ефективності розкладки.

Також були розглянуті найпоширеніші методи набору, їх поява та використання для різних типів клавіатур, початкове розташування пальців при використанні цих методів та встановлено відповідність між кожною клавішею та пальцем, який зазвичай на цю клавішу, для кожного методу друку.

Для досягнення ефективного розташування символів на початкових позиціях пальців було проведено аналіз слів української мови та сучасних

україномовних текстів. Як результат, був створений список літер у порядку спадання частоти використання з урахуванням кількості слів, які починаються з даної літери.

2 ГЕНЕТИЧНИЙ АЛГОРИТМ

2.1 Теорія генетичного алгоритму

Людство багато разів брало натхнення з природи, намагаючись використовувати її механізми та принципи для вирішення складних проблем. Одним з найцікавіших прикладів такого впливу є генетичні алгоритми, які базуються на еволюційних принципах та інспіровані природним відбором. Генетичні алгоритми є потужним інструментом для розв'язання оптимізаційних задач, а їх успіх може бути пояснений їхньою здатністю виробляти розв'язки, що надаються природою.

Головна ідея генетичних алгоритмів полягає в емуляції основного механізму еволюції — природного відбору, де кращі адаптовані організми мають більшу ймовірність виживання та передачі своїх генетичних характеристик наступним поколінням [15]. Аналогічно, у генетичних алгоритмах, розв'язки представляються у вигляді "індивідів", які мають свої генетичні властивості або параметри.

Однією з ключових переваг генетичних алгоритмів є їхнє можливості працювати зі складними, недиференційованими та мультимодальними функціями, де інші традиційні алгоритми можуть застрягати в локальних оптимумах. Генетичні алгоритми дозволяють здійснювати пошук в просторі розв'язків, сприяючи знаходженню оптимальних або наближених до оптимальних розв'язків.

Генетичний алгоритм ділиться на етапи. На рисунку 2.1 зображена діаграма зі статті [17], на які зображено наступні етапи алгоритму: ініціалізація, оцінка придатності, відбір, схрещування, мутація. Як ще видно на діаграмі, після етапу оцінки придатності йде перевірка чи задовільняє одна з хромосом популяції умові кінця алгоритму. Ця умова є дуже важлива, оскільки специфіка використання алгоритму полягає в

тому, що зазвичай оптимальне вирішення задачі не відоме, тому аллгоритм не знає де зупинитись. Такою умовою може кількість ітерацій, під час яких пристосованість індивідумів перестає помітно збільшуватися [15]. Також умовою зупинки може бути і обмежена кількість ітерацій, але в такому випадку важко гарантувати, що алгоритм наблизився оптимуму, чи локального оптимуму.

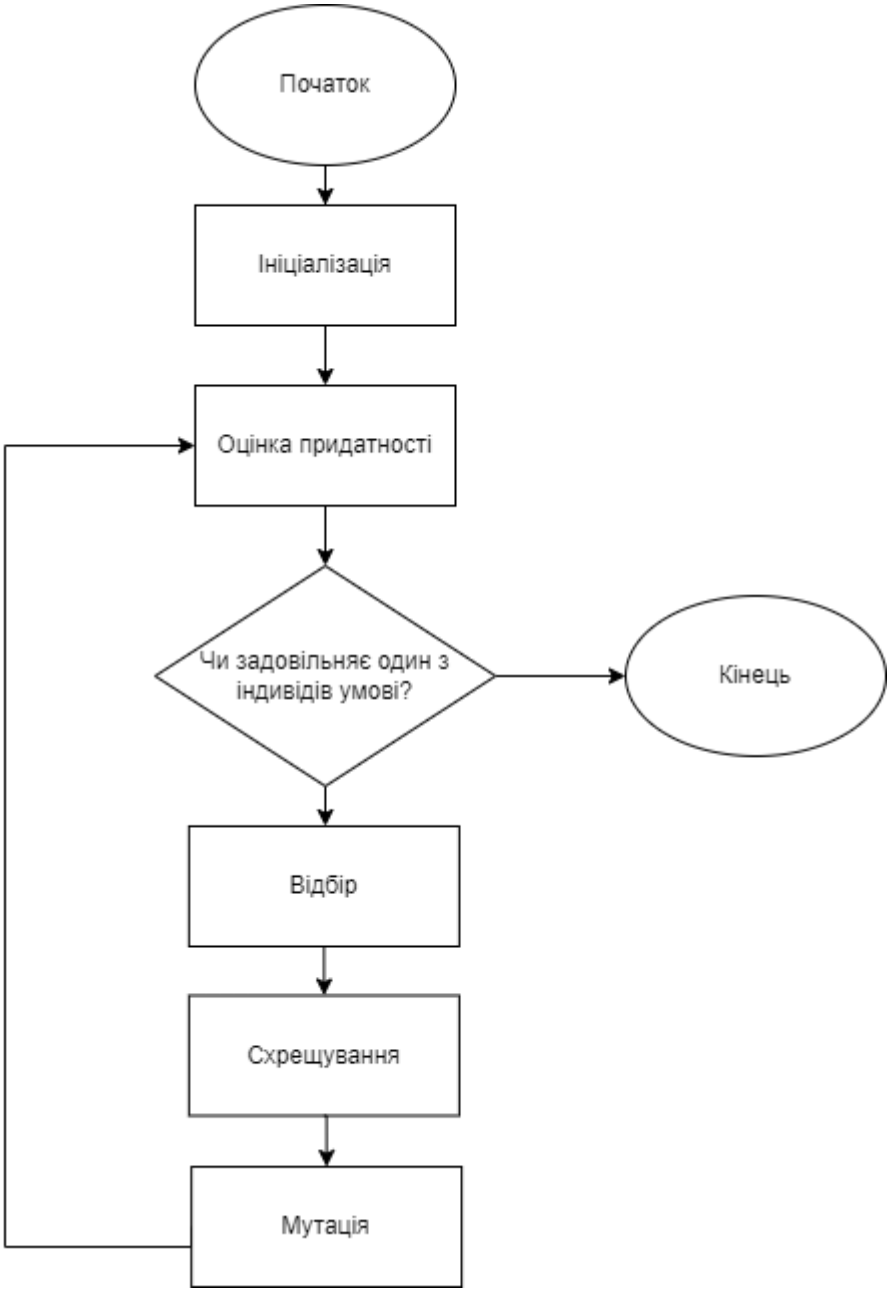


Рисунок 2.1

Починається генетичний алгоритм з популяції хромосом, при чому зазвичай випадкових [16]. Хромосоми складаються з певного набору генів. Гени — це зміні, які визначаються відповідно до задачі, яку вирішує генетичний алгоритм. Зазвичай ген представляють у вигляді однієї з цифр бінарного вектору, наприклад 1100010101, а сам вектор є хромосомою [16]. Відповідно у такому випадку ген може набувати одне з двох значень — 0 або 1.

Наступним етапом є оцінка придатності, він є одним з ключових етапів в генетичному алгоритмі. На цьому етапі кожному індивіду в популяції призначається значення пристосованості, яке відображає якість або ефективність цього розв'язку.

Оцінка придатності зазвичай здійснюється шляхом виконання функції оцінки. Функція оцінки приймає індивіда як вхідний параметр і повертає числове значення, яке відображає якість чи ефективність цього індивіда [15].

Мета оцінки придатності полягає в тому, щоб призначити вищі значення пристосованості розв'язкам, які вирішують поставлену задачу краще або наближаються до оптимального розв'язку. Часто функція оцінки базується на цілі або метриці, яку необхідно максимізувати чи мінімізувати в процесі пошуку розв'язку.

На базі оцінки придатності проводиться етап відбору. Частка індивідів, що пройде на наступний етап, визначається параметром генетичного алгоритму [15]. Відповідно на цьому етапі вибираються найкращі за оцінкою пристосованості індивіди у кількості розміру популяції помноженої на частку індивідів, що проходе відбір.

Індивіди, що пройшли відбір стають батьками наступного покоління на етапі схрещування. Етап схрещування є одним з ключових етапів генетичного алгоритму. На цьому етапі відбувається комбінування

генетичного матеріалу (хромосом) індивідів з метою створення нових потомків. Схрещування відбувається на основі обміну генетичною інформацією між батьківськими особинами.

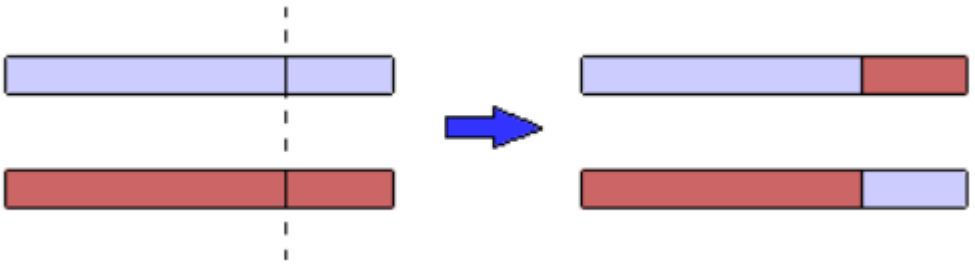


Рисунок 2.2 — Схема схрещування [15]

Процес схрещування полягає у виборі певної точки або ділянки на хромосомі, яка служить місцем поділу генетичної інформації. Є різні стратегії вибору цієї точки, найпопулярніші стратегії описуються у статті [18]:

Uniform scanning (рівномірне сканування): Це стратегія, при якій точка або ділянка для схрещування вибирається рівномірно випадковим чином. У цьому випадку, кожна позиція на хромосомі має однакову ймовірність бути обраною для схрещування.

Occurrence-based scanning (сканування на основі частоти): Ця стратегія враховує частоту входження певних генетичних елементів на хромосомах. Частіші елементи мають більшу ймовірність бути обраними для схрещування, що сприяє збереженню популяції популярних генетичних властивостей.

Fitness-based scanning (сканування на основі придатності): У цій стратегії точка або ділянка для схрещування вибирається на основі оцінки придатності індивіда. Індивіди з вищою придатністю мають більшу ймовірність бути обраними для схрещування, що дозволяє передавати корисні генетичні властивості наступному поколінню.

Використання різних стратегій схрещування дозволяє експериментувати з різними підходами до комбінації генетичного матеріалу і може мати вплив на різноманітність та якість потомства.

Останім етапом перед повернення нової популяції до етапу оцінки придатності є мутацію. Мутація впроваджує випадкові зміни в генетичну інформацію індивідів, що дозволяє розширити пошуковий простір і зберегти різноманітність у популяції. Мутація може виконуватись різними способами, залежно від природи задачі і налаштувань генетичного алгоритму.

Під час мутації вибирається певна кількість генів у хромосомі, і їх значення змінюється випадковим чином. Це дозволяє внести невеликі, але важливі зміни в генетичну конфігурацію індивіда. Мутація може бути реалізована шляхом зміни одного або декількох бітів, вставки або видалення генетичних елементів, заміни певних фрагментів хромосоми або інших методів залежно від задачі [19].

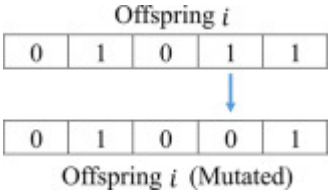


Рисунок 2.3 — Схема мутації [19]

Основною метою мутації є внесення випадковості і різноманістність у популяцію, щоб уникнути застрягання в локальних екстремумах та сприяти подальшому дослідженю пошукового простору. Мутація може допомогти знаходити нові, неочікувані рішення, а також пристосовувати популяцію до змінних умов і задач.

Важливим аспектом мутації є контрольований рівень і ймовірність впровадження змін. Занадто низький рівень мутації може призвести до

передчасної збіжності і втрати різноманітності, тоді як занадто високий рівень мутації може спричинити дестабілізацію алгоритму і призвести до втрати корисних рішень [19]. Тому важливо знаходити баланс між інтенсивністю мутації і збереженням різноманітності популяції.

Фінальний етап генетичного алгоритму включає зупинку процесу еволюції після виконання заданих умов і вибір найкращого рішення, який алгоритм знайшов під час своєї роботи.

2.2 Генетичний алгоритм для створення ефективної україномовної розкладки клавіатури

У контексті основної задачі даної дипломної роботи, геном генетичного алгоритму є кожна окрема клавіша розкладки. Відповідно, набір генів у кількості букв в абетці створює хромосому, тобто розкладку клавіатури.

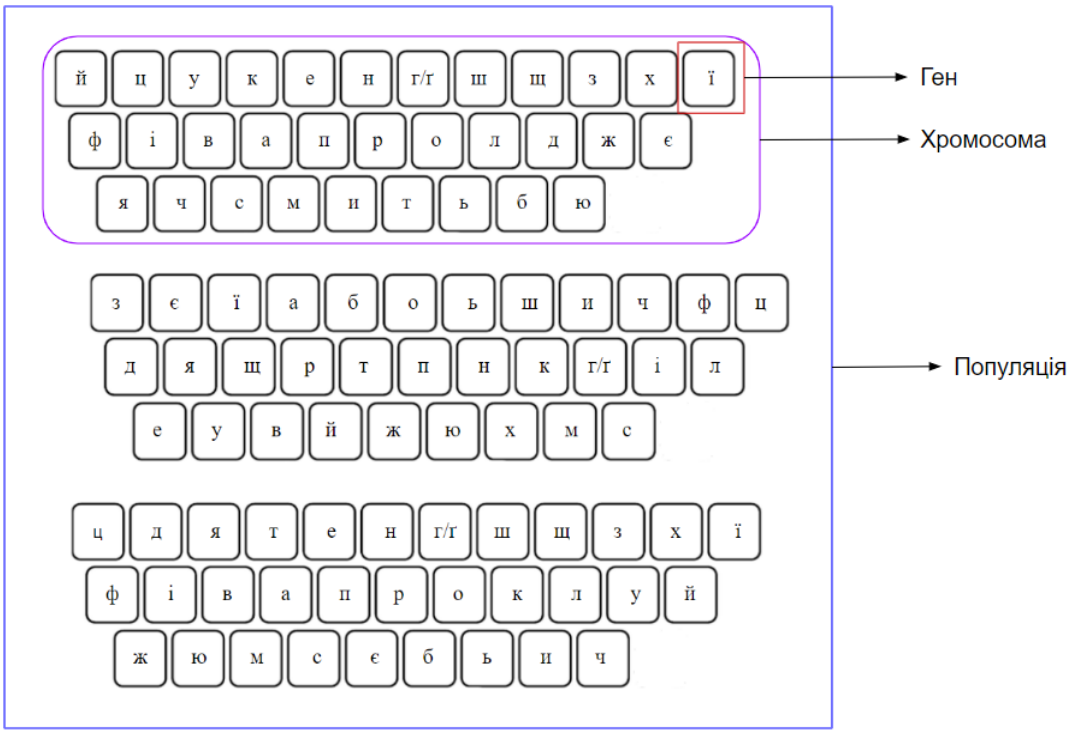


Рисунок 2.4 — Модель ГА для створення рокзладки клавіатури

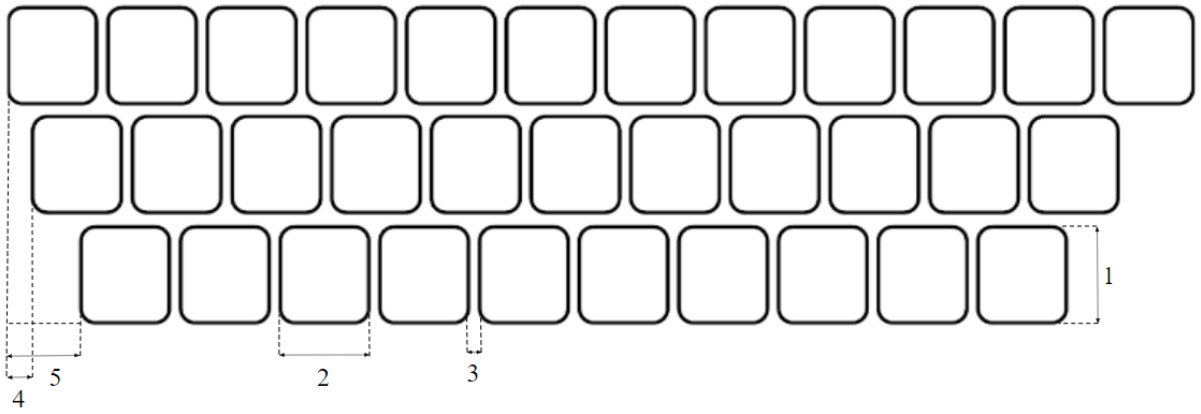
Під час етапу ініціалізації популяції необхідно створити кількість розкладок відповідній до кількості індивідів у популяції. Для більш ефективної початкової популяції необхідно провести попередній аналіз, щоб знайти найбільш часто використовувані символи у заданих текстових даних. Ці символи необхідно розмістити на початкові позиції пальців для методу набору, для якого буде створюватися ефективна розкладка. Всі інші символи розміщуються випадково.

Для оцінка придатності використовується алгоритм оцінки ефективності, який вираховує загальну відстань, пройдену пальцями під час друку. Для цього алгоритм використовує задану індивідом розкладку клавіатури, текстовий набір даних та метод друку, для яких створюється ефективна розкладка.

Текстовий набір даних є необхідним для послідовного проходження кожного символу, що міститься у тексті. На кожній окремій ітерації, коли розглядається певний символ, вираховується дистанція, що має пройти палець, від свого поточного місцезнаходження до клавіші, на якій знаходиться цей символ у заданій розкладці, та додається до суми пройдених до цього відстаней. Палець обирається за допомогою методу друку, який для кожного окремого використовуваного в методі пальця зазначає координати клавіш, які він може натискати.

Початкове місце знаходження пальців зазначає метод друку. Пальці автоматично переміщаються до свого початкового місцезнаходження, якщо палець не був використан у ітерації, а відстань яку він проходить при поверненні не враховується. Це необхідно для ефективного моделювання реального друку на клавіатурі, поскільки переміщення на початкове місце знаходження є вторинною дією, яка не впливає на послідовне введення тексту під час друку. Палець використаний на ітерації зберігає своє місцезнаходження на клавіші, яку було використано.

Відстань від пальця до клавіші вираховується за допомогою заданих ширини та довжини клавіш, вертикальної та горизонтальної дистанції між клавішами та відступами рядів клавіатури від крайньої лівої точки розкладки. На рисунку 2.5 зображену схему, де зазначено всі ці метрики клавіатури.



1 — довжина клавіші
2 — ширина клавіші
3 — дистанція між клавішами
4 — відступ другого ряду
5 — відступ третього ряду

Рисунок 2.5

Враховуючи ширину та довжину клавіш, а також відстані між клавішами, можна обчислити відстань між поточним положенням пальця та цільовою клавішею, враховуючи, що палець натискає на центр клавіші. Додавання відступу рядка клавіатури або відстані від крайньої лівої точки розкладки дозволяє врахувати положення клавіші в контексті розкладки клавіатури. Функція дистанції між двома клавішами спирається на теорему піфагора та її можна виразити як:

$$f(a_{i,j}, b_{m,n}) = (|x_i - x_m| + |i - m| \cdot |d_h - w|)^2 + ((j - n) \cdot (d_v - l))^2,$$

де $a_{i,j}$ – клавіша у рядку i та стовпчику j;

$b_{m,n}$ – клавіша у рядку m та стовпчику n;

x_i, x_m – відступи рядка i, – відступ рядка m;

d_h та d_v – горизонтальна та вертикальна дистанції між клавішами;

w та l – ширина та довжина клавіш.

Загальна сума дистанцій є сумою дистанцій для друку кожного окремого символу у текстовому наборі даниї. На рисунку 2.6 зображено схему вирахування загальної дистанції для друку слова "розкладка".

f(розкладка) = f(о, р) + f(р,о) + f(ж, з) + f(а,к) + f(л, л) + f(а, а) + f(д, д) + f(а, к) + f(к, а)

Рисунок 2.6

За обчисленими загальними пройденими відстанями для кожної розкладки у популяції, вираховується значення ефективності розкладки, яка визначається як кількість символів у тектовому наборі даних, поділена на загальну відстань. Найкращі індивіди у популяції вибираються на основі найбільших значень ефективності розкладок, оскільки чим більше є це значення, тим краще є розкладка. Після чого алгоритм переходить до наступного етапу – схрещування.

Для кожного окремого схрещування з вибірки індивідів, що пройшли селекцію, обираються дві розкладки. Ймовірність вибору кожної розкладки є пропорційною до її значення ефективності, поділеної на суму значень ефективності всіх індивідів, що пройшли селекцію.

Після вибору точка схрещування обирається за стратегією рівномірного сканування, генеруються дві випадкові змінні – координати початкової клавіші та кількість символів з першої розкладки. Звісно, ця кількість символів не може бути більше або дорівнювати кількості

символів в розкладці. Після генерації у розкладку-дитину схрещування починаючи з початкової клавіші символи з першої розкладки переносяться зліва направо та зверху вниз. Якщо під час цього процесу доходить до останнього символу в останньому рядку, тоді переходимо до першого символу в першому рядку і продовжуємо до тих пір, поки не перенесемо кількість символів, що відповідає згенерованій змінній. Після цього повторюємо це з другою розкладкою, починаючи з клавіші, де закінчили, поки не заповнимо розкладку повністю. Якщо при переносі з другої розкладки натикаємось на символ, що вже є у розкладці-дитині, то ігноруємо його та рухаємося далі. В результаті отримуємо нову розкладку, яка поєднує патерни розміщення клавіш розкладок-батьків. Процес вибору батьків та їх схрещування повторюється до тих пір, поки не отримаємо кількість нових розкладок, що дорівнює кількості індивідів у популяції.

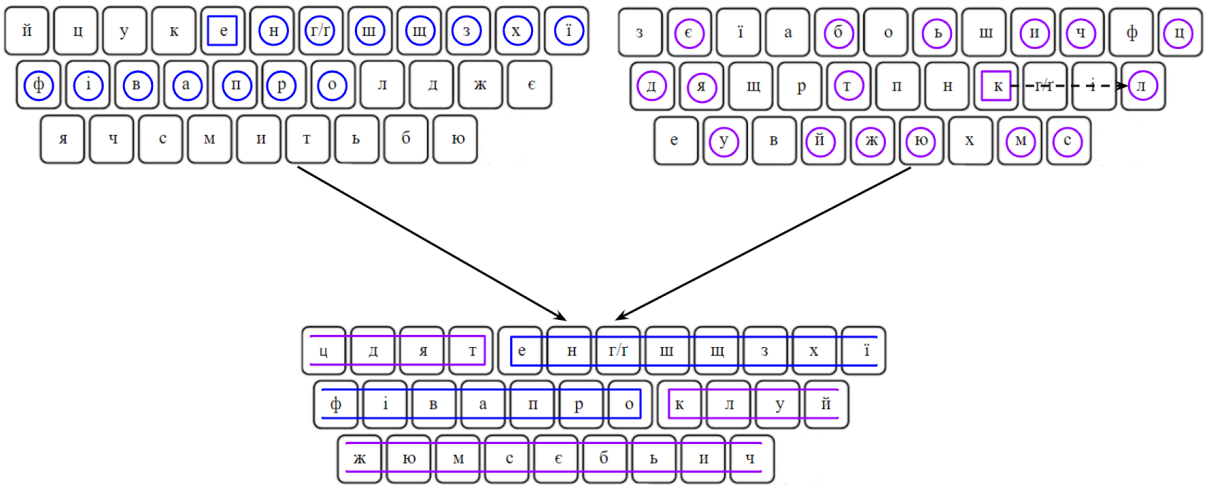


Рисунок 2.7 — Приклад схрещування розкладок

Щоб не потрапити у ситуацію, коли всі розкладки у популяції зійшлись до однакового розміщення символів на клавішах, а саме до локального максимуму показника ефективності розкладки, з певною ймовірністю відбувається мутацію. Під час мутації дві випадкові клавіші у розкладці міняються місцями.

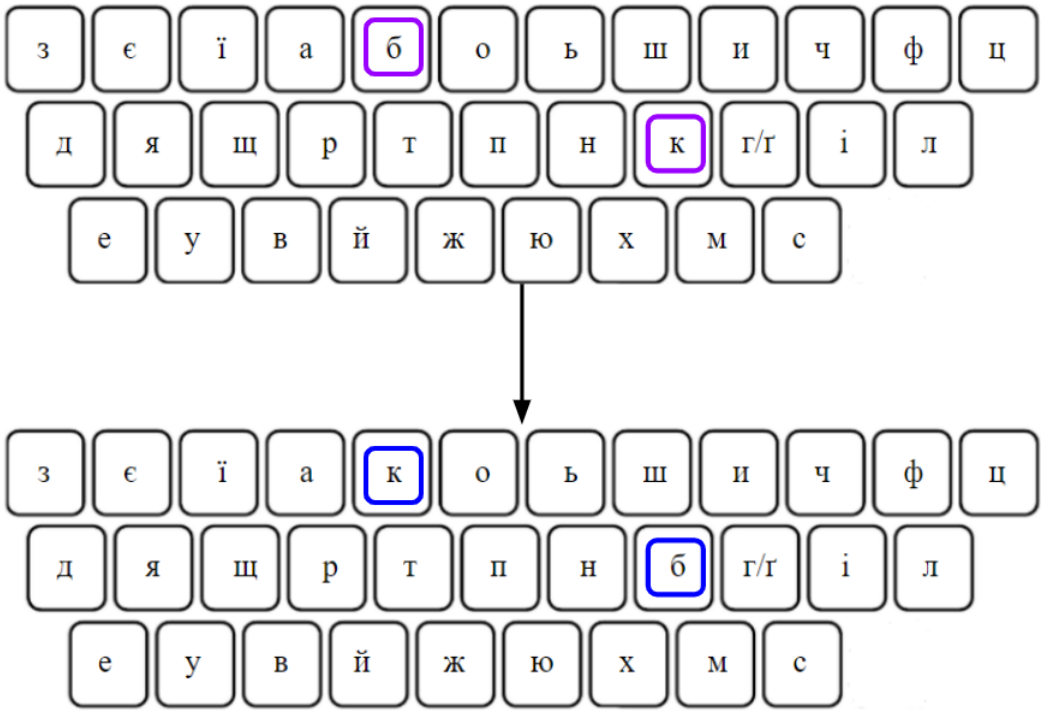


Рисунок 2.8 — Приклад мутації розкладки

Селекція, схрещування та мутація повторюються до тих пір, поки певна кількість ітерацій генетичного алгоритму не принесе більш ефективну розкладку, ніж попередній кращий результат, або ГА не дійде до заданого максимуму ітерацій.

Результуюча розкладка не тільки розміщує найбільш використовувані клавіші як можна ближче до початкових позицій пальців, але і знаходить певні патерни розміщення символів у тексті, на які звісно ще впливає метод друку.

2.3 Висновок до розділу 2

У розділі 2 було розглянуто теорію генетичного алгоритму, його ціль та етапи. Генетичний алгоритм є потужним інструментом оптимізації, який заснований на ідеях природної еволюції та генетики. Його основною

метою є знаходження оптимального розв'язку задачі шляхом імітації природного добору та процесів спадковості.

У контексті задачі створення ефективної україномовної розкладки клавіатури, була описана модель генетичного алгоритму, яка включає визначення розкладки клавіатури як хромоси, позиція окремих символів як генів. Було визначено функцію оцінки придатності на основі дистанції, що проходять пальці при друку на певній розкладці клавіатури, враховуючи заданий метод друку, що дає можливість створити ефективну розкладку для різних видів клавіатур та девайсів. Також для оцінки використовується заданий набір текстових даних, що робить алгоритм гнучким та дає можливість використовувати його для вирішення задачі створення ефективної розкладки для інших мов чи навіть окремих спеціальностей. Мутація у вигляді перестановки двох клавіш допомагає не впасти у локальний екстремум та знайти найбільш ефективну розкладку.

В цілому, генетичний алгоритм є потужним і гнучким методом оптимізації, який знаходить застосування в різних областях, включаючи створення ефективних розкладок клавіатур.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Обґрунтування вибору мови та платформи програмування

Для програмної реалізації було обрану мову програмування Python 3 та середовище Jupyter Notebook.

Мова програмування Python є лідером серед мов програмування, які використовуються для обробки даних. Вона пропонує широкий вибір бібліотек, призначених для обробки та візуалізації даних. У випадку даної цієї роботи, ці інструменти допоможуть здійснити очищення даних та ефективний аналіз результатів програми. Крім того, компактність та простота синтаксису Python та підтримка декількох парадигм програмування сприяють швидкій розробці програми.

Середовище Jupyter Notebook є потужним інструментом для поетапного виконання коду, що сприяє прискоренню роботи з даними. Одним з головних переваг цього середовища є можливість виконувати кожен крок довгого процесу обробки даних поетапно, без необхідності виконувати всю програму з початку кожного разу. Це значно збільшує продуктивність розробника, оскільки він може вносити зміни та перевіряти їх ефект безпосередньо у середовищі. Середовище Jupyter Notebook є особливо популярним у сфері аналізу даних та машинного навчання. Це через те, що етапи очищення, обробки та аналізу даних можуть бути виконані окремо в окремих "клітинках" ноутбуку. Це дозволяє аналізувати результати кожного кроку, вносити зміни та експериментувати з даними на льоту, зберігаючи всі проміжні результати. Крім того, Jupyter Notebook підтримує розмітку Markdown, що дозволяє вставляти пояснювальні коментарі, графіки та інші елементи документації безпосередньо в ноутбук, що полегшує розуміння та використання результатів аналізу даних.

Також для розробки програмного продукту були використані наступні бібліотеки:

- ThreadPool для паралельного обчислення оцінки ефективності кожної клавіатури у популяції;
- Matplotlib для візуального відображення отриманих результатів за допомогою графіків та гістограм;

3.2 Підбір набору даних

У цьому підрозділі описано процес підбору набору даних для створення ефективної україномовної розкладки клавіатури з використанням генетичного алгоритму. Враховуючи специфіку української мови та потреби користувачів, я використав україномовні статті новин як ключовий джерело даних для цього дослідження.

Одним із головних критеріїв підбору набору даних є їх репрезентативність. Для досягнення цілі створення ефективної україномовної розкладки клавіатури, було важливо використовувати текстові дані, що відображають реальне використання мови у повсякденному житті. З цією метою, в якості основного джерела даних були вибрані україномовні статті новин.

Використання україномовних статей новин має декілька переваг. По-перше, ці дані надають реалістичне відображення комунікації україномовних користувачів. Статті новин охоплюють широкий спектр тематик, включаючи політику, економіку, культуру, спорт та інші, що дає можливість врахувати різноманітність слів і фраз, що використовуються у повсякденному житті.

По-друге, україномовні статті новин допомагають врахувати мовну специфіку сучасної української мови. Врахування цих особливостей у наборі даних допомагає створити оптимізовану розкладку, яка забезпечує зручність та ефективність набору тексту українською мовою.

Самі дані були взяті з готового набору даних [20]. Цей набір даних представляє собою підбірку статей з різних сфер, таких як політика, бізнес, спорт, технології тощо. Використання такого набору даних надає можливість аналізувати актуальну інформацію у різних галузях на основі текстових даних.

Перед використанням для навчання генетичного алгоритму дані пройшли попередню обробку. Ця обробка включала видалення зайвих символів та форматування тексту для полегшення подальшої обробки.

3.3 Реалізація генетичного алгоритму для знайдення ефективної україномовної розкладки клавіатури

Програмна реалізація складається з трьох класів: KeyBoard, TypingMethod та GeneticAlgorithm.

3.3.1 Клас KeyBoard

Об’єкти класу KeyBoard складають популяцію генетичного алгоритму. При створенні цих об’єктів необхідно задати певні параметри.

Параметр "alphabet_string" має тип "string" і використовується для вказівки всіх символів алфавіту, з яких складається розкладка клавіатури.

Параметр "layout_sizes" має тип "list" та містить елементи типу "int", які вказують довжину кожного рядка. Сума елементів "layout_sizes" має відповідати кількості символі у "alphabet_string", інакше програма видасть помилку.

Параметри "button_width", "button_height", "buttons_distance_vertical", "buttons_distance_horizontal", "indents" мають відповідні значення за замовчуванням: 1.8, 1.8, 0.105, 0.105, [0, 0.6, 1.4]. Ці значення відповідають стандартним розмірам фізичної клавіатури з розділу 1. Кожен з параметрів відповідає за певну розмірну характеристику клавіатури, "button_width" та "button_height" визначають ширину та довжину клавіши, "buttons_distance_vertical" і "buttons_distance_horizontal" визначають вертикальну та горизонтальну відстань між клавішами, а параметр indents визначає відступи кожного ряду від крайньої лівої точки розкладки клавіатури.

За замовучанням при створені об’єкту класу за заданими параметрами створюється словник (структура мови Python, яка реалізує структуру даних хеш-таблиця) з відстанями між клавішами для подальшої оцінки ефективності клавіатури та розкладка клавіатури з випадковим розміщенням клавіш.

Розкладка клавіатури є масивом з підмасивами з кількістю елементів зазначеними у параметрі "layout_sizes". Елементи підмасиву є символами з абетки, і відповідно кожен символ з абетки зустрічається у розкладці лише один раз. За допомогою цього підходу можна отримувати символ клавіши за її координатами. Наприклад, координати "0,3" будуть означати клавішу у першому рядку та четвертому стовпці, бо нумерація починається з нуля. Це використовується у оцінки ефективності при наборі певним методом у класі TypingMethod.

3.3.2 Клас TypingMethod

Об’єкти класу TypingMethod використовуються для визначення функції оцінки ефективності при наборі певним методом. За допомогою параметрів "init_fingers_position" та "finger_buttons" визначається метод

набору, за замовченням це сліпий метод.

Параметр "init_fingers_position" визначає початкові позиції пальців у вигляді матриці розміром 2 на кількість пальців, які використовуються у певному методі. Кожен рядок складається з двох чисел, що визначають початкові позиції пальця за номером відповідним до номера рядка. Наприклад, значення "[[1, 3], [1, 7]]" буде означати, що в заданому методі друку перший палець на початковій позиції знаходиться на клавіші за координатами "1, 3".

Параметр "finger_buttons" визначає координати клавіш, які відносяться до кожного пальця. Наприклад, значення "[[0,0], [1,0]], [[0,1], [1,1]]]" для параметру буде означати, що для перший палец може натискати клавіші за кординатами "0,0" та "1,0".

Також можливо визначити метод друку за допомогою параметру "type", який може приймати одне з трьох значень: "ten_fingers", "two_fingers" та "one_finger". Ці значення відповідно зазначають сліпий метод друку, метод друку двома пальцями та метод друку одним пальцем.

3.3.3 Клас GeneticAlgorithm

Клас GeneticAlgorithm реалізує генетичний алгоритм для створення ефективної україномовної розкладки клавіатури, що було описано у підрозділі 2.2.

У об’єктів цього класу є наступні параметри, що використовуються для оцінки розкладок: "typing_method", "init_keyboard", "data_set". Параметри "typing_method", "init_keyboard" є об’єктами описаних вище класів, а саме TypingMethod та KeyBoard. За допомогою цих параметрів визначається метод друку, для якого буде оцінюватися ефективність розкладки, та клавіатуру, розміри якої будуть використовуватися для оцінки ефективність розкладки та абетку з якої будуть створюваться

розкладки у початковій популяції. Набір даних визначається за допомогою параметру "data_set".

Параметри "probabilty_of_mutations", "amount_of_mutations", "crossover_probability", "population_size" визначають гіперпараметри генетичного алгоритму, що впливають на швидкість навчання і сходимість алгоритму. Визначення оптимальних значень для цих параметрів є дуже важливим для ефективної роботу алгоримту. Параметр "population_size" зазначає кількість хромосом у популяції, а "selection_size" кількість з них, що проходять етап селекції.

Параметр "crossover_probability" відповідає за ймовірність схрещування пари. Цей параметр є необхідним для збереження різноманітності популяції.

Для меньшої ймовірності потрапляння у локальний екстремум було створено параметр "amount_of_mutations", що зазначає кількість мутацій, що може відбутися для кожного нового індивіда, а "probabilty_of_mutations" визначає ймовірність кожної з цих мутацій.

Для зупинки алгоритму використовуються параметри "max_amount_of_populations" та "max_populations_without_new_best". Перший відповідає за максимальну допустиму кількість ітерацій алгоримту, а другий а максимальну допустиму кількість ітерацій алгоримту без поліпшення показника кращого результату.

3.4 Результати роботи алгоритму

Кінцевим результатом роботи є розкладка клавіатури створена для заданого методу друку. Створено розкладки для сліпого методу друку, методу двох пальців та методу одного пальця. Для аналізу будемо використовувати показник нормалізованої ефективності клавіатури.

Створені клавіатури будемо порівнювати з розкладками "ЙЦУКЕН" та "Шарапівка".

На рисунках 3.1, 3.2 та 3.3 зображено створені алгоритмом розкладки клавіатур для сліпого методу друку, методу двох пальців та методу одного пальця відповідно.

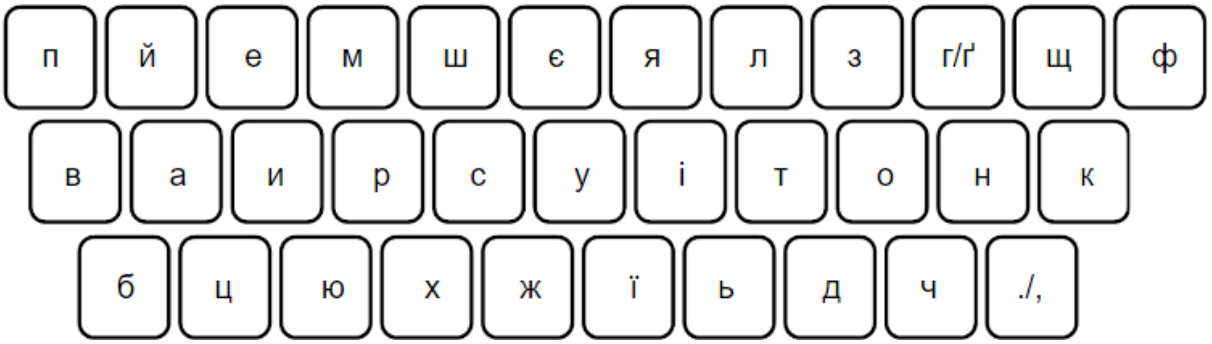


Рисунок 3.1

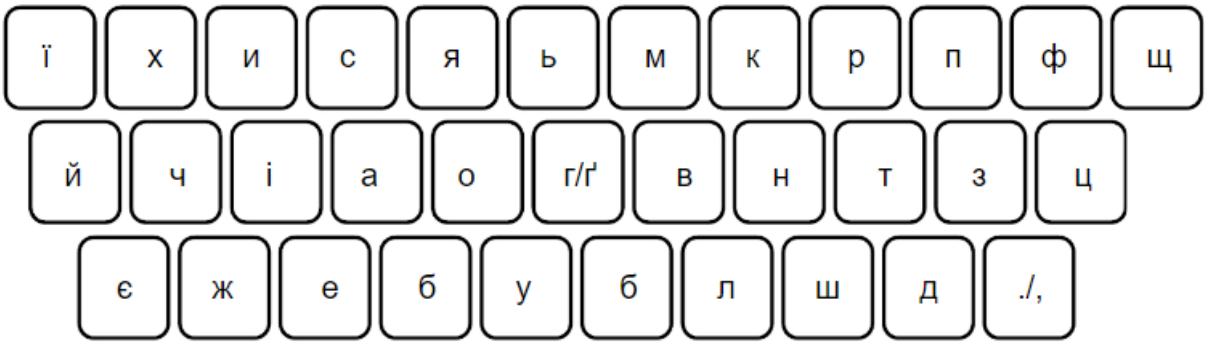


Рисунок 3.2

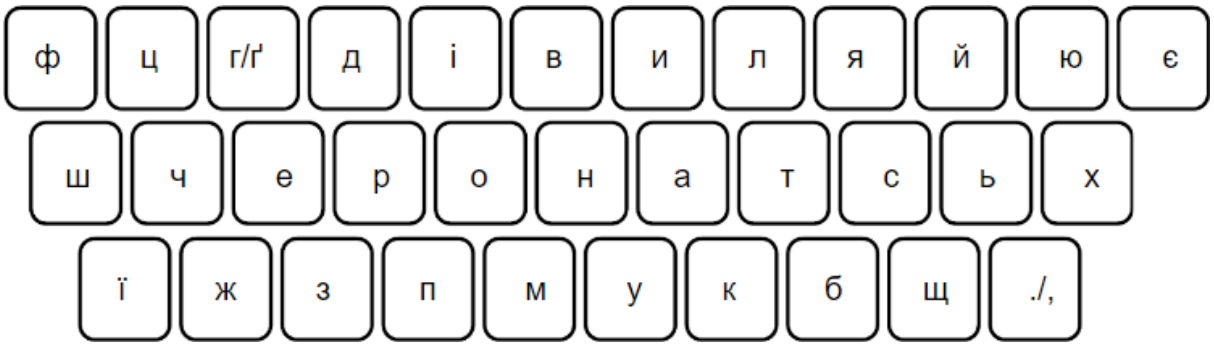


Рисунок 3.3

На рисунках 3.4, 3.5 та 3.6 зображено графіки зміни найкращого показнику нормалізованої ефективності розкладки клавіатури до кожної популяції відповідно для сліпого методу друку, методу двох пальців та методу одного пальця. З графіків видно, що алгоритм є сходимим. Метод одного пальця сходиться найдовже, за 180 ітерацій. Сліпий метод друку та метод двох пальців зійшлися швидше, за 60 та 80 ітерацій відповідно.

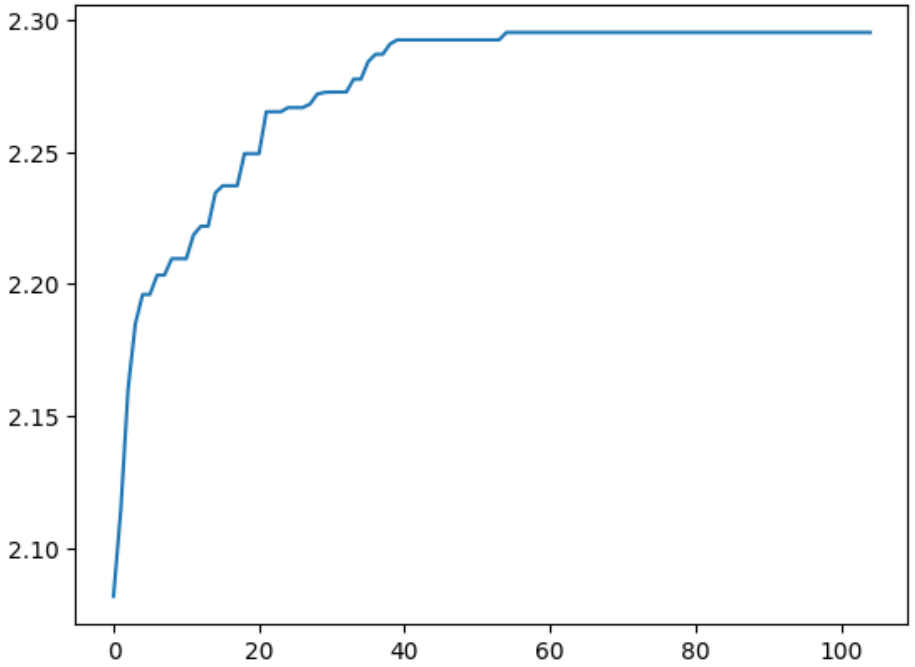


Рисунок 3.4

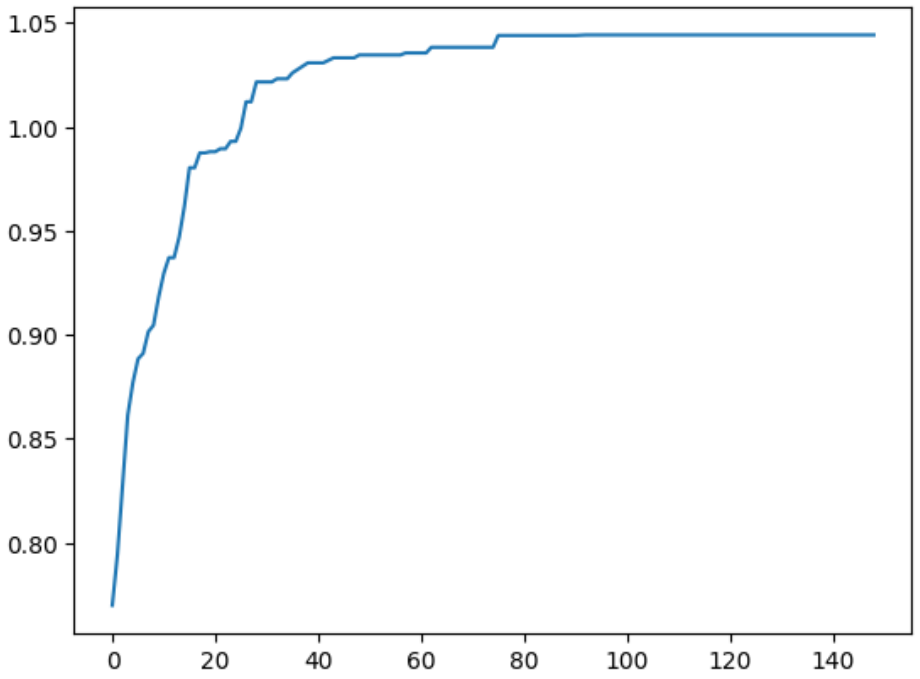


Рисунок 3.5

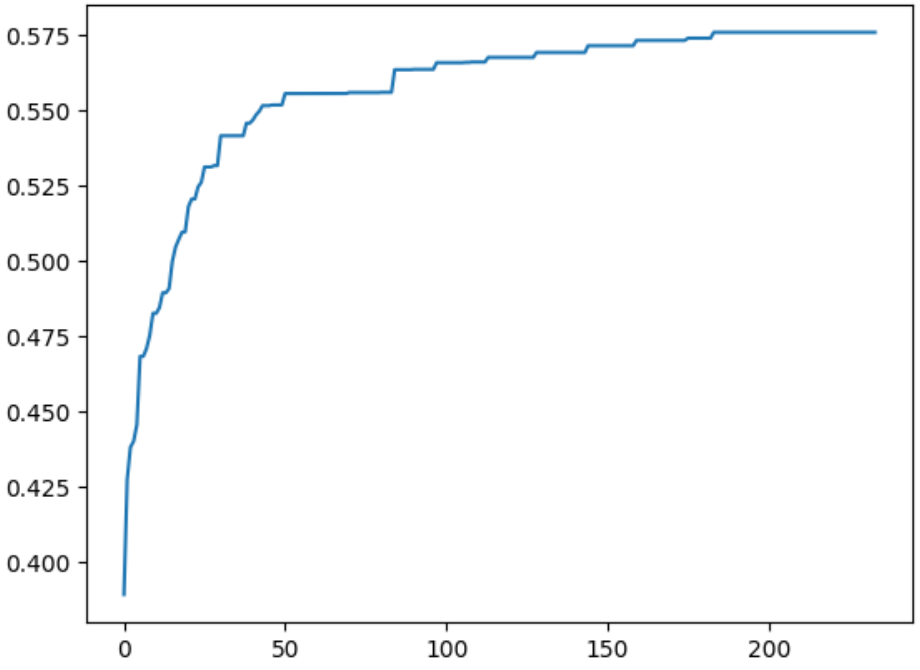


Рисунок 3.6

На рисунку 3.7 зображена гістограма порівняння нормалізованої ефективності розкладок клавіатур при сліпому методі друку. З цього маємо

наступні значення нормалізованої ефективності розкладок: "ЙЦУКЕН" – 1.317, "Шарапіка" – 1.514, створена алгоритмом розкладка – 2.303.

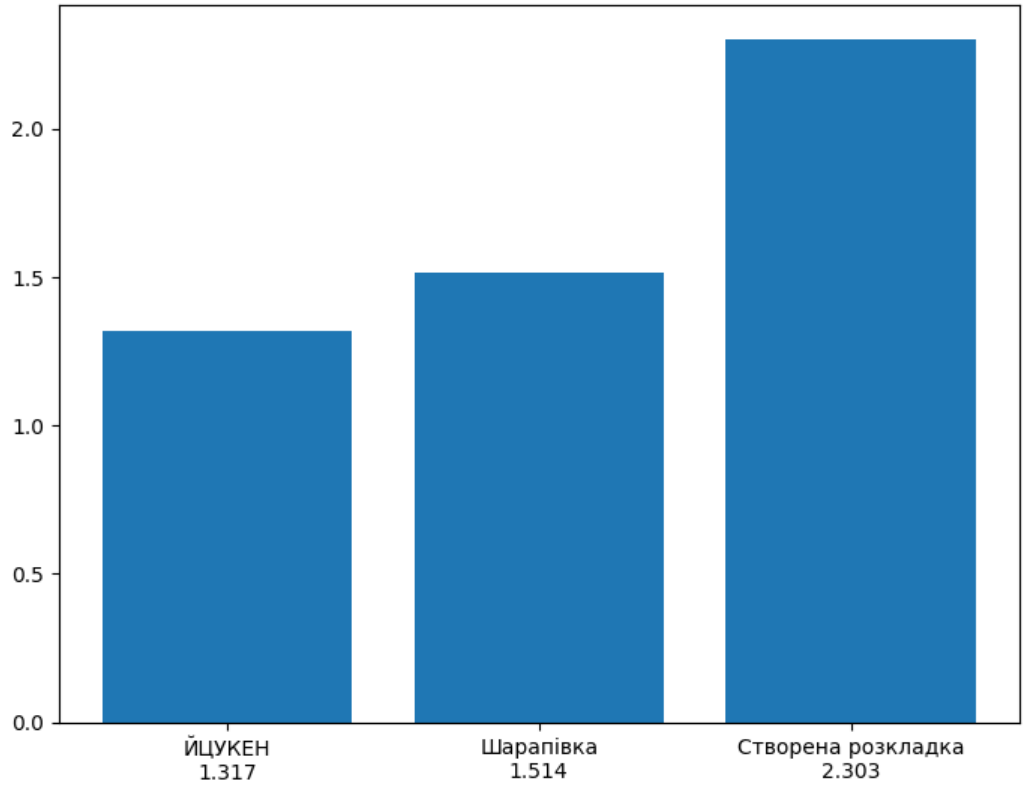


Рисунок 3.7

Щоб порівняти ефективність розкладок клавіатур на основі наданих даних, розрахуємо їх відсоткову ефективність. Для цього поділимо значення кожної розкладки на максимальне значення (2.303) та помножимо на 100, отримаємо відсотки. З цього відсоткові ефективності розкладок дорівнюють: "ЙЦУКЕН" – 57.24%, "Шарапіка" – 65.75%, створена алгоритмом розкладка – 100%. Ці відсотки свідчать про те, що створена розкладка є більш ефективної ніж інші розкладки більше ніж у півтора рази.

На рисунку 3.8 зображена гістограма порівняння нормалізованої ефективності розкладок клавіатур при методі друку двома пальцями. З цього маємо наступні значення нормалізованої ефективності розкладок:

"ЙЦУКЕН" – 0.713, "Шарапіка" – 0.785, створена алгоритмом розкладка – 1.044. Відсоткові ефективності розкладок дорівнюють: "ЙЦУКЕН" – 68.28%, "Шарапіка" – 75.10%, створена алгоритмом розкладка – 100%.

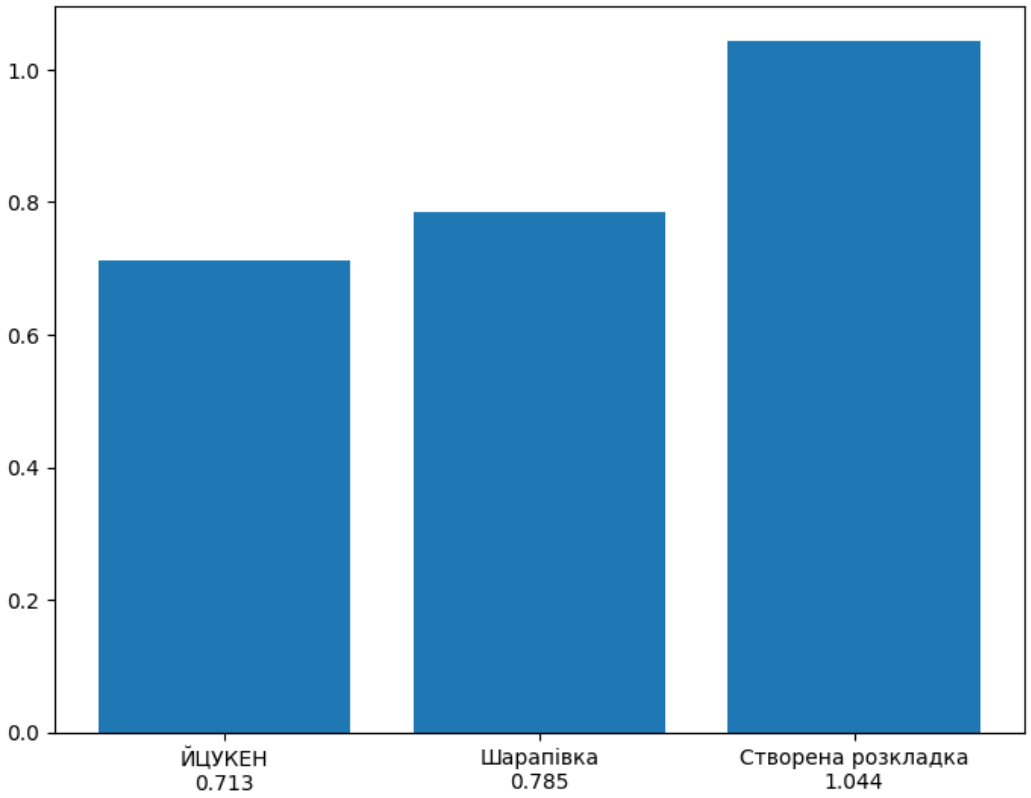


Рисунок 3.8

На рисунку 3.9 зображена гістограма порівняння нормалізованої ефективності розкладок клавіатур при методі друку одним пальцем. З цього маємо наступні значення нормалізованої ефективності розкладок: "ЙЦУКЕН" – 0.437, "Шарапіка" – 0.378, створена алгоритмом розкладка – 0.576. Відсоткові ефективності розкладок дорівнюють: "ЙЦУКЕН" – 75.87%, "Шарапіка" – 65.63%, створена алгоритмом розкладка – 100%.

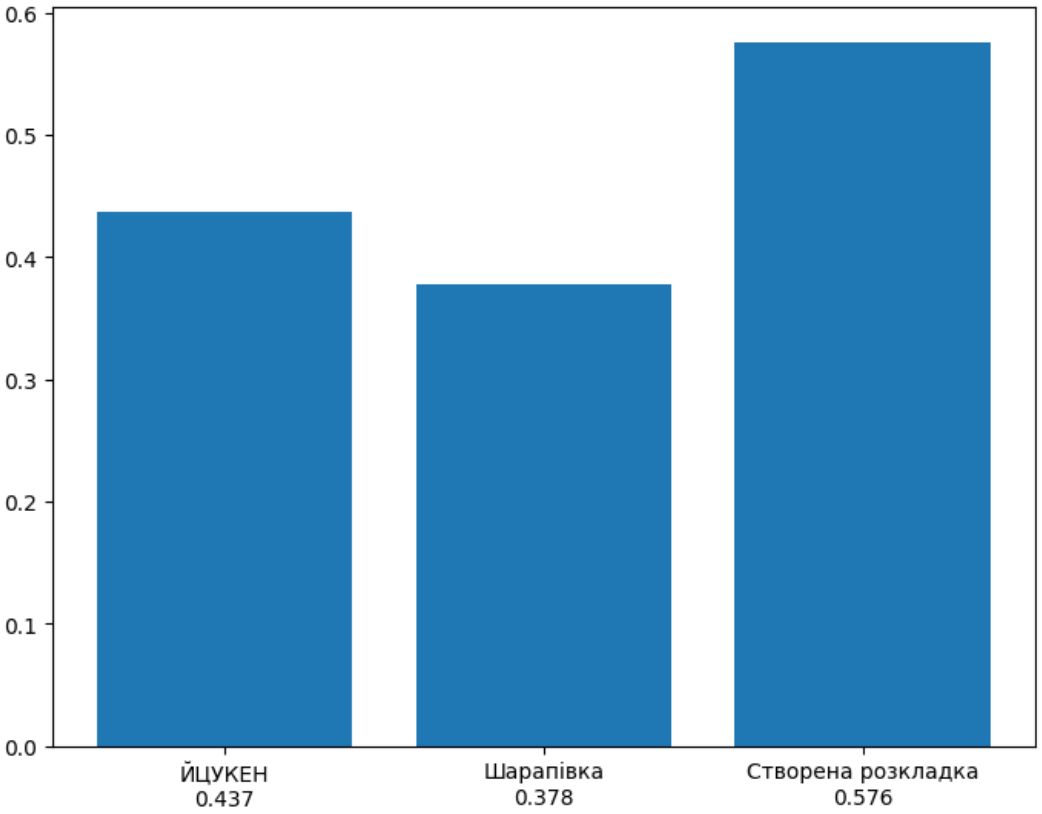


Рисунок 3.9

З цього маємо, що для кожного методу друку створена розкладка є більш ефективною ніж аналоги на 25 чи більше відсотків.

3.5 Висновок до розділу 3

Для програмної реалізації генетичного алгоримту для створення ефективної україномовної розкладки клавіатури було використано сучасні інструмнти розробки та актуальний набір даних, що відображає найбільш популярний стиль написання сучасної української мови. В результаті було створено програму з великою кількістю налаштовуваних параметрів, що дозволяють користувачу створити ефективну україномовну розкладку для клавіатури користувача та використовуваний метод набору.

Створені алгоритмом розкладки показують найбільшу ефективність у порівнянні з іншими аналогами, що дозволяє користувачу збільшити швидкість друку на 25% або більше.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задачі створення ефективної україномовної розкладки клавіатури.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору найоптимальнішого враховуючи як економічні фактори, так і характеристики продукту, які впливають на продуктивність роботи та сумісність з апаратним забезпеченням. Для цього був використаний метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) є засобом оцінки реальної вартості продукту або послуги, незалежно від організаційної структури компанії. ФВА застосовується з метою виявлення потенційних можливостей зниження витрат шляхом ефективнішого виробництва та поліпшення співвідношення між споживчою вартістю продукту та витратами на його виготовлення. Для проведення аналізу використовуються економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу передбачає визначення послідовності кроків у процесі розробки продукту, визначення загальних річних витрат та кількості робочих годин, встановлення джерел витрат та підсумковий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У даній роботі використовується метод ФВА для проведення техніко-економічного аналізу розробки ефективної україномовної розкладки клавіатури з використанням генетичного алгоритму. Оскільки

рішення щодо проектування та реалізації компонентів мають вплив на всю систему, кожна окрема підсистема повинна задовольняти встановлені вимоги. Тому фактичний аналіз включає оцінку функцій програмного продукту, що призначений для знаходження оптимального рішення для задачі пошуку ефективної україномовної розкладки клавіатури та налаштування генетичного алгоритму для вирішення цієї задачі.

Технічні вимоги до програмного продукту включають наступне:

– функціонування на персональних комп’ютерах із стандартним набором компонентів;
– забезпечення зручного та зрозумілого інтерфейсу для користувача;
– висока швидкість обробки даних та миттєвий доступ до інформації у реальному часі;
– можливість легкого масштабування та обслуговування;
– мінімізація витрат на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір методу впровадження алгоритмів;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) мова програмування Python;

б) мова програмування C++;

Функція F_2.

а) використання готових бібліотек;

б) написання алгоритму вручну;

Функція F_3:

а) Google Colab;

б) Jupiter Notebook;

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

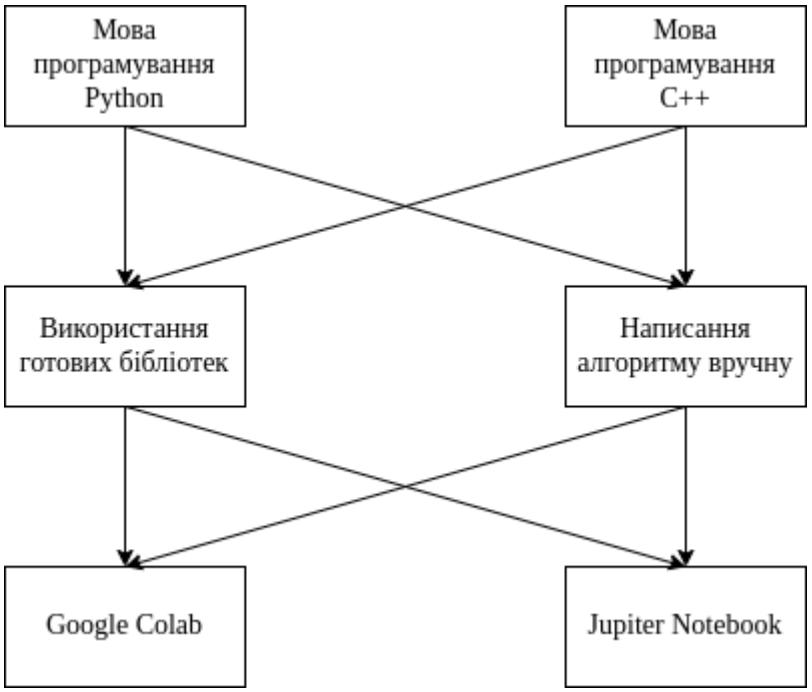


Рисунок 4.1 — Морфологічна карта

Морфологічна карта надає огляд всіх можливих варіантів основних функцій.

За допомогою цієї карти ми створюємо матрицю позитивних та негативних варіантів для основних функцій (таблиця 4.1). Зробивши

аналіз, ми приходимо до висновку, що певні варіанти реалізації функцій не відповідають поставленим завданням програмного продукту, і їх слід відкинути.

Таблиця 4.1 — Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Кросплатформеність, доступність бібліотек, займає менше часу при написанні коду	Повільна швидкість виконання коду
	Б	Код швидко виконується	Займає більше часу при написанні коду
F_2	*А*	Швидкість реалізації	Менша гнучкість
	Б	Оптимально для вирішення заданої задачі	Необхідність тестування
F_3	*А*	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету
	Б	Багато інструментів, безпечність	Підтримує одночасно лише одну мову програмування

Зробимо висновки щодо кожної окремої функції та виберемо найоптимальніші варіанти.

Функція F1:

Перевагу даємо швидкості розробки та доступності готових рішень. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F2:

Поскільки поставлена задача є дуже специфічної для генетичного алгоритму, перевагу надамо гнучкості рішення. Обираємо варіант Б.

Функція F3:

Обидва варіанти можна використовувати в розробці.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2б – F_3а

F_1а – F_2б – F_3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– X1 – швидкодія мови програмування;
– X2 – об’єм пам’яті для обчислень та збереження даних;
– X3 – час роботи алгоритму оцінки ефективності;
– X4 – потенційний об’єм програмного коду.

X1 відображає швидкість операцій залежно від обраної мови програмування. X2 відображає об'єм пам'яті в оперативній пам'яті персонального комп'ютера, необхідний для збереження та обробки даних під час виконання програми. X3 показує розмір програмного коду, який потрібно створити розробникам. X4 представляє потенційний об'єм програмного коду, який може виникнути під час розробки.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП, як показано у таблиці 4.2.

Таблиця 4.2 — Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Недоліки		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	16000	20000
Об’єм пам’яті	X2	Мб	256	128	64
Час роботи алгоритму	X3	мс	10	6	1
Потенційний об’єм програмного коду	X4	кількість рядків коду	2000	1500	800

На основі інформації з таблиці 4.2 будуються графічні характеристик параметрів, які відображені на рис. 4.2 – рис. 4.5.

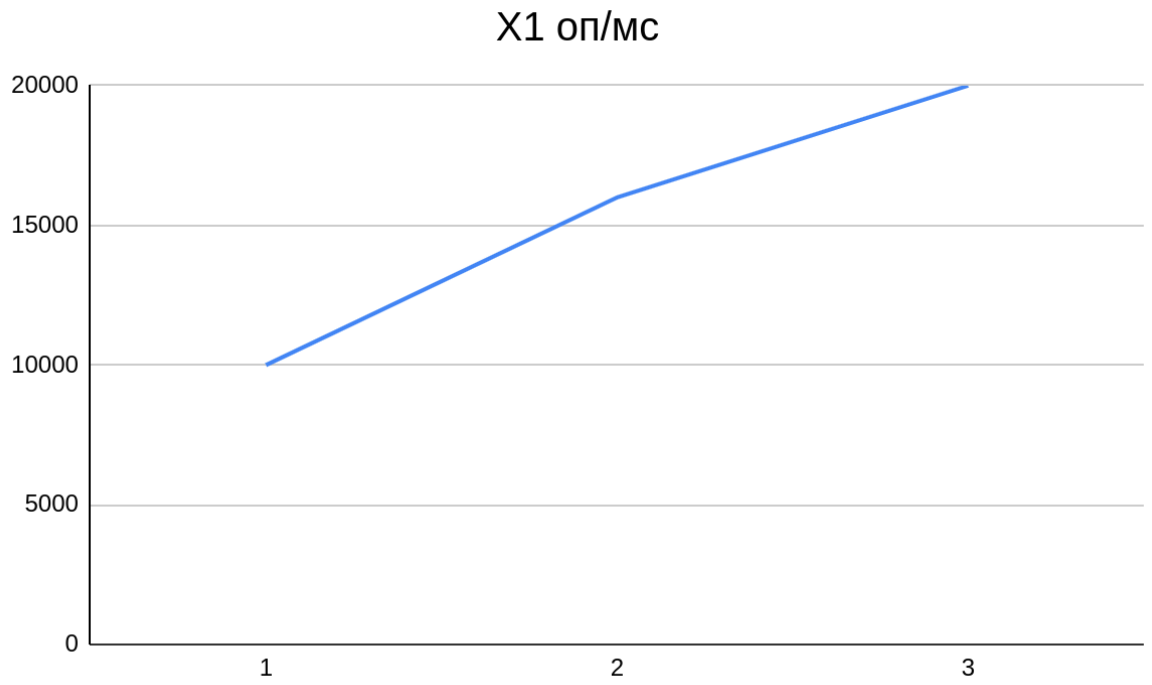


Рисунок 4.2 — X1, швидкодія мови програмування

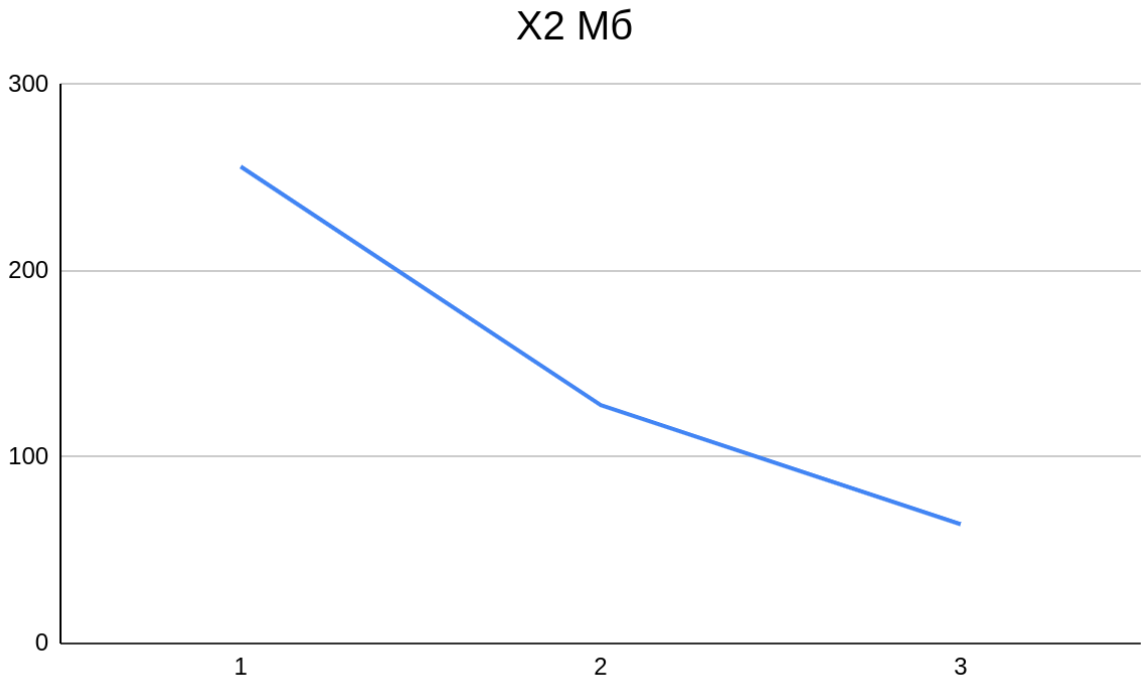


Рисунок 4.3 — X2, об’єм пам’яті

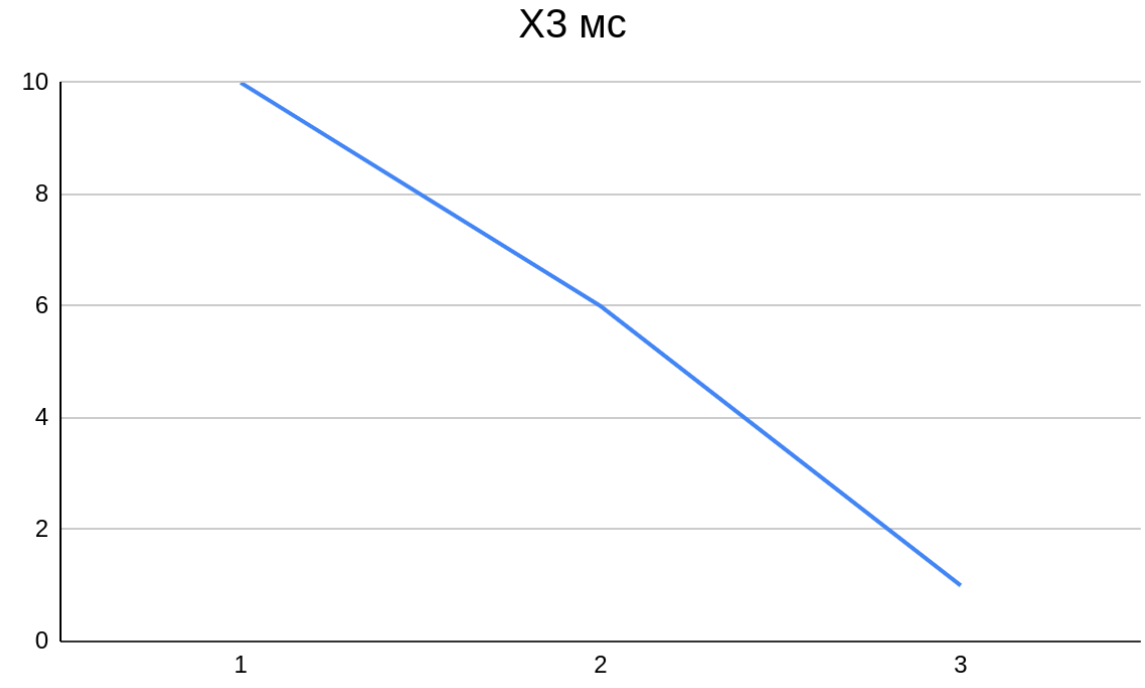


Рисунок 4.4 — X3, час роботи алгоритму

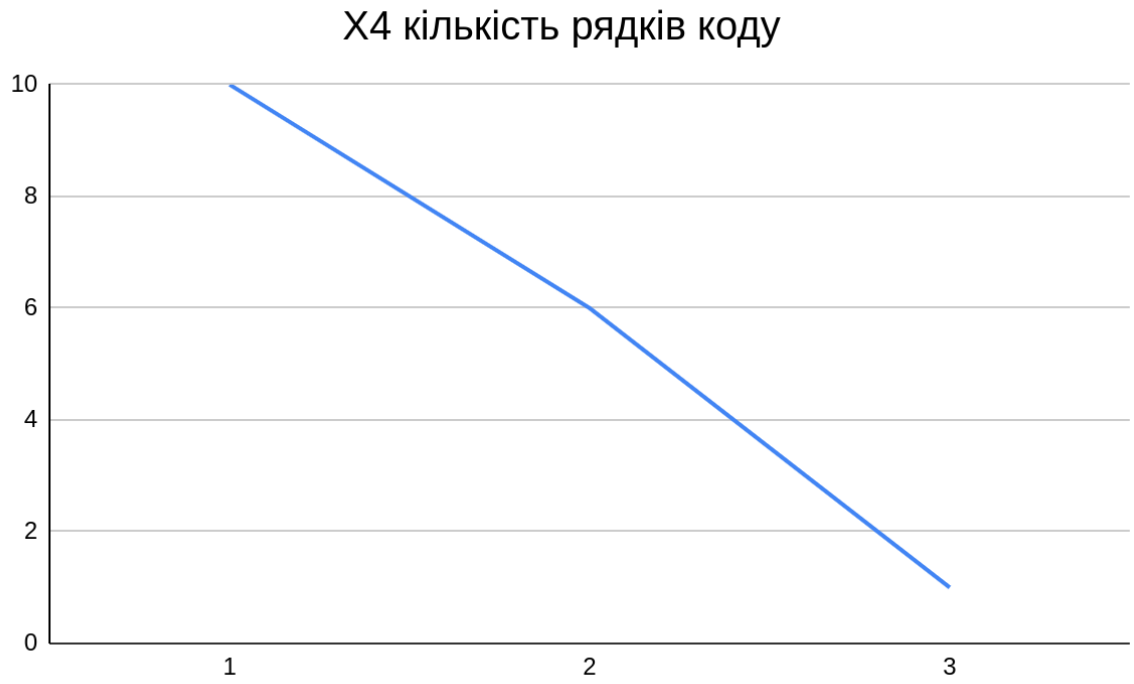


Рисунок 4.5 — X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після грунтовного обговорення й аналізу, кожен експерт оцінює важливість кожного параметру для досягнення конкретної мети - розробки програмного продукту, який забезпечує найточніші результати при визначенні параметрів моделей адаптивного прогнозування та обчисленні прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;
– перевірку придатності експертних оцінок для подальшого використання;
– визначення оцінки попарного пріоритету параметрів;
– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 — Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангі в R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	1	2	1	2	2	2	2	12	-5,5	30,25
X2	Об’єм пам’яті	Мб	3	4	4	4	3	3	4	25	7,5	56,25
X3	Час роботи алгоритму	мс	2	1	3	1	1	1	1	10	-7,5	56,25
X4	Потенційний об’єм програмного коду	кількість рядків коду	4	3	2	3	4	4	3	23	5,5	30,25
	Разом		10	10	10	10	10	10	10	70	0	173

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 173$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 173}{7^2\left(4^3 - 4\right)} = 0,706 > W_k = 0,67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 — Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	>	<	>	>	>	>	>	1,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	>	>	<	<	>	>	1,5
X3 і X4	<	<	>	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}.$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b_i = \sum_{i=1}^{N} a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}},$$

$$b_i^{'} = \sum_{i=1}^{N} a_{ij} b_j$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 — Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	1,5	0,5	3,5	0,22	12,25	0,21	44,875	0,21
X2	1,5	1	1,5	1,5	5,5	0,34	21,25	0,36	77,875	0,36
X3	0,5	0,5	1	0,5	2,5	0,16	9,25	0,16	34,125	0,15
X4	1,5	0,5	1,5	1	4,5	0,28	16,25	0,27	59,125	0,28
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (Об’єм пам’яті), X3 (час роботи алгоритму) та X4 (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1 (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості і–го параметра;

B_i – оцінка і–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	15000	12	0,21	2,52
F2	А	X2	128	15	0,36	5,4
F3	А	X3	8	8	0,15	1,2
	Б	X3	6	9	0,28	2,52

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 2{,}52 + 5{,}4 + 1{,}2 = 9{,}12;$$

$$K_{K2} = 2{,}52 + 5{,}4 + 2{,}52 = 10{,}44.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р = 60$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ} = 0.8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 60 \cdot 1.7 \cdot 0.8 = 81,6 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р = 32$ людино-днів, $К_П = 0.9$, $К_{СК} = 1$, $К_{СТ} = 0.8$:

$$Т_2 = 32 \cdot 0,9 \cdot 0,8 = 23,04 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (81,6 + 23,04 + 4,8 + 23,04) \cdot 8 = 1059,84 \text{ людино-годин.}$$

$$Т_{II} = (81,6 + 23,04 + 6,91 + 23,04) \cdot 8 = 1076,72 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 36600 грн., один аналітик в області даних з окладом 25000. Визначимо середню зарплату за годину за формулою:

$$\text{СЧ} = \frac{\text{М}}{T_m \cdot t}\text{грн.},$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$\text{СЧ} = \frac{36600+36600+25000}{3 \cdot 21 \cdot 8} = 194,84 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$\text{СЗП} = \text{С}_\text{ч} \cdot \text{Т}_i \cdot \text{КД},$$

де $С_ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 194,84 \cdot 1059,84 \cdot 1.2 = 247799,07$ грн.

II. $С_{ЗП} = 194,84 \cdot 1076,72 \cdot 1.2 = 251745,75$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 247799,07 \cdot 0,22 = 54515,79$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 251745,75 \cdot 0,22 = 55384,07$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 36600 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 36600 \cdot 0,2 = 87840 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 87840 \cdot (1 + 0.2) = 105408 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 105408 \cdot 0,22 = 23189,76 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 30000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 30000 = 8625 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 30000 \cdot 0,05 = 1725 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0,9 =$$

$$= 1677,6 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1324.8 \cdot 0.7 \cdot 0.2 \cdot 4.87 = 903,249 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 30000 \cdot 0,67 = 20100 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н},$$

$С_{ЕКС}$ = 105408 + 23189,76 + 8625 +1725 + 903,249 + 20100 = 159951,01 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 159951.01 / 1677,6 = 95,34 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_{М} = С_{М-Г} \cdot T,$$

I. $С_{М} = 95,34 \cdot 1059,84 = 101045,15$ грн.

II. $С_{М} = 95,34 \cdot 1076,72 = 102654,48$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_{Н} = С_{ЗП} \cdot 0,67,$$

I. $С_{Н} = 247799,07 \cdot 0,67 = 166025,38$ грн.

II. $С_{Н} = 251745,75 \cdot 0,67 = 168669,65$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н},$$

I. $С_{ПП}$ = 247799,07 + 54515,79 + 101045,15 + 166025,38 = 569385,39 грн.

II. $С_{ПП}$ = 251745,75 + 55384,07 + 102654.48 + 168669.65 = 578453,95 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j},$$

$$К_{ТЕР1} = 9,12 / 569385,39 = 1,601 \cdot 10^{-5},$$

$$К_{ТЕР2} = 10,44 / 578453.95 = 1,804 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1,804 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,804 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- написання алгоритму вручну;
- середовище розробки – Jupiter Notebook.

Даний варіант виконання програмного комплексу дає користувачу оптимізовану гнучку програму з зручним інтерфейсом, та доступний функціонал для роботи.

4.8 Висновки до четвертого розділу

У даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу розроблюваного програмного комплексу, було визначено та оцінено основні функції програмного продукту, а також знайдено його параметри, які його характеризують. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

На основі аналізу було обрано варіант реалізації програмного продукту.

ВИСНОВОК

В ході виконання даної дипломної роботи було побудовано модель генетичного алгоритму для створення ефективної україномовної розкладки клавіатури та реалізовано її програмно. Було розглянуто найпопулярніші та альтернативні розкладки клавіатур, відмінності між фізичною та сенсорною клавіатурами, а також різні методи набору: сліпий метод набору, метод набору двома пальцями, метод набору одним пальцем. Для більш ефективної роботи алгоритму було проаналізовано сучасну українську мову, сучасний словник української та визначено які букви найчастіше використовуються на початку слів та визначено найбільш часто використовувані букви в цілому.

Побудована модель генетичного алгоритму спирається на введені метрики ефективності розкладки, яка вираховується за допомогою сумарної дистанції, яку проходять пальці. Були описані всі етапи моделі генетичного алгоритму для вирішення задачі створення ефективної україномовної розкладки клавіатури, зокрема ініціалізація, оцінка придатності, відбір, схрещування та мутація.

Програмна реалізація дозволяє користувачу створити ефективну україномовну розкладку клавіатури під свої вимоги. Користувач може налаштувати задані розміри клавіатури та обрати необхідний метод набору. Також є можливість завантажити власний набір даних, який створений з текстових даних користувача, наприклад, завантаження власних документів для оптимізації роботи юриста. Це персоналізує розкладку під щоденні вимоги користувача.

В результаті створена розкладка є на 25% або більше ефективнішою, ніж аналоги. Це дозволяє користувачу збільшити швидкість набору та зменшити втому рук.

Можливим подальшим розвитком проекту є оптимізація швидкості роботи алгоритму, надання можливості зміни абетки розкладки та подальша оптимізація гіперпараметрів моделі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Yasuoka, Koichi; Yasuoka, Motoko. "On the Prehistory of QWERTY." March 2011. Kyoto University Research Information Repository. URL: https://repository.kulib.kyoto-u.ac.jp/dspace/bitstream/2433/139379/1/42_161.pdf

2. "Пишучі машинки. Розташування клавіш та символів на клавіатурі." ГОСТ СРСР 6431-75. 1975. URL: http://vsegost.com/Catalog/16/16199.shtml

3. Windows Keyboard Layouts. 20 March 2023. Microsoft. URL: https://learn.microsoft.com/en-us/globalization/windows-keyboard-layouts

4. ДСТУ 3470-96, 1997. URL: http://www.uazone.org/multiling/koi8-u/DSTU_3470.html

5. Лютий, Олександр. "ol03a та ol03g". 27 лютого 2017. Wayback Machine. URL: http://web.archive.org/web/20170227000000/http://liutyi.info/ol03a

6. Шарапов, О. "Розкладка клавіатури Шараповка". 10 квітня 2016. Wayback Machine. URL: https://web.archive.org/web/20160401093256/http://layout.asharapov.com/about

7. Judd, William. "Full-size, TKL, 60% and more: a guide to mechanical keyboard sizes". The Keyboard Company. 9 August 2017. URL: https://www.keyboardco.com/blog/index.php/2017/08/full-size-tkl-60-and-more-a-guide-to-mechanical-keyboard-sizes/

8. Raymond Sam. "The Reference Guide to Keyboard Sizes & Layouts [Infographic]". The Gaming Setup. 27 April 2023. Автор: URL: https://thegamingsetup.com/gaming-keyboard/buying-guides/keyboard-sizes

9. Shaicy P Shaji, Leo Pius, Maritta Simon P, Tressa Francis, Vishnu Babu K, Vineeth Francis. "Virtual Keyboard." International Journal of

Engineering and Computer Science, ISSN: 2319-7242, Vol. 6, Issue 3, March 2017, pp. 20797-20800. Department of ECE, Jyothi Engineering College, Cheruthuruthy, Thrissur, India. URL: http://www.ijecs.in/index.php/ijecs/article/view/3482/3238

10. Onscreen Keyboards. In Human Interface Guidelines. Apple Inc. URL: https://developer.apple.com/design/human-interface-guidelines/onscreen-keyboards

11. Keyboard. In Android Developers. Google LLC. URL: https://developer.android.com/reference/android/inputmethodservice/Keyboard

12. Yamada, H. (1980). A Historical Study of Typewriters and Typing Methods: from the Position of Planning Japanese Parallels. Journal of Information Processing, 2(4), 175-202. URL: https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_action_common_download&item_id=60002&item_no=1&attribute_id=1&file_no=1

13. Л. Шевченко, В. Чумак, Г. Ярун, І. Шевченко, О. Бугаков, В. Білоноженко. Словник української мови. В 20 томах. Наукова думка. 2010.

14. Архіпов О.О., Журавльов В.М. Частотний аналіз використання букв української мови. "Радіоелектроніка. Інформатика. Управління", ISSN 1607-3274, 2009. URL: https://cyberleninka.ru/article/n/chastotniy-analiz-vikoristannya-bukv-ukrayinskoyi-movi/pdf

15. Кононюк, А.Ю. Нейронні мережі і генетичні алгоритми. Київ: 2008. 446 с. URL: http://ecat.diit.edu.ua/ft/NN_GA.pdf

16. Whitley, D. A Genetic Algorithm Tutorial. Department of Computer Science. Colorado State University. November 10, 1993. URL: https://www.cs.colostate.edu/TechReports/Reports/1993/tr-103.pdf

17. Zhou, W. (Journal of Physics: Conference Series 1952). Fast Implementation of Genetic Algorithm Based on Software/Hardware Co-design

Method. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1952/3/032044/pdf

18. Eiben, A.E., Rau6, P-E., & Ruttkay, Zs. (October 1994). Genetic algorithms with multi-parent recombination. Artificial Intelligence Group, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1952/3/032044/pdf

19. Juan Lujano-Rojas, Rodolfo Dufo-López, José A. Domínguez-Navarro. "Genetic algorithms and other heuristic techniques in power systems optimization." Renewable and Sustainable Energy Reviews, vol. 58, 2016, pp. 1077-1089. URL: https://www.sciencedirect.com/science/article/abs/pii/B9780128238899000011

20. B. Ivanyuk-Skulskiy, A. Zaliznyi, O. Reshetar, O. Protsyk, B. Romanchuk, V. Shpihanovych, "ua_datasets: a collection of Ukrainian language datasets," 2021. URL: https://github.com/fido-ai/ua-datasets.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
# In[247]:
import random
import math
import os
import numpy as np
import matplotlib.pyplot as plt
import time
from multiprocessing.pool import ThreadPool

# In[248]:

def string_to_dict(string):
    dict = {}
    for letterIndex in range(len(string)):
      dict[letterIndex] = string[letterIndex]
    return dict

# In[249]:

current_dir = os.getcwd()
# f = open('/content/drive/MyDrive/Дипломна_Робота_Бакалавр/dataset.txt',
"r")
f = open(current_dir + '\\250_dataset.txt"', "r", encoding="utf-8")
# data_set = f.read()
data_set = f.read().lower()
```

```
f2 = open(current_dir + '\\250_test_dataset.txt'", "r", encoding="utf-8")
test_data_set = f2.read().lower()

# In[251]:

alphabet = list('абвгдеєжзиіїйклмнопрстуфхцчшщьюя')
alphabet_string = 'абвгдеєжзиіїйклмнопрстуфхцчшщьюя' #г
jcuken = [list('йцукенгшщзхї'), list('фівапролджє'), list('ячсмитьбю')]
sharapivka = [list('йцукеплшщзхї'), list('фіиаорнтвдє'), list('ячсмжгьбю')]

# In[252]:

ten_fingers_method_init_fingers_position = [[1, 0], [1, 1], [1, 2], [1, 3], [1, 6],
[1, 7], [1, 8], [1, 9]]
ten_fingers_method_finger_buttons = [['0:0', '1:0', '2:0'], ['0:1', '1:1', '2:1'], ['0:2',
'1:2', '2:2'], ['0:3', '1:3', '2:3', '0:4', '1:4', '2:4'],
                                     ['0:5', '1:5', '2:5', '0:6', '1:6', '2:6'], ['0:7', '1:7', '2:7'], ['0:8',
'1:8', '2:8'], ['0:9', '1:9', '0:10', '1:10', '0:11']]

two_fingers_method_init_fingers_position = [[1, 3], [1, 7]]
two_fingers_method_finger_buttons = [['0:0', '1:0', '2:0', '0:1', '1:1', '2:1', '0:2',
'1:2', '2:2', '0:3', '1:3', '2:3', '0:4', '1:4', '2:4', '0:5', '1:5', '2:5'],
                                     ['0:6', '1:6', '2:6', '0:7', '1:7', '2:7', '0:8', '1:8', '2:8', '0:9',
'1:9', '0:10', '1:10', '0:11']]

one_finger_method_init_fingers_position = [[1, 5]]
one_finger_method_finger_buttons = [['0:0', '1:0', '2:0', '0:1', '1:1', '2:1', '0:2',
'1:2', '2:2', '0:3', '1:3', '2:3', '0:4', '1:4', '2:4', '0:5', '1:5', '2:5', '0:6', '1:6', '2:6', '0:7',
'1:7', '2:7', '0:8', '1:8', '2:8', '0:9', '1:9', '0:10', '1:10', '0:11']]
```

```
class TypingMethod:
  # init_fingers_position = [[]]
  # current_fingers_position = [[]]
  # finger_buttons = [[]]
  # prev_finger = -1
  # alphabet_string = 'абвгґдеєжзиіїйклмнопрстуфхцчшщьюя' #ґ
  # alphabet = list('абвгґдеєжзиіїйклмнопрстуфхцчшщьюя') #ґ
  def list_to_dict(self, list):
    dict = {}
    for i in range(len(list)):
      dict[i] = list[i]
    return dict

  def finger_buttons_to_dict(self, finger_buttons):
    dict = {}
    for finger in range(len(finger_buttons)):
      for button in finger_buttons[finger]:
        dict[button] = finger
    return dict



  def __init__(self, init_fingers_position =
ten_fingers_method_init_fingers_position, finger_buttons =
ten_fingers_method_finger_buttons,
alphabet_string='абвгґдеєжзиіїйклмнопрстуфхцчшщьюя', type =
'ten_fingers'):

    if type == 'ten_fingers':
```

```
        self.init_fingers_position =
self.list_to_dict(ten_fingers_method_init_fingers_position.copy())
        self.current_fingers_position =
self.list_to_dict(ten_fingers_method_init_fingers_position.copy())
        self.buttons_finger =
self.finger_buttons_to_dict(ten_fingers_method_finger_buttons.copy())

    elif type == 'two_fingers':
        self.init_fingers_position =
self.list_to_dict(two_fingers_method_init_fingers_position.copy())
        self.current_fingers_position =
self.list_to_dict(two_fingers_method_init_fingers_position.copy())
        self.buttons_finger =
self.finger_buttons_to_dict(two_fingers_method_finger_buttons.copy())

    elif type == 'one_finger':
        self.init_fingers_position =
self.list_to_dict(one_finger_method_init_fingers_position.copy())
        self.current_fingers_position =
self.list_to_dict(one_finger_method_init_fingers_position.copy())
        self.buttons_finger =
self.finger_buttons_to_dict(one_finger_method_finger_buttons.copy())

    else:
        self.init_fingers_position = self.list_to_dict(init_fingers_position.copy())
        self.current_fingers_position = self.list_to_dict(init_fingers_position.copy())
        self.buttons_finger = self.finger_buttons_to_dict(finger_buttons.copy())

    self.alphabet = list(alphabet_string)
```

```
    self.alphabet_string = alphabet_string
    self.prev_finger = 0

  def find_button_finger(self, row, column):
    button = str(row) + ':' + str(column)
    return self.buttons_finger[button]

  def neighbour_distance_mean(self, keyboard):
    total = 0
    total += keyboard.distances['1->1:1']
    total += keyboard.distances['1->1:1']
    total += keyboard.distances['1->0:0']
    total += keyboard.distances['1->2:0']
    return total/4

  def count_total_distance(self, keyboard, data):
    # data = data.lower()
    total_distance = 0
    amount_of_letters = 0

    for letter in data:

      if letter in self.alphabet_string:
        if (letter == 'r' and 'r' in self.alphabet_string):
          letter = 'r'

        finger = self.find_button_finger(keyboard.layout_dict[letter][0],
keyboard.layout_dict[letter][1])
        finger_current_position = self.current_fingers_position[finger]
```

```
        letter_position = keyboard.layout_dict[letter]
        total_distance += keyboard.distances[str(finger_current_position[0]) + '->'
+ str(letter_position[0]) + ':' +
str(abs(finger_current_position[1]-letter_position[1]))]
        amount_of_letters += 1
        self.current_fingers_position[finger] = letter_position

        if finger != self.prev_finger:
          self.current_fingers_position[self.prev_finger] =
self.init_fingers_position[self.prev_finger]

        self.prev_finger = finger

      else:
        self.current_fingers_position[self.prev_finger] =
self.init_fingers_position[self.prev_finger]

    return [total_distance,  amount_of_letters/total_distance,
(self.neighbour_distance_mean(keyboard)*amount_of_letters)/total_distance]

# In[253]:

class KeyBoard:
  # layout = [[]]
  # layout_dict = {}
  layout_sizes = [12, 11, 9]
  alphabet_string = 'абвгдеєжзиіїйклмнопрстуфхцчшщьюя' #г
  alphabet = list('абвгдеєжзиіїйклмнопрстуфхцчшщьюя') #г
  most_used_alphabet = 'оаниівтерск'
```

```
  # distances = {}
  # button_width = 1.7
  # button_height = 1.7
  # buttons_distance_vertical = 0.1
  # buttons_distance_horizontal = 0.1
  # indents = [0, 0.5, 1.3]

  def __init__(self, layout = None, button_width = 1.8, button_height = 1.8,
buttons_distance_vertical = 0.105, buttons_distance_horizontal = 0.105, indents
= [0, 0.6, 1.4]):
    self.layout_dict = {}
    self.button_width = button_width
    self.button_height = button_height
    self.buttons_distance_vertical = buttons_distance_vertical
    self.buttons_distance_horizontal = buttons_distance_horizontal
    self.indents = indents
    if (layout == None):
      self.set_random_layout()
    else:
      self.layout = layout.copy()
    self.build_layout_dict()

    self.calculate_distances()

  def build_layout_dict(self):
    for i in range(len(self.layout)):
      for j in range(len(self.layout[i])):
        self.layout_dict[self.layout[i][j]] = [i, j]
```

```
def set_layout(self, layout):
   self.layout = layout.copy()
   self.build_layout_dict()


def layout_to_string(self):
  result_string = ''
  for i in range(len(self.layout)):
    for j in range(len(self.layout[i])):
      result_string += self.layout[i][j]
  return result_string

def layout_to_string_static(layout):
  result_string = ''
  for i in range(len(layout)):
    for j in range(len(layout[i])):
      result_string += layout[i][j]
  return result_string


def set_layout_from_string(self, string):
  result_layout = [[] for _ in range(len(self.layout_sizes))]
  for i in range(len(result_layout)):
    row_length = self.layout_sizes[i]
    result_layout[i] = ['']*row_length
    for j in range(row_length):
      result_layout[i][j] = string[alphabet_index]
      alphabet_index += 1
```

```
    self.set_layout(result_layout)


  def layout_from_string_static(string, layout_sizes):
    result_layout = [[] for _ in range(len(layout_sizes))]
    alphabet_index = 0
    for i in range(len(result_layout)):
      row_length = layout_sizes[i]
      result_layout[i] = ['']*row_length
      for j in range(row_length):
        result_layout[i][j] = string[alphabet_index]
        alphabet_index += 1

    return result_layout

  def set_random_layout(self):
    random_alphabet = list(self.alphabet_string)
    random_alphabet = [letter for letter in random_alphabet if letter not in
self.most_used_alphabet]
    random_most_used_alphabet = list(self.most_used_alphabet)

    random.shuffle(random_alphabet)
    random.shuffle(random_most_used_alphabet)

    alphabet_index = 0
    most_used_alphabet_index = 0

    self.layout = [[] for _ in range(len(self.layout_sizes))]
    for i in range(len(self.layout)):
```

```
      row_length = self.layout_sizes[i]
      self.layout[i] = ['']*row_length
      for j in range(row_length):
        if i == 1:
          self.layout[i][j] =
random_most_used_alphabet[most_used_alphabet_index]
          most_used_alphabet_index += 1

        else:
          self.layout[i][j] = random_alphabet[alphabet_index]
          alphabet_index += 1

    self.calculate_distances()
    return self.layout

  def print_layout(self):
    for i in range(len(self.layout)):
      print(' '*i, end = '')
      for j in self.layout[i]:
        print(j, end = ' ')
      print('')

  def calculate_distances(self):
    self.distances = {}
    for rowFrom in range(len(self.layout)):
      for columnFrom in range(len(self.layout[rowFrom])):
        for rowTo in range(len(self.layout)):
          for columnTo in range(len(self.layout[rowTo])):
```

```
            key = str(rowTo)+'->'+str(rowFrom) + ':' +
str(abs(columnTo-columnFrom))
            if key not in self.distances:
              horizontal_distance = abs(self.indents[rowTo] - self.indents[rowFrom]
+ (columnTo -
columnFrom)*(self.buttons_distance_horizontal+self.button_width))
              vertical_distance = abs((rowTo -
rowFrom)*(self.buttons_distance_vertical+self.button_height))
              self.distances[key] = math.sqrt(horizontal_distance**2 +
vertical_distance**2)
              # print(key, vertical_distance, horizontal_distance, self.distances[key])

# In[254]:
def take_total_distance(result):
  return result[0][0]
class GeneticAlgorithm:
  def __init__(self, data_set, population_size = 300, selection_size=100,
crossover_probability = 0.7, amount_of_populations = 1000, typing_method =
TypingMethod(), init_keyboard = None):
    self.population_size = population_size
    self.crossover_probability = crossover_probability
    self.selection_size = selection_size
    self.amount_of_populations = amount_of_populations
    self.best_result = None
    self.times_same_best = 0

    self.progress = []

    self.data_set = data_set
```

```
    self.typing_method = typing_method

    self.init_keyboard = init_keyboard

    if init_keyboard == None:
      self.population = [KeyBoard() for _ in range(self.population_size)]
    else:
      self.population = [KeyBoard(button_width = init_keyboard.button_width,
button_height = init_keyboard.button_height, buttons_distance_vertical =
init_keyboard.buttons_distance_vertical, buttons_distance_horizontal =
init_keyboard.buttons_distance_horizontal, indents = init_keyboard.indents) for
_ in range(self.population_size)]
  def get_best_of_population(self, results):
    # results.sort(key=self.take_total_distance)
    return [results[i][1] for i in range(self.selection_size)]

  def get_result_of_keyboard(self, element):
    return [self.typing_method.count_total_distance(element, data_set), element]
  def run(self):
    max_populations_without_new_best = 50
    for i in range(self.amount_of_populations):
      if self.times_same_best >= max_populations_without_new_best:
        break

      # pool = ThreadPool(processes=self.population_size)
      results = list(map(self.get_result_of_keyboard, self.population))
      results.sort(key=take_total_distance)
      self.progress.append(results[0][0])
      # for j in range(len(self.population)):
```

```
    #   results[j] = [self.typing_method.count_total_distance(self.population[j],
self.data_set), self.population[j]]
    # results = pool.map(lambda x: [typing_method.count_total_distance(x,
data_set), x], self.population)
    # pool.close()
    # pool.join()

    result_distances = list(map(lambda el: el[0][0], results))
    sorted_population = list(map(lambda el: el[1], results))
    if (self.best_result is None) or (results[0][0][0] < self.best_result[0][0]):
      self.best_result = results[0]
      self.times_same_best = 0
    else:
      self.times_same_best
      self.times_same_best += 1
    if (i + 1)%(self.amount_of_populations / 10) == 0:
      print('Calculating:', str(i+1) + '/' + str(self.amount_of_populations), 'Best of
population:', results[0][0])
    best_of_population = self.get_best_of_population(results)
    # best_of_population[0].print_layout()
    selection_slice = slice(self.selection_size)
    selected_pairs =
GeneticAlgorithm.selection(sorted_population[selection_slice],
result_distances[selection_slice], int(self.population_size/2))
    new_population = []
    for pair in selected_pairs:
      children = self.crossover_double(pair[0].layout_to_string(),
pair[1].layout_to_string())
      children = [self.mutation(children[0]), self.mutation(children[1])]
```

```
new_population.append(KeyBoard(KeyBoard.layout_from_string_static(''.join(c
hildren[0]), pair[0].layout_sizes)))

new_population.append(KeyBoard(KeyBoard.layout_from_string_static(''.join(c
hildren[1]), pair[0].layout_sizes)))
    # for _ in range(self.population_size):
    #   board1 = random.choice(best_of_population)
    #   board2 = random.choice(best_of_population)
    #   child = GeneticAlgorithm.crossover(board1.layout_to_string(),
board2.layout_to_string())
    #   child = self.mutation(child)
    #
    #   if(self.init_keyboard == None):
    #
new_population.append(KeyBoard(KeyBoard.layout_from_string_static(''.join(c
hild), board1.layout_sizes)))
    #   else:
    #
new_population.append(KeyBoard(KeyBoard.layout_from_string_static(''.join(c
hild), board1.layout_sizes), button_width = self.init_keyboard.button_width,
button_height = self.init_keyboard.button_height, buttons_distance_vertical =
self.init_keyboard.buttons_distance_vertical, buttons_distance_horizontal =
self.init_keyboard.buttons_distance_horizontal, indents =
self.init_keyboard.indents))
     self.population = new_population
   print('')
   print('Best result:')
```

```
    print(self.best_result[0])
    self.best_result[1].print_layout()

  def selection(population, fitness_values, num_pairs):
      # Calculate total fitness
      modified_fitness_values = list(map(lambda el: el**2, fitness_values))
      total_fitness = sum(modified_fitness_values)
      # Calculate selection probabilities
      probabilities = [fitness / total_fitness for fitness in modified_fitness_values]

      selected_pairs = []
      # Perform selection of pairs
      for _ in range(num_pairs):
          pair = []

          # Select first parent
          parent1 = GeneticAlgorithm.select_individual(population, probabilities)
          pair.append(parent1)

          # Select second parent (avoid selecting the same parent)
          parent2 = GeneticAlgorithm.select_individual(population, probabilities,
exclude=[parent1])
          pair.append(parent2)
          selected_pairs.append(pair)
      return selected_pairs
  def select_individual(population, probabilities, exclude=None):
      # Create a list of indices for the population
      indices = list(range(len(population)))
      # Exclude certain individuals if provided
```

```
        if exclude:
            indices = [i for i in indices if population[i] not in exclude]
        # Perform roulette wheel selection
        selected_index = GeneticAlgorithm.roulette_wheel_selection(probabilities,
indices)
        selected_individual = population[selected_index]
        return selected_individual
    def roulette_wheel_selection(probabilities, indices):
        # Select an index based on roulette wheel selection
        rand = random.random()
        cumulative_probability = 0
        for i in indices:
            cumulative_probability += probabilities[i]
            if rand <= cumulative_probability:
                return i
        # In case of rounding errors, return the last index
        return indices[-1]
    def crossover_double(self, board1, board2):
      if self.crossover_probability < random.random():
        return [list(board1), list(board2)]
      board_length = len(board1)
      idx = random.randint(0, board_length-1)
      length = random.randint(0, board_length-1)
      return [GeneticAlgorithm.crossover(board1, board2, idx, length),
GeneticAlgorithm.crossover(board2, board1, idx, length)]
    def crossover(board1, board2, idx=None, length=None):
      board_length = len(board1)
      if(idx is None):
        idx = random.randint(0, board_length-1)
```

```
  if(length is None):
    length = random.randint(0, board_length-1)
  child = ['_' for i in range(board_length)]
  # Add keys from keyboard 1
  for i in range(length):
    if idx > board_length-1:
      idx = 0
    child [idx] = board1[idx]
    idx += 1
  # Add remaining keys from keyboard 2
  child_idx= idx
  while '_' in child:
    if idx > board_length-1:
      idx = 0
    if child_idx > board_length-1:
      child_idx = 0
    char = board2[idx]
    if char in child:
      idx += 1
      continue
    child[child_idx] = board2[idx]
    child_idx+= 1
    idx += 1
  return child

def mutation(self, board):
  # 10% chance of random mutation
  amount_of_mutations = 3
  probability_of_mutations = 0.1
```

```
    for i in range(amount_of_mutations):
      prob = random.random()
      if prob <= probability_of_mutations:
        point1 = random.randint(0, 29)
        point2= random.randint(0, 29)
        allele2 = board[point1]
        allele1 = board[point2]
        board[point1] = allele1
        board[point2] = allele2
    return board
# In[255]:
desktop_key_board = KeyBoard()
desktop_key_board.set_random_layout()

mobile_key_board = KeyBoard(button_width = 42, button_height = 60,
buttons_distance_vertical = 15, buttons_distance_horizontal = 8, indents = [0, 0,
54])
mobile_key_board.set_random_layout()
# key_board.print_layout()

keyboard_jcuken = KeyBoard(layout=jcuken)
keyboard_sharapivka = KeyBoard(layout=sharapivka)
mobile_keyboard_jcuken = KeyBoard(layout=jcuken, button_width = 42,
button_height = 60, buttons_distance_vertical = 15, buttons_distance_horizontal
= 8, indents = [0, 0, 54])
mobile_keyboard_sharapivka = KeyBoard(layout=sharapivka, button_width =
42, button_height = 60, buttons_distance_vertical = 15,
buttons_distance_horizontal = 8, indents = [0, 0, 54])
```

```
# print(ten_fingers_method.count_total_distance(key_board_jcuken.layout,
key_board_jcuken.layout_dict, key_board_jcuken.distances, data_set))
# In[256]:
ten_fingers_method = TypingMethod(type='ten_fingers')
genetic_algorithm_ten_fingers_method = GeneticAlgorithm(data_set=data_set,
typing_method=ten_fingers_method, init_keyboard=desktop_key_board)

tic = time.perf_counter()
genetic_algorithm_ten_fingers_method.run()
toc = time.perf_counter()
print(f"Calculated GA in {toc - tic:0.4f} seconds")
# In[257]:
plt.plot(list(map(lambda x: x[2],
genetic_algorithm_ten_fingers_method.progress)))
plt.show()
# In[258]:
two_fingers_method = TypingMethod(type='two_fingers')
genetic_algorithm_two_fingers_method = GeneticAlgorithm(data_set=data_set,
typing_method=two_fingers_method, init_keyboard=mobile_key_board)
genetic_algorithm_two_fingers_method.run()
# In[259]:
plt.plot(list(map(lambda x: x[2],
genetic_algorithm_two_fingers_method.progress)))
plt.show()
# In[260]:
one_finger_method = TypingMethod(type='one_finger')
genetic_algorithm_one_fingers_method = GeneticAlgorithm(data_set=data_set,
typing_method=one_finger_method, init_keyboard=mobile_key_board)
genetic_algorithm_one_fingers_method.run()
```

```
# In[261]:
plt.plot(list(map(lambda x:
x[2],genetic_algorithm_one_fingers_method.progress)))
plt.show()
# In[262]:
def make_bar_plot(method, method_name, keyboard, keyboard_jcuken,
keyboard_sharapivka):
    fig = plt.figure()
    ax = fig.add_axes([0,0,1,1])
    # ax.set_title(method_name)
    keyboard_results = [method.count_total_distance(keyboard_jcuken,
test_data_set)[2], method.count_total_distance(keyboard_sharapivka,
test_data_set)[2], method.count_total_distance(keyboard, test_data_set)[2]]
    keyboards_names =
['ЙЦУКЕН\n'+"{:.3f}".format(round(keyboard_results[0], 3)),
'Шарапівка\n'+"{:.3f}".format(round(keyboard_results[1], 3)), 'Створена
розкладка\n'+"{:.3f}".format(round(keyboard_results[2], 3))]
    print(method.count_total_distance(keyboard_jcuken, data_set)[2],
method.count_total_distance(keyboard_sharapivka, data_set)[2],
method.count_total_distance(keyboard, data_set)[2])

    ax.bar(keyboards_names, keyboard_results)
    plt.show()
# In[263]:
make_bar_plot(method=ten_fingers_method, method_name='Сліпий метод',
keyboard=genetic_algorithm_ten_fingers_method.best_result[1],
keyboard_jcuken=keyboard_jcuken,
keyboard_sharapivka=keyboard_sharapivka)
```

```
# In[264]:

make_bar_plot(method=two_fingers_method, method_name='Метод двох
пальців', keyboard=genetic_algorithm_two_fingers_method.best_result[1],
keyboard_jcuken=mobile_keyboard_jcuken,
keyboard_sharapivka=mobile_keyboard_sharapivka)

# In[265]:

make_bar_plot(method=one_finger_method, method_name='Метод одного
пальця', keyboard=genetic_algorithm_one_fingers_method.best_result[1],
keyboard_jcuken=mobile_keyboard_jcuken,
keyboard_sharapivka=mobile_keyboard_sharapivka)
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Дипломна робота на тему:

Створення ефективної україномовної розкладки клавіатури за допомогою генетичного алгоритму

Виконав:
студент групи КІ-92
Бурков Антон Олексійович
Науковий керівник:
д.т.н., професор Чумаченко О.І.

Актуальність задачі

Актуальність задачі походить з наступних мінусів найпопулярніших розкладок клавіатури:

- Застарілість
- Невідповідність вимогам сучасної української мови

Та мінусів альтернативних розкладок:

- Розгляд найбільш використовуваних букв українського алфавіту, але ігнорування найбільш частих буквосполучень
- Відсутність адаптованості під різні методу набору на клавіатурі
- Відсутність адаптованості під різні види та розміри клавіатур

Постановка задачі

- Провести аналіз методів набору на клавіатурі
- Провести аналіз сучасної української мови
- Визначити метрики ефективності клавіатури
- Побудувати модель генетичного алгоритму для вирішення задачі
- Підібрати найефективніші параметри генетичного алгоритму
- Виконати порівняльний аналіз існуючих розкладок з результатом роботи генетичного алгоритму

Огляд предметної області

Існуючі розкладки клавіатур

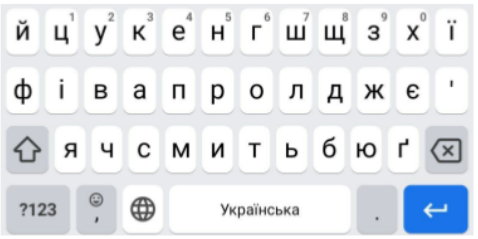
Розкладка "ЙЦУКЕН"

Розкладка "Шарапівка"

Огляд предметної області

Види клавіатур

Фізична клавіатура



Віртуальна клавіатура

Огляд предметної області

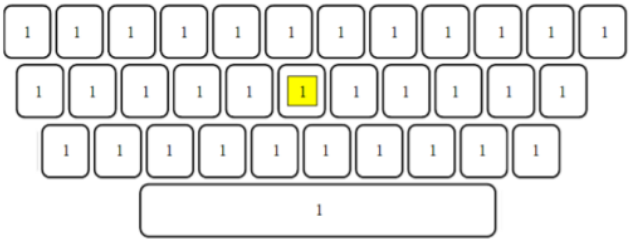
Методи набору на клавіатурі

Сліпий метод набору

Метод набору двома пальцями

Огляд предметної області

Методи набору на клавіатурі

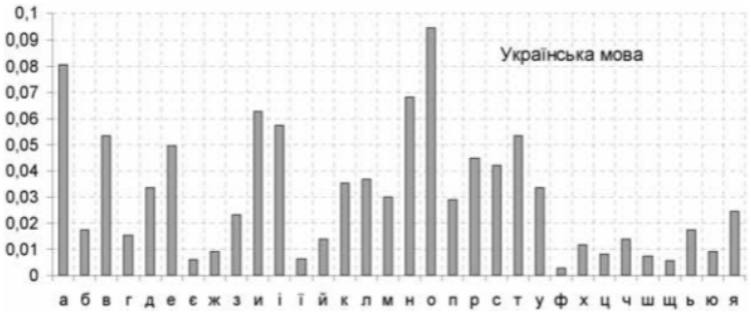


1 — використовуваний палець

Метод набору одним пальцем

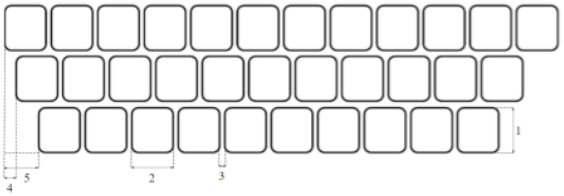
Огляд предметної області

Сучасна українська мова



Найбільш часто використовувані букви в українській мові згідно з частотним аналізом:
о, а, н, и, і, в, т, е, р, с, к.

Метрика ефективності



1 — довжина клавіші
2 — ширина клавіші
3 — дистанція між клавішами
4 — відступ другого ряду
5 — відступ третього ряду

$$f(a_{i,j}, b_{m,n}) = (|x_i - x_m| + |i - m| \cdot |d_h - w|)^2 + ((j - n) \cdot (d_v - l))^2$$

де $a_{i,j}$ – клавіша у рядку i та стовпчику j;

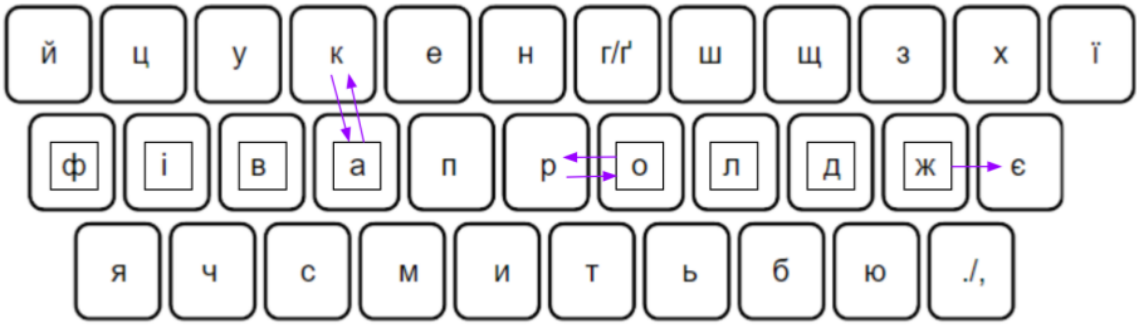
$b_{m,n}$ – клавіша у рядку m та стовпчику n;

x_i ,x_m – відступи рядка i, – відступ рядка m;

d_h та d_v – горизонтальна та вертикальна дистанції між клавішами;

w та l – ширина та довжина клавіш.

Метрика ефективності



f(розкладка) = f(о, р) + f(р,о) + f(ж, з) + f(а,к) + f(л, л) + f(а, а) + f(д, д) + f(а, к) + f(к, а)

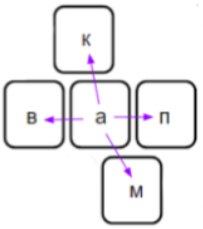
Метрика ефективності

Ефективність розкладки =

len(data_set) / f(data_set), де len(data_set) - кількість символів у датасеті

Нормалізована ефективність розкладки =

(len(data_set) * mean_distance) / f(data_set), де mean_distance - середня дистанція на клавіатурі від клавіші до верхнього, нижнього, правого та лівого сусіда

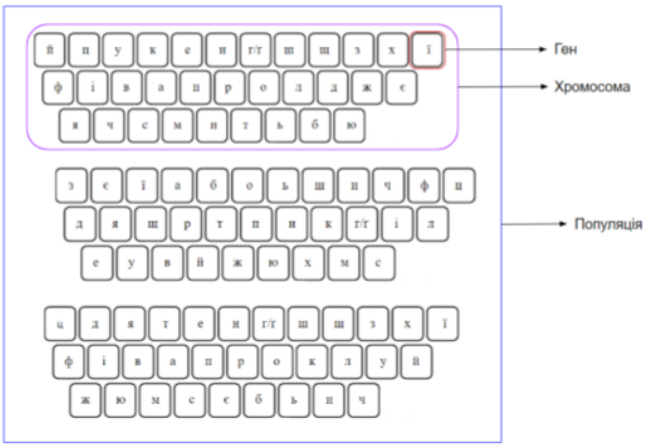


Модель генетичного алгоритму

Блок-схема генетичного алгоритму

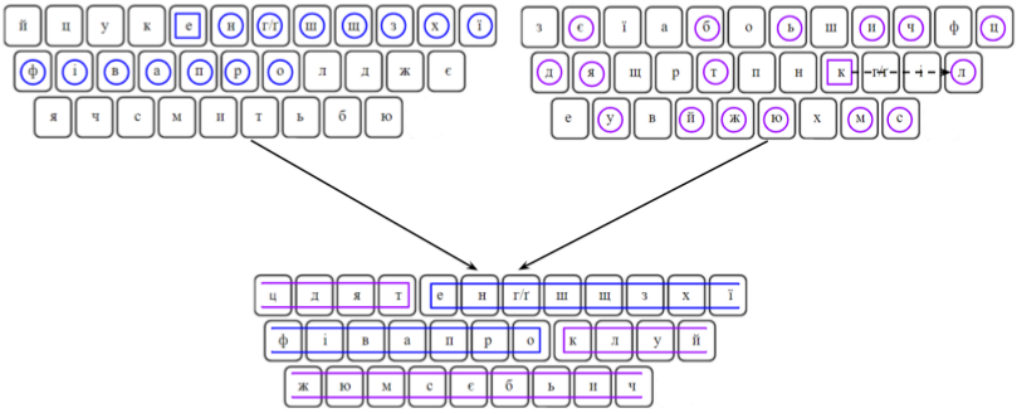
Модель генетичного алгоритму

Популяція, хромосома та ген



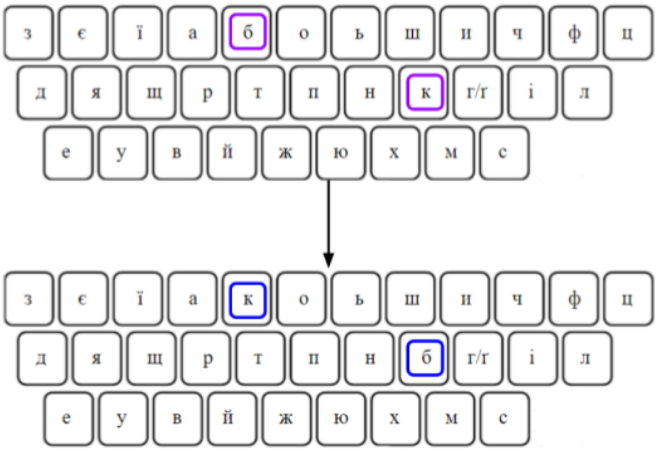
Модель генетичного алгоритму

Схрещування



Модель генетичного алгоритму

Мутація

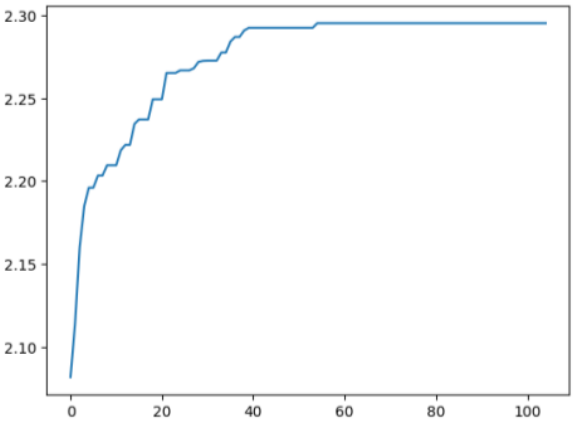


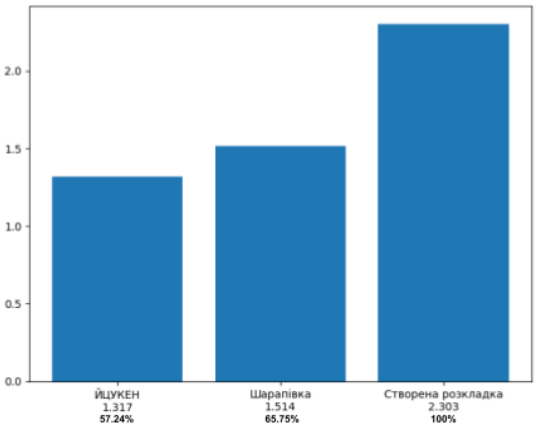
Набір даних

Набір даних може бути заданий користувачем відповідно до його потреб та повсякденного стилю тексту, який користувач використовує.

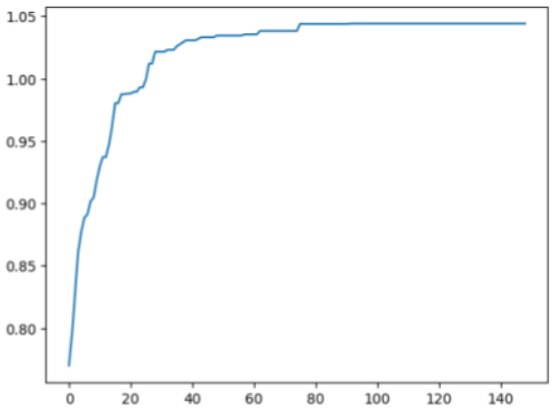
В нашому випадку ми використовуємо статті новин, оскільки вони відображають сучасну українську мову та найпоширеніший стиль тексту при друку - публіцистичний.

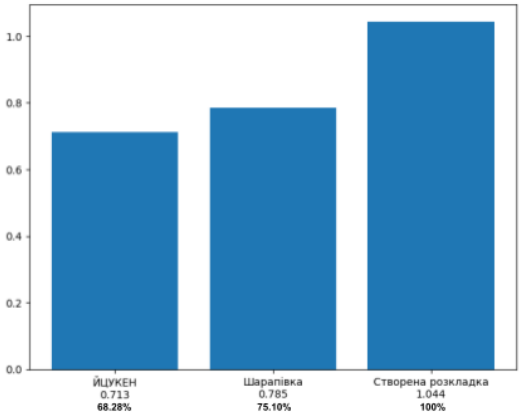
Результат роботи
Сліпий метод





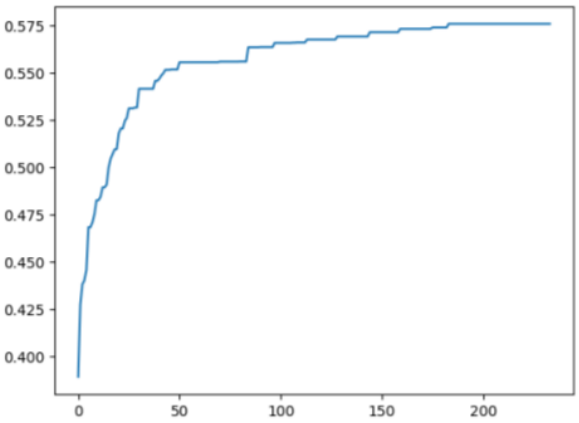
Результат роботи
Метод двох пальців

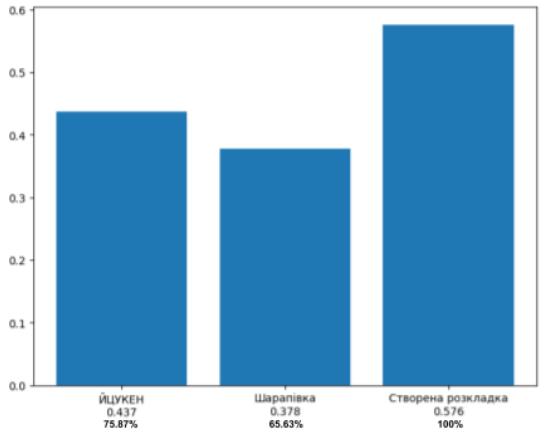




Результат роботи

Метод одного пальця





Висновки

- Створені алгоритмом розкладки показують найбільшу ефективність у порівнянні з іншими аналогами, що дозволяє користувачу збільшити швидкість друку на 25% або більше.
- Необхідна оптимізація алгоритму.
- Потрібно додати можливість створення розкладки для інших абеток.