Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження та порівняння генетичних алгоритмів з відновлення зображень

(тема)

Виконав:

студент II курсу, групи СПм-20-2

Торба О.О
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Каргін А.О
(посада, прізвище, ініціали)

Допускається до захисту

В.о.зав, кафедри ЕОМ _____ Волк М.О
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Торбі Олександру Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та порівняння генетичних алгоритмів відновлення зображень

затверджена наказом по університету від " 24 " березня 2021 р. № 413 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022р

3. Вхідні дані до роботи Тип обладнання – процесор M1 Pro, мова програмування Python

4. Перелік питань, що потрібно опрацювати у роботі

1) огляд літератури за темою роботи;

2) аналіз предметної області;

3) вибір та обґрунтування методики дослідження;

4) проведення експериментальних досліджень;

5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 11 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою роботи	30.03.22-05.04.22	
2	Вибір та обґрунтування методики дослідження	06.04.22-16.04.22	
3	Вибір інструментальних засобів		
4	Проведення експериментів	17.04.22-28.04.22	
5	Оформлення матеріалів атестаційної роботи	29.04.22-05.05.22	
6	Подання кваліфікаційної роботи керівникові та	11.05.22-13.05.22	
	її попередній захист		
7	Подання кваліфікаційної роботи на	14.05.22-18.05.22	
	рецензування		

Дата видачі завдання 28 березня 2022р

Студент _____
(підпис)

Керівник роботи _____ проф. Каргін А.О
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 72 с., 31 рис., 2 дод., 25 джерел.

ГЕНЕТИЧНІ АЛГОРИТМИ, ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ, PYTHON, ОБРОБКА ЗОБРАЖЕНЬ.

Метою кваліфікаційної роботи є дослідження генетичних алгоритмів з реставрування зображень.

Реконструкція зображень з багатокутників – це один з найпопулярніших прикладів застосування генетичних алгоритмів в обробці зображень - реконструкція зображення за допомогою набору напівпрозорих фігур, що перекриваються. Ці експерименти дозволяють використовувати останні досягнення в області аналізу та стиснення зображення.

У ході виконання кваліфікаційної роботи досліджується два способи побудови зображення схожого на оригінал:

1) попіксельна середньоквадратична помилка (СКО);
2) структурна подібність (Structural Similarity – SSIM).

ABSTRACT

72 pages, 31 figures, 2 appendices, 25 sources.

GENETIC ALGORITHMS, IMAGE VISION, PYTHON, IMAGE PROCESSING.

The method of qualification work is the follow-up of genetic algorithms for the restoration of the image.

Reconstruction of an image from rich cutouts is one of the most popular applications of genetic algorithms in the processing of an image - reconstruction of an image for an additional set of inverted shapes that intersect. These experiments allow us to win the rest of the reach in the field of analysis and image compression.

There are two ways to create an image similar to the original in the course of the vikonnanny of the qualification work:

1) pixel by pixel mean square pardon (RMS);
2) structural similarity (SSIM).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MSE – попіксельна середньоквадратична помилка

SSIM – структурна подібність

FPS – пристосований пропорційний відбір (англ, «fitness proportionate selection»)

SUS – стохастична універсальна вибірка (англ, «stochastic universal sampling»)

ВСТУП

Завдання комп'ютерної графіки полягає у обробці інформації, пов'язаної із зображеннями. Вона поділяється на три основних напрямки: візуалізація, обробка зображень і розпізнавання зображень.

Обробка зображень будь-яка форма обробки інформації, для якої вхідні дані представлені зображенням і результат – теж зображенням.

Завданням обробки може бути як поліпшення зображення за яким–небудь певним критерієм, так і перетворення, що кардинально змінює зображення. Різні методи застосовуються для оцифрованих зображень (фотографій, відео) та зображень, отриманих за допомогою комп'ютерної візуалізації. У разі оцифрування можна помітити шум на зображенні.

Важливою і актуальною проблемою досі є відновлення фотографій після їх оцифрування, тобто є потреба у видаленні різного роду шумів, подряпин та інших дефектів. Завдання відновлення зображення полягає в тому, щоб по зображенню знайти більш повні характеристики вихідного об'єкта. На жаль, немає такого універсального методу, який вирішував би це завдання, і вибір методів для її вирішення безпосередньо залежить від того, що саме входитиме в завдання обробки.

У цій роботі досліджуються алгоритми з реставрації та відновлення зображень на основі оригіналу за допомогою генетичних алгоритмів. У ядрі алгоритму реставрації зображення використовуються алгоритми порівняння зображення яке відновлюється і оригіналу, а саме:

1) попіксельна середньоквадратична помилка (СКО);
2) структурне подібність (Structural Similarity - SSIM).

1 АНАЛІЗ ТЕХНОЛОГІЙ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Один з найдивовижніших методів вирішення завдань, який запозичив ідею в теорії еволюції Чарльза Дарвіна, заслужено отримав назву "еволюційні обчислення". Найвідомішими та найпоширенішими представниками цієї родини є генетичні алгоритми. Генетичні алгоритми є одним із найдивовижніших методів вирішення задач пошуку, оптимізації та навчання. Вони можуть призвести до успіху там, де традиційні алгоритми не здатні дати адекватних результатів за прийнятний час.

Генетичні алгоритми – це сімейство пошукових алгоритмів, ідеї яких засновані на принципах еволюції у природі. Імітуючи процеси природного відбору та відтворення, генетичні алгоритми можуть знаходити високоякісні рішення задач, що включають пошук, оптимізацію та навчання. У той же час аналогія з природним відбором дозволяє цим алгоритмам долати деякі перешкоди, пов'язані з традиційними алгоритмами пошуку та оптимізації, особливо в задачах з багатьма параметрами та складними математичними уявленнями.

Генетичні алгоритми реалізують спрощений варіант дарвінської еволюції. Нижче наведено принципи еволюційної теорії Дарвіна [11]:

- мінливість. Ознаки, тобто атрибути, окремих особин, що входять до складу популяції, можуть бути змінені. Тому особини відрізняються один від одного, наприклад на вигляд або поведінці;
- спадковість. Деякі властивості стійко передаються від особи до її нащадків. Тому нащадки схожі на батьків більше, ніж на інших особин, не пов'язаних з ними спорідненістю;
- природний вибір. Зазвичай популяції виборюють ресурси, наявні у навколишньому середовищі. Особи, що мають властивості, краще пристосовані до навколишнього середовища, більш успішні у боротьбі за виживання і привносять більше нащадків у наступне покоління.

Іншими словами, еволюція зберігає популяцію особин, які відрізняються одна від одної. Ті, хто краще пристосований до навколишнього середовища, мають більше шансів на виживання, розмноження та передачі своїх ознак наступному поколінню. Так популяція від покоління до покоління стає дедалі більше пристосованої до довкілля і труднощам, що постають на її шляху.

Важливим механізмом еволюції є схрещування, або рекомбінація, коли нащадок набуває комбінації ознак своїх батьків [15]. Схрещування допомагає підтримувати різноманітність популяції та з часом закріплювати найкращі ознаки. Крім того, важливу роль в еволюції відіграють мутації – випадкові варіації ознак – оскільки вони вносять зміни, завдяки яким популяція час від часу робить стрибок у розвитку.

Мета генетичних алгоритмів – знайти оптимальне вирішення певної задачі. Якщо дарвінська еволюція розвиває популяцію окремих особин, то генетичні алгоритми розвивають популяцію потенційних розв'язків цього завдання, які називаються індивідуумами. Ці рішення ітеративно оцінюються та використовуються для створення нового покоління рішень. Ті, що краще проявили себе під час вирішення завдання, мають більше шансів пройти відбір та передати свої якості наступному поколінню. Так поступово потенційні рішення удосконалюються у вирішенні поставленого завдання.

1.1 Основні компоненти генетичних алгоритмів

Генотип – набір генів, згрупованих у хромосоми. У природі схрещування, відтворення та мутація реалізуються за допомогою генотипу. Коли дві особи схрещуються та виробляють потомство, кожна хромосома нащадка несе комбінацію генів батьків.

У разі генетичних алгоритмів кожному індивідууму відповідає хромосома, що представляє набір генів. Хромосому можна уявити двійковим рядком, де кожен біт відповідає одному гену, як показано на рисунку 1.1.

Рисунок 1.1 – Просте двійкове кодування хромосоми

Популяція. У будь–який час генетичний алгоритм зберігає популяцію індивідуумів – набір потенційних рішень поставленої задачі. Оскільки кожен індивід представлений деякою хромосомою, цю популяцію можна розглядати як колекцію хромосом, як показано на рисунку 1.2.

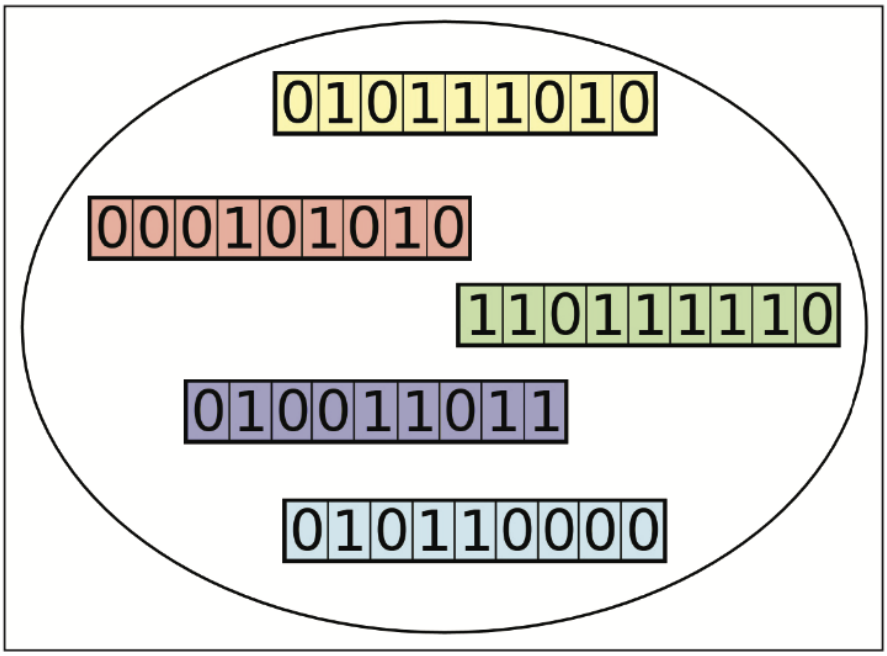

Рисунок 1.2 – Популяція індивідуумів, представлених двійково-кодованими хромосомами

Популяція завжди представляє поточне покоління і еволюціонує з часом, коли поточне покоління замінюється новим.

Функція пристосованості. На кожній ітерації алгоритму індивіди оцінюються за допомогою функції пристосованості, або цільової функції. Це функція, яку потрібно оптимізувати, або завдання, яке необхідно вирішити.

Індивідууми, для яких функція пристосованості дає найкращу оцінку, є кращими рішеннями і з більшою ймовірністю будуть відібрані для відтворення та представлені в наступному поколінні. Згодом якість рішень підвищується, значення функції пристосування зростають, а коли буде знайдено задовільне значення, процес можна зупинити.

Відбір. Після того, як обчислені пристосованості всіх індивідуумів у популяції, починається процес відбору, який визначає, які індивідууми будуть залишені для відтворення, тобто створення нащадків, що утворюють наступне покоління.

Процес відбору ґрунтується на оцінці пристосованості індивідуумів. Ті, чия оцінка вища, мають більше шансів передати свій генетичний матеріал наступному поколінню.

Погано пристосовані індивіди все одно можуть бути відібрані, але з меншою ймовірністю. Таким чином, їхній генетичний матеріал не повністю виключений.

Схрещування. Для створення пари нових індивідуумів батьки зазвичай вибираються з поточного покоління, а частини їх хромосом змінюються місцями (схрещуються), внаслідок чого створюються дві нові хромосоми, що становлять нащадків. Ця операція називається схрещуванням, або рекомбінацією. Приклад схрещування показано на рисунку 1.3.

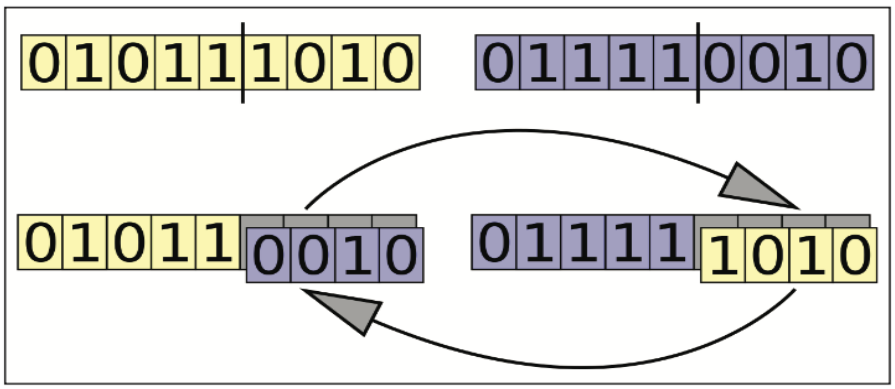

Рисунок 1.3 – Операція схрещування двох двійково-кодованих хромосом

Мутація. Мета оператора мутації – періодично випадковим чином оновлювати популяцію, тобто вносити нові поєднання генів у хромосоми, стимулюючи цим пошук у недосліджених галузях простору рішень.

Мутація може бути випадковою зміною ген. Мутації реалізуються за допомогою внесення випадкових змін у значення хромосом, наприклад, інвертування одного біта в двійковому рядку, як показано на рисунку 1.4.

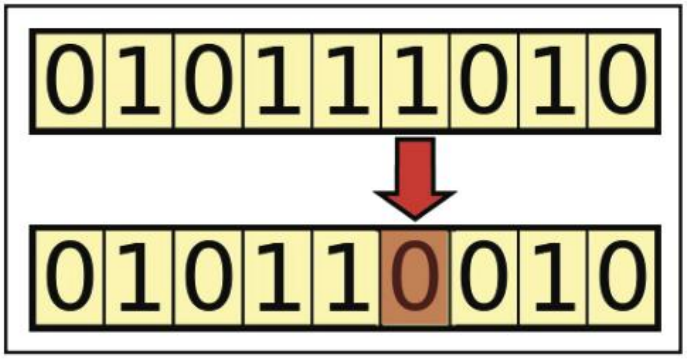

Рисунок 1.4 – Застосування оператора мутації до двійково-кодованої хромосоми

Гіпотеза структурних елементів, що лежить в основі генетичних алгоритмів, полягає в тому, що оптимальне розв'язання задачі може бути зібране з невеликих структурних елементів, і чим більше, тим ближче досягнення оптимального рішення.

Індивідуумам, які містять деякі з бажаних структурних елементів, призначається більш висока оцінка. Повторні операції відбору і схрещування призводять до появи кращих індивідуумів, що передають ці структурні елементи наступному поколінню, можливо, у поєднанні з іншими успішними структурними елементами. Тим самим створюється генетичний тиск, що спрямовує популяцію у бік появи дедалі більшої кількості індивідуумів, що мають структурні елементи, що утворюють оптимальне рішення.

У результаті кожне покоління виявляється кращим за попереднє і містить більше індивідуумів, близьких до оптимального рішення.

1.2 Переваги та недоліки генетичних алгоритмів

До переваг генетичних алгоритмів можна віднести наступне [15]:

1) глобальна оптимізація. У багатьох задачах оптимізації є точки локального максимуму і мінімуму, які представляють рішення, кращі, ніж ті, що знаходяться поблизу, але необов'язково найкращі у глобальному розумінні. Більшість традиційних алгоритмів пошуку та оптимізації, а особливо ті, що ґрунтуються на обчисленні градієнта, можуть застрягати в локальному максимумі, замість знайти глобальний. Це з тим, що у околиці локального максимуму навіть невелика зміна рішення погіршує оцінку. Генетичні алгоритми менш схильні до цієї напасті і мають більше шансів відшукати глобальний максимум. Пояснюється це тим, що використовується популяція потенційних рішень, а не єдине рішення, а операції схрещування та мутації часто породжують рішення, що далеко віддаляються від раніше розглянутих. Це залишається справедливим за умови, що підтримується різноманітність популяції та уникнення передчасної збіжності;

2) застосування генетичних алгоритмів до складних завдань. Оскільки генетичним алгоритмам потрібно знати тільки значення функції пристосованості кожного індивідуума, а решта її властивостей, зокрема похідні, несуттєві, їх можна застосовувати до завдань зі складним математичним уявленням, що включає функції, які важко або неможливо продиференціювати. До складних випадків, коли переваги генетичних алгоритмів розкриваються у всьому блиску, відносяться також завдання з більшим числом параметрів або зі змішаними параметрами, наприклад, безперервними та дискретними;

3) застосування к задачам, що не мають математичного уявлення. Генетичні алгоритми можна застосувати і до завдань, які взагалі не мають математичного уявлення. Один із таких випадків, який має особливий інтерес, це коли оцінка пристосованості ґрунтується на думці людини. Нехай, наприклад, потрібно знайти найбільш привабливу палітру для веб-сайту.

Можна спробувати різні комбінації кольорів та попросити користувачів оцінити привабливість сайту. А потім застосувати генетичний алгоритм, щоб знайти кращу комбінацію, використовуючи функцію пристосованості на основі оцінок користувачів. Алгоритм буде працювати, незважаючи на те, що ніякого математичного уявлення немає і неможливо обчислити оцінку заданої комбінації безпосередньо;

4) стійкість до шуму. Є безліч задач для яких є характерним наявність шуму. Навіть при наявності приблизних значень вхідних параметрів результати їх вимірювань можуть значно відрізнятися. Наприклад, так буває коли дані зчитуються з показників, чи коли оцінка базована на думці людини. Така поведінка може зробити непридатним бага традиційних алгоритмів пошуку, але генетичні алгоритми в загальному випадку стійкі до цього завдяки повторюваним операціям збору та оцінюванню індивідуувім;

5) розпаралелювання. Генетичні алгоритми добре піддаються розпаралелюванню та розподіленої обробці. Функція пристосованості незалежно рахується для кожного індивідууму, що значить, що кожен індивідуум в популяції може оброблятися одночасно. Також операції відбору, схрещування і мутації можуть одночасно виконуватися для індивідуумів та пар індивідуумів. Завдяки цьому підхід заснований на генетичних алгоритмів природно адаптується до розподіленим та хмарних реалізацій;

6) безперервне навчання. В природі еволюція ніколи не зупиняється. Якщо умови змінні, популяція пристосовується до них. Так і генетичні алгоритми можуть безперервно працювати в умовах, що постійно змінюються, і є завжди можливість отримати і використовувати найкраще на даний момент рішення.

Але це можливо тільки якщо навколишнє середовище змінюється повільно порівняно зі швидкістю зміни поколінь у генетичному алгоритмі.

Обмеження та недоліки генетичних алгоритмів [15]:

1) спеціальні визначення. Намагаючись застосувати генетичні

алгоритми до певного завдання, необхідно створити відповідне уявлення – визначити функцію пристосованості та структуру хромосом, а також оператори відбору, схрещування та мутації. Найчастіше це зовсім не просто і займає багато часу. На щастя, генетичні алгоритми вже незліченну кількість разів застосовувалися до найрізноманітніших завдань, отже багато визначень стандартизовані;

2) налаштування гіперпараметрів. Поведінка генетичних алгоритмів контролюється набором гіперпараметрів, наприклад, розміром популяції та швидкістю мутації. Точних правил для вибору значень гіперпараметрів немає. Однак така ситуація практично з усіма алгоритмами пошуку та оптимізації;

3) великий обсяг лічильних операцій. Робота з потенційно великими популяціями та ітеративний характер генетичних алгоритмів обумовлюють великий обсяг обчислень, тому на отримання бажаного результату може бути затрачено багато часу. Таку проблему можна згладити за рахунок гарного вибору гіперпараметрів, розпаралелювання та в деяких випадках кешування проміжних результатів;

4) передчасна збіжність. Якщо пристосованість якогось індивідуума набагато більша, ніж у решти популяції, то не виключено, що такий індивідуум продублюється багато разів, що в кінцевому рахунку, крім нього, в популяції нічого не залишиться. В результаті генетичний алгоритм може застрягли в локальному максимумі і не знайде глобального. Щоб запобігти такому розвитку подій, важливо підтримувати різноманітність популяції;

5) відсутність гарантованих рішень. Використання генетичних алгоритмів не гарантує знаходження глобального максимума. Така поведінка типова для всіх алгоритмів пошуку та оптимізації, якщо тільки у завдання не має аналітичного рішення. Але при правильному застосуванню генетичні алгоритми знаходять задовільне рішення за допустимий час.

1.3 Сценарії використання генетичних алгоритмів

До сценаріїв використання генетичних алгоритмів можна віднести наступні завдання [16]:

1) завдання зі складним математичним уявленням. Оскільки генетичним алгоритмам необхідно знати тільки значення функцій пристосованості, то їх можливо використовувати для вирішення задач, в яких цільову функцію складно або неможливо продиференціювати, задачі з великою кількістю параметрів та задачі з параметрами різних типів;

2) задачі, що не мають математичного уявлення. Генетичні алгоритми не потребують математичного уявлення задачі, тому що скоро можна отримати значення оцінки або існуючого методу порівняння двох рішень;

3) задачі, що мають зашумлене навколишнє середовище. Генетичні алгоритми стійкі до зашумлення даних. До таких даних можна навести показання датчиків, або дані що базовані на оцінках, зроблених людиною;

4) задачі в котрих навколишнє середовище змінюється з часом. Генетичні алгоритми можна адаптуватися к повільним змінам навколишнього середовища, оскільки постійно відтворюють нові покоління, які пристосовуються до змін.

2 СТРУКТУРА ГЕНЕТИЧНИХ АЛГОРИТМІВ

2.1 Базова структура генетичних алгоритмів

На рисунку 1.5 зображена базова структура генетичних алгоритмів [17].

Рисунок 1.5 – Базова структура генетичних алгоритмів

До базових кроків в генетичному алгоритмі можна навести наступні [15]:

1. створення початкової популяції. Початкова популяція містить в собі випадковим шляхом обраних потенційних рішень – індивідуумів. Оскільки в генетичних алгоритмах індивідууми представлені хромосомами, то початкова популяція – це набір хромосом. Формат хромосом повинен відповідати прийнятим для рішення задачі правилам, наприклад, це може бути двійкові строки відповідної довжини;

2. вирахування пристосованості. Для кожного індивідууму розраховується функція пристосованості. Це робиться один раз для початкової популяції, а після для кожного нового покоління після використання операторів відбору, схрещування та мутації. Оскільки пристосованість будь якого індивідууму не залежить від всіх інших, ці обчислення можна виробляти паралельно. Через те, що після розрахування пристосованості більш пристосовані індивідууми зазвичай будуть кращими рішеннями, генетичні алгоритми більше підходять для пошуку максимумів функції пристосованості. Якщо в якійсь задачі потрібен мінімум, то при обчисленні пристосованості необхідно інвертувати находжене значення, наприклад помножив його на -1;

3. використання операторів відбору, схрещування та мутації. Використання генетичних операторів до популяції привиде до створенню нової популяції, яка базується на кращих індивідуумів серед поточних. Оператор відбору відповідає за відбір індивідуумів серед поточній популяції таким чином, що перевага віддається кращим. Оператор схрещування створює нащадка обраних індивідуумів. Як правило для цього обираються два індивідуума, після чого частини їх хромосом міняються містами, в результаті чого створюються дві нові хромосоми, що представляють двох нащадків. Оператор мутації вносить випадкові зміни в один або декілька генів хромосоми новоствореного індивідуума. Зазвичай ймовірність мутації

більш менш мала;

4. перевірка умов зупинення. Може існувати декілька умов, завдяки яким процесс зупиняється. Перший – це коли досягнуто максимальна кількість поколінь. Така умова також дозволяє обмежити час роботи алгоритма та споживання ресурсів системи. Другий – на протязі декількох останніх поколінь немає помітних поркащень. Це можна реалізувати шляхом запам'ятовування кращої пристосованості, досягнутої в кожному поколінні, та порівнюючи кращого поточного значення зі значенням у декількох попередніх поколіннях. Якщо різниця менше заздалегідь заданного порога, то алгоритм можна зупиняти.

Також можуть бути ще такі ймовірні умови зупинення:

- з моменту початку пройшов заздалегідь заданий час;
- перевищено деякий ліміт витрат, наприклад це може бути процесорний час або пам'ять;
- найкраще рішення зайняло частина популяції, більша ніж заздалегідь заданого порогу.

Підсумовуючи, генетичний алгоритм починається з популяції випадково вибраних потенційних рішень, індивідуумів, для яких обчислюється функція пристосованості. Алгоритм виконує цикл, у якому послідовно застосовуються оператори відбору, схрещування та мутації, після чого пристосованість індивідуумів перераховується. Цикл триває, доки не виконана умова зупинки, після чого найкращий індивід у поточній популяції вважається рішенням.

2.2 Методи відбору

Відбір здійснюється на початку кожної ітерації циклу генетичного алгоритму, щоб вибрати з поточної популяції тих індивідуумів, які стануть батьками індивідуумів у наступному поколінні. Відбір носить імовірнісний характер, причому ймовірність вибору індивіда залежить від його

пристосованості, так що у більш пристосованих індивідуумів шанси відібратися вище.

Далі будуть розглянуті деякі з найпоширеніших методів відбору та їх характеристики.

Правило рулетки [17]. Метод відбору за правилом рулетки, або відбір пропорційної пристосованості (англ, fitness proportionate selection – FPS), влаштований так, що ймовірність відбору індивідуума прямо пропорційна його пристосованості. Можна провести аналогію з обертанням колеса рулетки, де кожному індивідууму відповідає сектор, вартість якого дорівнює пристосованості індивідуума. Шанси, що кулька зупиниться в секторі індивіда, пропорційні розміру цього сектора.

Нехай, наприклад, є популяція з шести індивідуумів з такими значеннями пристосованості, як наведено на рисунку 2.1.

Індивідуум	Пристосованість	Відносна доля
A	8	7 %
B	12	11 %
C	27	24 %
D	4	3 %
E	45	40 %
F	17	15 %

Рисунок 2.1 – Популяція з шести індивідуумів

За цими значеннями обчислюються частки, які займають сектори кожного індивідуума, як показана на рисунку 2.2.

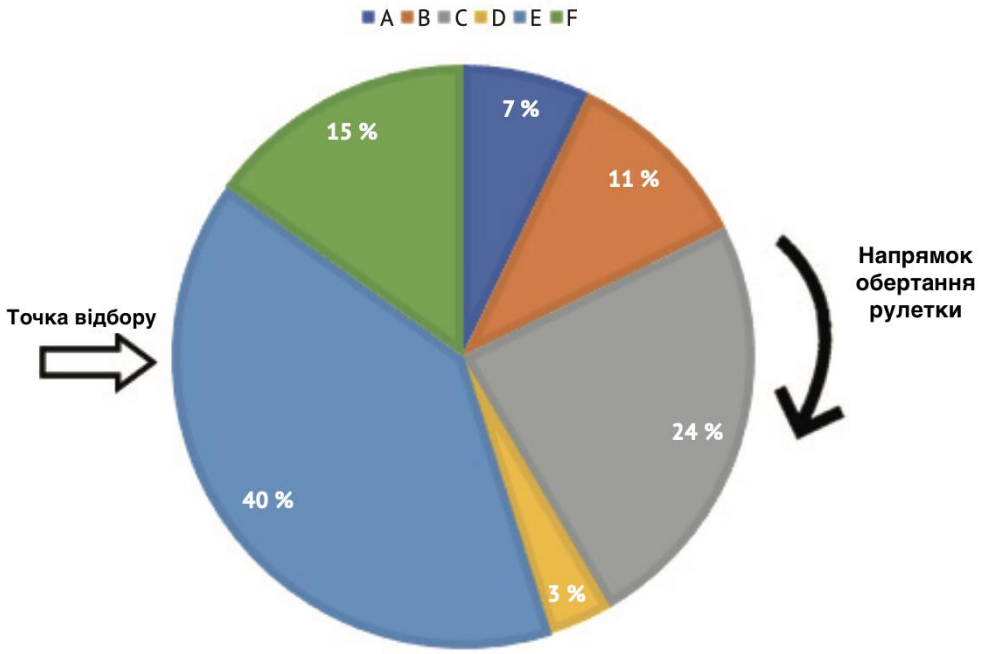

Рисунок 2.2 – Приклад відбору за правилом рулетки

Після кожного запуску рулетки відбір індивіда з популяції проводиться у точці відбору. Потім рулетка запускається ще раз для вибору наступного індивідуума, і так доти, доки не набереться достатньо індивідуумів для утворення наступного покоління. В результаті один і той же індивід може бути обраний кілька разів.

Стохастична універсальна вибірка [15]. Стохастична універсальна вибірка (англ, Stochastic universal sampling – SUS) – трохи модифікований варіант правила рулетки. Використовується та сама рулетка з такими ж секторами, але замість однієї точки відбору та багаторазового запуску рулетки обертається колесо лише один раз, а відбір індивідуумів робиться у кількох точках, рівномірно розставлених по колу. Тим самим усі індивіди вибираються одночасно, як показано на рисунку 2.3. Цей метод відбору не дає індивідуумам з особливо високою пристосованістю заповнити наступне покоління в результаті повторного вибору. Тому слабкішим індивідуумам надається шанс, а несправедливість чистого правила рулетки певною мірою згладжується.

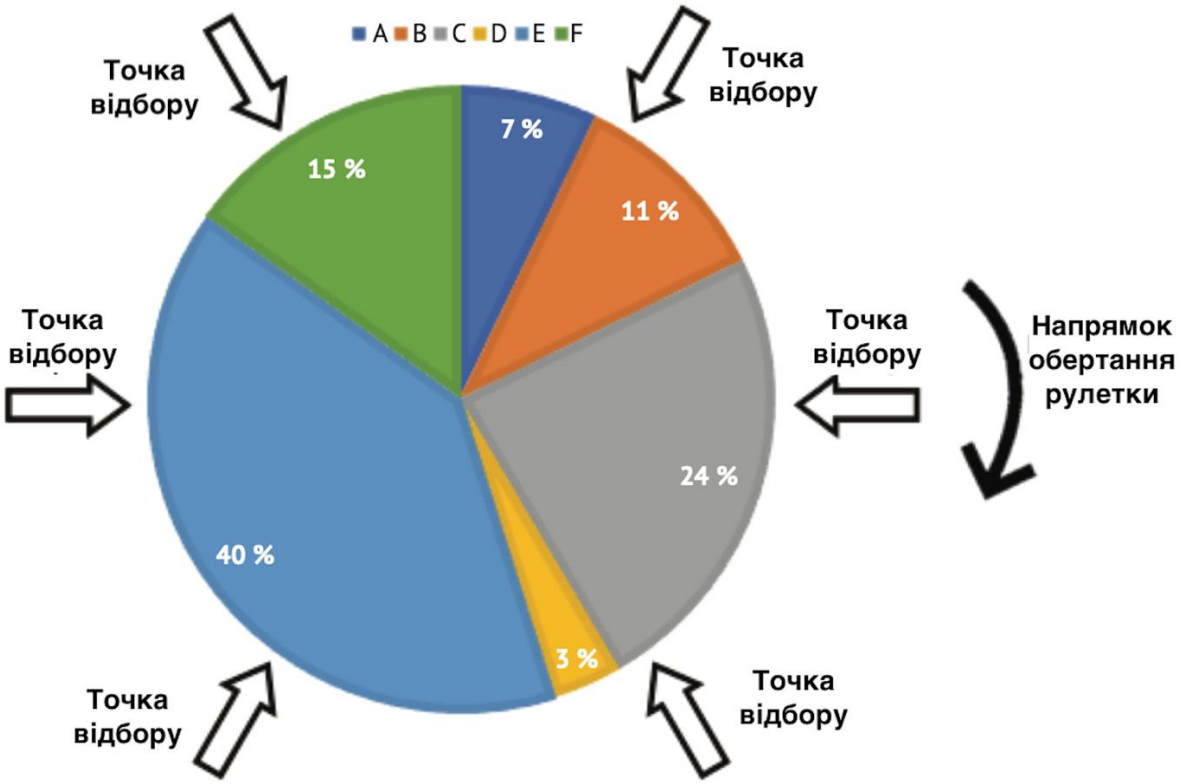

Рисунок 2.3 – Приклад стохастичної універсальної вибірки

Ранжований відбір [17]. Метод ранжованого відбору схожий на правило рулетки, але значення придатності використовуються не для обчислення ймовірностей вибору, а для сортування індивідуумів. Після сортування кожному індивідууму призначається ранг, який відповідає його позиції у списку, а ймовірності секторів рулетки обчислюються на основі цих рангів.

Необхідно взяти ту ж саму популяцію із шести індивідуумів, що й раніше та додати до таблиці стовпець із рангом індивідуума. Оскільки розмір популяції дорівнює 6, найвищий можливий ранг також дорівнює 6, наступний по порядку – 5 і так далі, як показано на рисунку 2.4.

Індивідуум	Пристосованість	Ранг	Відносна доля
A	8	2	9 %
B	12	3	14 %
C	27	5	24 %
D	4	1	5 %
E	45	6	29 %
F	17	4	19 %

Рисунок 2.4 – Нова таблиця з рангом індивідуума

Кожному індивідууму зіставляється сектор рулетки, обчислений за цими рангами, а не за значеннями функції пристосованості. На рисунку 2.5 показана відповідна рулетка.

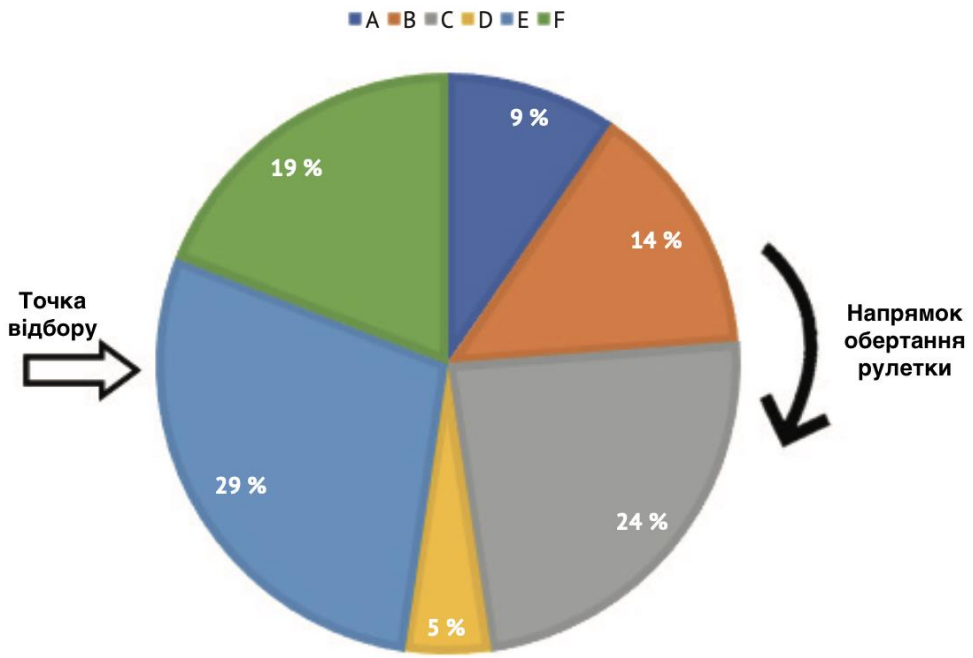

Рисунок 2.5 – Приклад ранжування відбору

Ранжований відбір корисний, коли є кілька індивідуумів, набагато краще пристосованих, ніж решта. Використання рангу замість самої пристосованості заважає цим індивідуумам захопити всю популяцію наступного покоління, оскільки ранжування згладжує значні відмінності.

Крім того, коли всі індивідууми мають майже однакову пристосованість, ранжований відбір дозволяє розділити їх, віддаючи перевагу найкращим, навіть коли відмінності малі.

Масштабування пристосованості [17]. Якщо при ранжованому відборі пристосованості замінюються рангами, то у разі масштабування пристосованості до вихідних значень пристосованості застосовується лінійне перетворення, яке переводить їх у бажаний діапазон:

масштабована пристосованість =

a × (вихідна пристосованість) + b, (2.6)

де a і b – постійні значення, що обираються користувачем.

Якщо необхідно перевести їх до діапазону від 50 до 100, тоді a та b вираховується з рівняння:

50 = a × 4 + b (найменша пристосованість) (2.7)

100 = a × 45 + b (найбільша пристосованість) (2.8)

Вирішуючи цю систему двох лінійних рівнянь, отримуємо такі параметри масштабування:

a = 1.22, b = 45.12

Таким чином, масштабування значень пристосованості вираховується по формулі:

масштабування пристосованості =

1.22 × (вихідна пристосованість) + 45.12 (2.9)

Додавши в таблицю стовпець, що містить значення масштабованої пристосованості, можна переконатися, що значення дійсно потрапляють у діапазон від 50 до 100, як наведено на рисунок 2.6.

Індивідуум	Пристосованість	Масштабування пристосованості	Відносна доля
A	8	55	13 %
B	12	60	15 %
C	27	78	19 %
D	4	50	12 %
E	45	100	25 %
F	17	66	16 %

Рисунок 2.6 – Приклад додавання стовбцю з масштабованою пристосованістю

Перехід до нового діапазону дає набагато більш рівномірне розбиття рулетки на сектори, порівняно з вихідним. Імовірність відбору кращого індивідуума (з масштабованою пристосованістю 100) тепер лише вдвічі вища, ніж гіршого (з масштабованою пристосованістю 50), а не в 11 разів, як спочатку, як показано на рисунку 2.7.

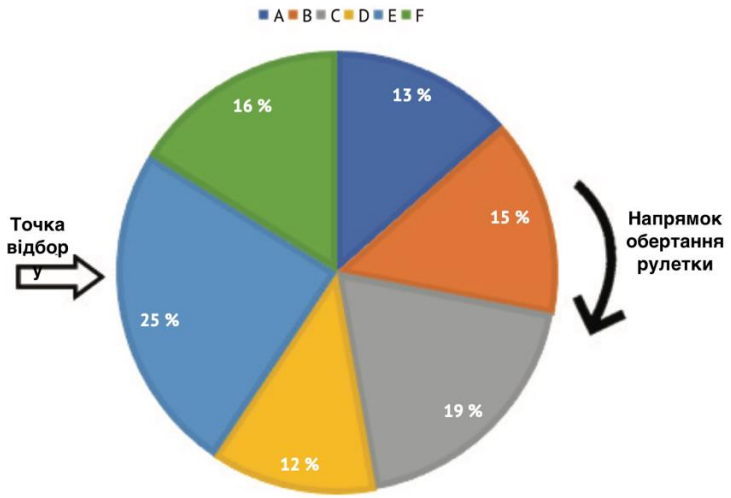

Рисунок 2.7 – Приклад колеса рулетки після масштабування пристосованості

Турнірний відбір [17]. У кожному раунді турнірного відбору з популяції обираються два або більше індивідуумів, і той, у кого більша пристосованість, виграє і відбирається в наступне покоління.

На наступному рисунку 2.8 показаний результат випадкового вибору трьох з них (A, B і F) з наступним оголошенням F переможцем, оскільки у нього максимальна пристосованість з трьох, а саме 17. Кількість індивідуумів, що беруть участь у кожному раунді турнірного відбору (у прикладі – три), називається розміром турніру. Чим більший розмір турніру, тим вищі шанси, що в раундах братимуть участь кращі індивідууми, і тим менше шансів у слабких учасників перемогти у турнірі та відібратися.

Рисунок 2.8 – Приклад турнірного відбору на турнірі з трьома учасниками

Цей метод відбору має цікаву особливість: якщо користувач вміє порівнювати будь-яких двох індивідуумів і визначати, який з них кращий, то самі значення функції пристосованості і не потрібні.

2.3 Методи схрещування

Оператор схрещування, або рекомбінації, відповідає біологічному схрещування при статевому розмноженні. Він використовується для комбінування генетичної інформації двох індивідуумів, які виступають у ролі батьків, у процесі породження нащадків – зазвичай двох.

Як правило, оператор схрещування застосовується не завжди, а з деякою високою ймовірністю. Якщо схрещування не застосовується, копії обох батьків переходять у наступне покоління без зміни.

Далі будуть розглянуті деякі методи схрещування та типові приклади їх застосування. Втім, у деяких ситуаціях доводиться вигадувати спеціальний метод, який відповідає конкретному завданню.

Одноточкове схрещування [14]. При такому схрещуванні позиція в хромосомах обох батьків вибирається випадковим чином. Ця позиція називається точкою схрещування або точкою розрізу. Гени однієї хромосоми, розташовані праворуч від цієї точки, обмінюються з так само розташованими генами іншої хромосоми. В результаті будуть двоє нащадків, які несуть генетичну інформацію обох батьків.

На рисунку 2.9 показано одноточкове схрещування пари двійкових хромосом, коли точка схрещування знаходиться між п'ятим та шостим геном.

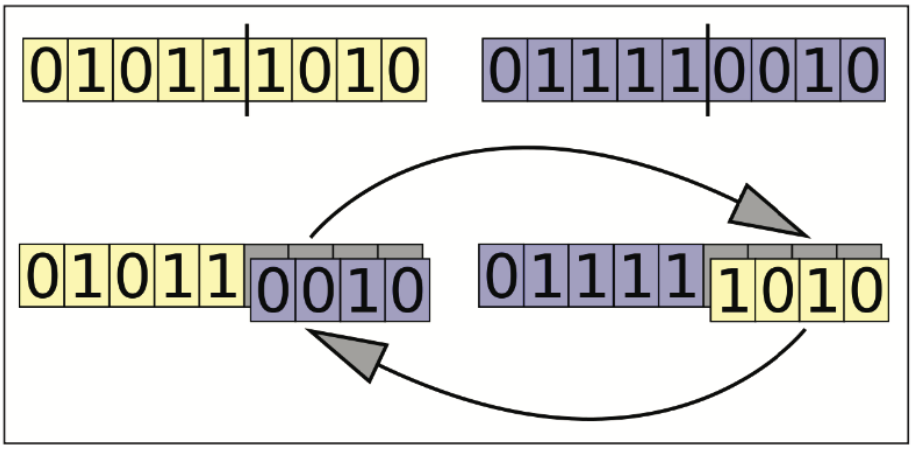

Рисунок 2.9 – Приклад одноточкового схрещування

Двоточкове та k-точкове схрещування [17]. При двоточковому схрещуванні випадково вибираються по дві точки схрещування в кожній хромосомі. Гени однієї хромосоми, розташовані між цими точками, обмінюються з так само розташованими генами іншої хромосоми.

На рисунку 2.10 показано двоточкове схрещування пари двійкових хромосом, коли перша точка схрещування знаходиться між третім та четвертим геном, а друга – між сьомим та восьмим.

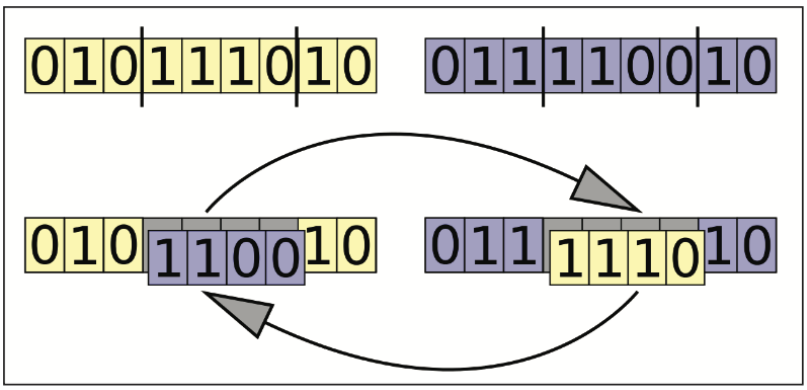

Рисунок 2.10 – Приклад двоточкового схрещування

Метод двоточкового схрещування можна реалізувати за допомогою двох одноточкових схрещувань з різними точками схрещування. Його узагальненням є метод k-точкового схрещування, де k – ціле позитивне число.

Рівномірне схрещування [17]. При рівномірному схрещуванні кожен ген обох батьків визначається незалежно від випадкового вибору з рівномірним розподілом. Коли вибирається 50 % генів, обидва батьки мають однакові шанси вплинути на нащадків. Приклад рівномірного схрещування показано на рисунку 2.11. Треба звернути уваго, що у цьому прикладі гени обох нащадків змінюються місцями, але також нащадків можна створювати незалежно. Оскільки в цьому методі не здійснюється обмін цілих ділянок хромосом, потенційно він може підвищити різноманітність нащадків.

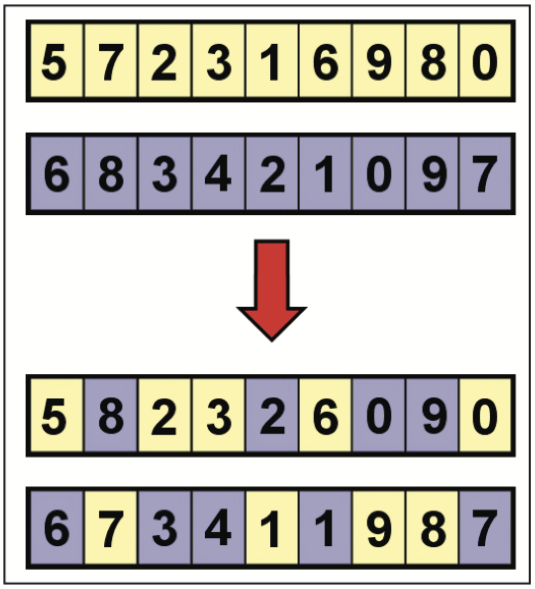

Рисунок 2.11 – Приклад рівномірного схрещування

2.4 Методи мутації

Мутація – останній генетичний оператор, застосовуваний під час створення нового покоління. Він застосовується до нащадка, створеного внаслідок операцій відбору та схрещування.

Операція мутації ймовірна, зазвичай вона виконується зрідка, з дуже низькою ймовірністю, оскільки може погіршити якість індивідуума, до якого застосована. У деяких варіантах генетичних алгоритмів ймовірність мутації поступово збільшується, щоб запобігти стагнації та підвищити різноманітність популяції. З іншого боку, якщо частота мутації дуже велика, то генетичний алгоритм виродиться у випадковий пошук.

Далі буде наведено деякі методи мутації та типові приклади їх застосування.

Інвертування біта [14].

Для двійкової хромосоми є можливість випадково вибрати ген та інвертувати його, тобто взяти його двійкове доповнення, як показано на рисунку 2.12.

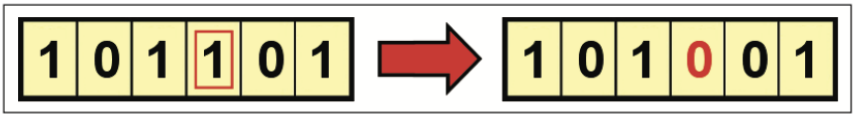

Рисунок 2.12 – Приклад мутації інвертування біту

Можна піти далі та інвертувати не один, а одразу кілька генів.

Мутація обміном [14].

Цей метод застосовується як до двійкових, так і до цілих хромосом: випадково вибираються два гени, і їх значення змінюються місцями, як показано на рисунку 2.13.

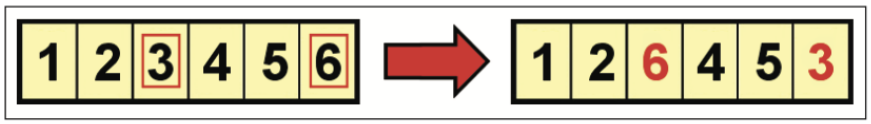

Рисунок 2.13 – Приклад мутації обміном

Така операція мутації придатна для хромосом, що представляють упорядковані списки, оскільки набір генів у новій хромосомі такий самий, як у вихідній.

Мутація зверненням [14].

При застосуванні цього методу до двійкової або цілої хромосоми вибирається випадкова послідовність генів, і порядок генів у ній змінюється на протилежний, як показано на рисунку 2.14.

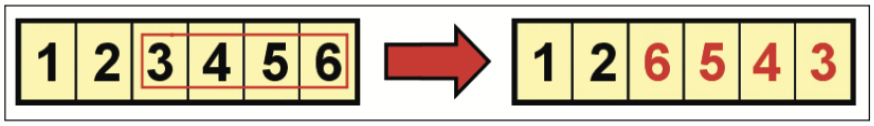

Рисунок 2.14 – Приклад мутації зверненням

Як і мутація обміном, мутація зверненням може використовуватись для хромосом, які мають упорядковані списки.

Мутація перетасовки [14].

Ще один оператор мутації, що підходить для хромосом, що представляють упорядковані списки, – мутація перетасовуванням. У цьому випадку вибирається випадкова послідовність генів, і порядок генів у ній змінюється випадково, тобто тасується, як показано на рисунку 2.15.

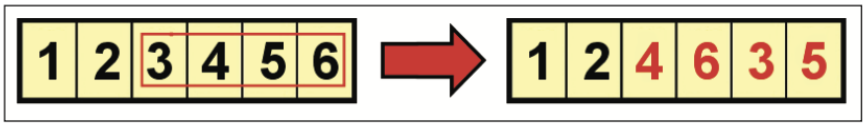

Рисунок 2.15 – Приклад мутації перетасовки

2.5 Елітизм

Хоча середня пристосованість популяції в генетичному алгоритмі, власне кажучи, зростає від покоління до покоління, у будь-який момент може статися так, що найкращі індивіди в поточному поколінні зникнуть. Це пов'язано з тим, що оператори відбору, схрещування та мутації змінюють індивідуумів у процесі створення наступного покоління. У багатьох випадках тимчасова втрата, оскільки ці, або навіть кращі, індивідууми знову з'являться в майбутньому поколінні.

Але якщо потрібно гарантувати, що найкращі індивіди обов'язково переходять у наступне покоління, то можна застосувати факультативну стратегію елітизму [14]. Це означає, що n кращих індивідуумів (n – невелике, заздалегідь задане число) копіюються в наступне покоління, до того як усі місця будуть зайняті нащадками, отриманими в результаті відбору, схрещування та мутації. Скопійовані елітні індивідууми, як і раніше, можуть використовуватися як батьки нових індивідуумів.

Іноді елітизм надає помітний позитивний ефект на якість алгоритму, оскільки не потрібно витрачати час на повторне відкриття хороших рішень, втрачених у результаті еволюції.

Ще один цікавий спосіб покращити результати генетичного алгоритму – скористатися утворенням ніш.

У природі будь-яке довкілля розбивається на кілька менших середовищ, або ніш, зайнятих різними видами, які звертають собі на користь унікальні ресурси ніші, наприклад їжу та укриття. Так, у лісі можна виділити верхівки дерев, підлісок, лісову підстилку, коріння дерев і т. д. У кожній такій ніші мешкають різні види, пристосовані до існування саме в ній.

Якщо в одній ніші мешкають кілька різних видів, то вони конкурують за ті самі ресурси, тому спостерігається тенденція до пошуку нових вільних ніш та їх заселення.

У генетичних алгоритмах феномен ніш можна використовувати для підтримання різноманітності популяції, а також для пошуку кількох оптимальних рішень, кожне з яких вважається нішою.

Нехай, наприклад, генетичний алгоритм має максимізувати функцію пристосованості, що має кілька піків різної висоти, як показано на рисунку 2.16.

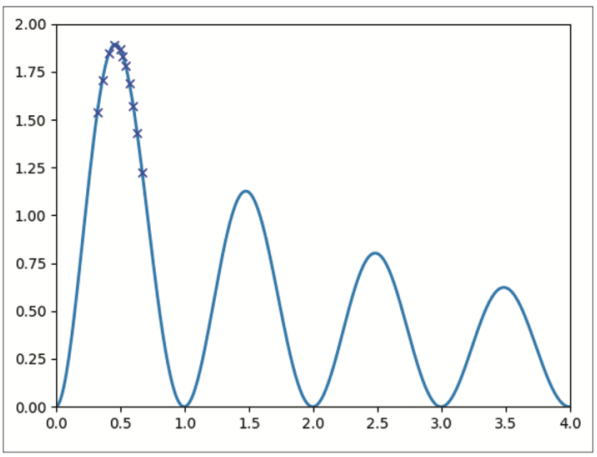

Рисунок 2.16 – Очікувані результати генетичного алгоритму без утворення ніш

Оскільки генетичний алгоритм зазвичай прагне знайти глобальний максимум, можна очікувати, що через деякий час більшість популяції зосередиться в районі верхнього піку. На рисунку 2.20 це показано хрестиками на графіку функції, які представляють індивідуумів у поточному поколінні.

Але іноді потрібно знайти не лише глобальний максимум, а й деякі інші вершини. Для цього потрібно розглянути кожен пік як нішу, що пропонує ресурси обсягом, пропорційним висоті. Далі необхідно шукати спосіб поділити ці ресурси між індивідуумами, що займають нішу. В ідеалі популяція при цьому повинна розподілитися відповідно, тому найвищий пік залучатиме найбільше індивідуумів, оскільки пропонує найбільшу винагороду, а інші піки будуть заселені не так щільно, тому що винагорода в них менша.

Така ідеальна ситуація зображена на рисунку 2.17.

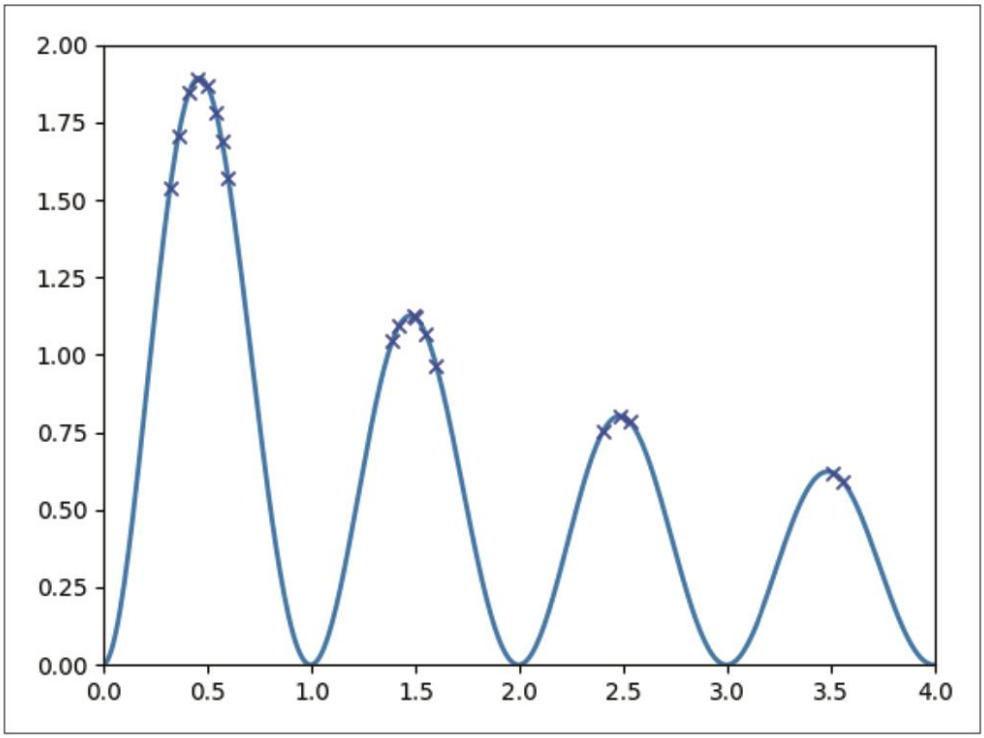

Рисунок 2.17 – Результати ідеального генетичного алгоритму з утворенням ніш

Питання тільки тепер в тому, як реалізувати цей механізм розподілу. Один із варіантів – розділити вихідне значення пристосованості кожного індивідуума на число, рівне якоїсь функції від комбінації відстаней до решти індивідуумів. Інший варіант – розділити вихідну пристосованість кожного індивідуума на число інших індивідуумів, що віддаляються від нього не більше ніж на деяку граничну відстань.

Послідовна та паралельна освіта ніш.

На жаль, описану вище ідею ніш важко реалізувати, тому що при цьому зростає складність обчислення пристосованості. На практиці знадобиться також помножити вихідний розмір популяції на кількість очікуваних піків, яке взагалі невідоме.

Один із способів подолати ці проблеми – шукати піки по одному, тобто послідовна освіта ніш, а не намагатися знайти все одразу, тобто паралельна освіта ніш. Для реалізації послідовної освіти ніш використовується генетичний алгоритм як завжди і знаходиться найкраще рішення. Потім функція пристосованості змінюється в такий спосіб, щоб сплющити околиця точки максимуму, після чого генетичний алгоритм виконується вкотре. В ідеалі знайдеться наступний висотою пік, оскільки вихідного піку більше не існує. Цей процес можна повторювати ітеративно, відшукуючи кожної ітерації черговий пік.

У наступній главі будуть розглянуті обрані рішення для використання генетичних алгоритмів з реставрування зображень.

3 АНАЛІЗ ТЕХНОЛОГІЙ ГЕНЕТИЧНОЇ РЕКОНСТРУКЦІЇ ЗОБРАЖЕНЬ

3.1 Уявлення та оцінювання рішення

Генетичні алгоритми надають потужний та гнучкий інструмент, що дозволяє вирішувати широкий спектр завдань. Приступаючи до нового завдання, необхідно підігнати цей інструмент до його особливостей. Для цього є кілька підходів, які будуть наведені далі.

Насамперед необхідно визначити функцію пристосованості. З її допомогою оцінюються всі індивідууми: чим більша пристосованість, тим краще індивідуум. Функція необов'язково має бути виражена як математичної формули. Вона може бути представлена алгоритмом, або зверненням до зовнішньої служби, або навіть результат гри – і це далеко не всі можливості. Головне, щоб була можливість програмно отримати пристосованість будь-якого рішення – індивіда.

Потім потрібно вибрати відповідний спосіб кодування хромосом, який заснований на параметрах, що передаються функцією пристосованості.

Далі слід визначитися з методом відбору. Більшість методів працюють для хромосом будь-якого типу. Якщо функція пристосованості як така недоступна, але є можливість надати, який із двох кандидатів краще, то можна скористатися турнірним методом відбору.

Вибір операторів схрещування та мутації залежить від кодування хромосом. Для кожного рішення необхідно власноруч вигадувати власні методи схрещування та мутації, що відповідають специфіці завдання.

Нарешті, необхідно пам’ятати про гіперпараметри алгоритма. Найбільш поширені:

- розмір популяції;

- частота схрещування;
- частота мутації;
- максимальна кількість поколінь;
- інші умови зупинки;
- елітизм – використовувати чи ні, та я кого розміру.

Для застосування генетичних алгоритмів в обробці зображень, а саме реконструкції зображень за допомогою набору напівпрозорих багатокутників, було прийняте рішення що до уявлення структури генетичного алгоритма:

1) уявлення індивідуума. У якості індивідуума буде виступати потенційне рішення, яке містить в собі набір з багатокутників у межах зображення. У кожного багатокутника є колір та ступінь прозорості. Все це представлено у якості списка чисел з плаваючою точкою;

2) популяція буде містити в собі набір таких зображень індивідуумів;

3) у якості функції пристосованості буде виступати один з алгоритмів порівняння зображень, які будуть розглянуті детальніше у розділі 3;

4) уявлення оператору відбору. Для оператора відбору було прийняте рішення обрати тип турнірного оператор відбору через його простоту;

5) уявлення оператору схрещування. У якості оператора схрещування через роботу з рішенням, яке представлене списком чисел з плаваючою точкою в обмеженому діапазоні, було прийняте рішення використовувати спеціальну реалізацію імітації двійкового схрещування cxSimulatedBinaryBounded, яка буде розглянуте далі;

6) уявлення оператору мутації. Для оператора мутації був обраний обмежений оператор мутації, в якому розподіл ймовірностей визначається поліноміальною, а не гаусовою функцією. Цей оператор також приймає аргументи low та up – нижню та верхню межі області пошуку. Крім того, він приймає коефіцієнт скупченості, параметр eta, – чим він більший, тим ближчий мутант до вихідного значення. Відповідно, при малих значеннях eta мутант сильно відрізнятиметься від оригіналу.

3.2 Огляд засобів розробки

Набір інструментів, який застосовується при роботі в проектах і включає мови програмування, системи управління базами даних, фреймворки, компілятори і т.д. – це стек технологій. Від обраного розробником стека технологій залежать продуктивність роботи, вимоги до апаратних ресурсів, надійність роботи програмного забезпечення.

Для розробки додатку, який буде застосовувати генетичні алгоритми для реконструкції зображень була обрана мова Python.

Для того щоб полегшити написання, а також тестування та налагодження програмного коду зазвичай використовують спеціальні середовища розробки. Для створення програмного рішення на Python використовується безкоштовне та повнофункціональне середовище розробки – JetBrains PyCharm. Необхідно також додатково встановити Python версії 3.9 або більше на ПК. Для зберігання проекту на віддаленому репозиторії застосовується система контролю версій Git. Для роботи з даною системою використовується один з найбільших веб-сервісів для спільної розробки програмного забезпечення GitHub.

Python – високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій. Основні переваги мови:

1) python у багато разів підвищує продуктивність праці розробника. Обсяг програмного коду на мові Python зазвичай становить третину або навіть п'яту частину еквівалентного програмного коду на мові C ++ або Java. Програми на мові Python запускаються відразу ж, минуючі тривалі етапи компіляції і зв'язування, необхідні в деяких інших мовах програмування, що ще більше збільшує продуктивність праці програміста;

2) програми на мові Python виконується без змін на всіх основних платформах. Перенесення програмного коду між операційними системами

зазвичай полягає в простому копіюванні файлів програм з однієї машини на іншу;

3) безліч готових бібліотек з різних галузей програмування;

4) інтеграція компонентів. Сценарії Python легко можуть взаємодіяти з іншими частинами програми завдяки різноманітним механізмам інтеграції;

5) простота. Python має простий, легкий для читання синтаксис і ясну модель програмування. Python має можливості модульного і об'єктно-орієнтованого програмування, що спрощує можливість багаторазового використання програмного коду.

3.3 Огляд обробки зображень в Python

При розробці програмного забезпечення часто буває так, що використання перевіреної спеціалізованої бібліотеки може полегшити життя. Вона допомагає створювати рішення швидше і з меншою кількістю помилок, а також пропонує на вибір багато відпрацьованих заготовок, замість винаходити велосипед.

В експериментах з реконструкції, найчастіше знаменита картина або її фрагмент, використовується як зразок. Мета – скласти схоже зображення з набору фігур, що перекриваються, зазвичай багатокутників, різного кольору і ступеня прозорості.

Це завдання вирішується за допомогою генетичного алгоритму та бібліотеки deap. Також необхідно малювати зображення та порівнювати їх із зразком.

Для досягнення поставленої мети необхідно виконувати різноманітні операції з зображеннями, наприклад створювати зображення з нуля, малювати на зображенні фігури, виводити зображення на друк, відкривати файл зображення, зберігати зображення у файлі, порівнювати два зображення та іноді змінювати розмір зображення.

Бібліотека Pillow. Pillow – це відгалуження, що досі підтримується,

оригінальної бібліотеки Python Imaging Library (PIL). Ця бібліотека дозволяє відкривати файли зображень різного формата, маніпулювати ними та зберігати в файлах. Pillow була обрана як основний інструмент створення реконструйованого зображення завдяки її здатності малювати фігури, задавати ступінь прозорості та маніпулювати пікселями.

Бібліотека scikit-image. Розширює можливості scipy.image та пропонує різні алгоритми обробки зображень, в тому числі й ввід-вивід, фільтрацію, роботу з кольорами та розпізнавання об'єктів. В подальшому буде використовуватися модуль metrics, який містить в собі засоби порівняння двох зображень.

Бібліотека opencv-python. OpenCV – дуже розвинена бібліотека, яка містить різні алгоритми, що стосуються комп'ютерного зору та машинного навчання. Її версії існують для багатьох мов програмування та платформ. Бібліотека opencv-python – інтерфейс між OpenCV та Python. Вона поєднує швидкість C++ API із зручністю мови Python. Далі вона буде використовувана заради обчислення різниці двох зображень.

Для того щоб намалювати зображення з нуля, використовувались класи Pillow Image та ImageDraw, як показано в лістингу 3.1.

Лістинг 3.1 – Створення зображення з нуля

```
image = Image.new('RGB', (width, height))
draw = ImageDraw.Draw(image, 'RGBA')
```

Тут 'RGB' та 'RGBA' – значення аргументу mode. 'RGB' означає, що кожен піксель є трьома 8-бітовими значеннями – по одному для основних кольорів: червоний ('R'), зелений ('G') і синій ('B'). 'RGBA' додає ще одне 8-бітове значення 'A', що представляє альфа-канал (рівень прозорості). Комбінація базового RGB-зображення та малювання у форматі RGBA дозволяє малювати багатокутники різного ступеня прозорості на чорному тлі.

Необхідно додати до базового зображення багатокутник, викликавши метод polygon класу ImageDraw, як показано в наступному лістингу 3.2, де

малюється трикутник.

Лістинг 3.2 – Додавання багатокутника

```
draw.polygon([(x1, y1), (x2, y2), (x3, y3)], (red, green, blue,
alpha))
```

Кортежі (x1, y1), (x2, y2) та (x3, y3) являють собою три вершини трикутника. Кожен кортеж містить координати x, y однієї вершини.

Кортеж (red, green, blue) – цілі числа в діапазоні [0, 255], що представляють яскравість відповідної компоненту кольору багатокутника. Аргумент alpha – ціле число в діапазоні [0, 255], що представляє ступінь непрозорості багатокутника (чим менше значення, тим прозоріше колір).

Рисунок отриманий в результаті використання модуля Pillow, показаний на рисунку 3.1.

Рисунок 3.1 – Зображення з багатокутників різного кольору та прозорості

3.4 Реалізації генетичного алгоритма на мові Python

Для створення початкової популяції з багатокутників

використовуються засоби модулю Pillow, де потрібно задати наступні параметри:

1. список кортежей [(x1, y1), (x2,y2),...,(xn,yn)], що представляє вершини багатокутника. Кожний кортеж містить в собі координати x, y однієї вершини в системі координат зображення. Завдяки чому значення x належить діапазону [0, ширина зображення - 1], а значення y – діапазону [0, висота зображення - 1];

2. три цілих числа в діапазоні [0, 255], що представляє червону, зелену і синю компоненти кольору багатокутника;

3. останній параметр – ціле число в діапазоні [0, 255], що представляє альфа-канал, або ступінь непрозорості багатокутника.

Це означає, що для кожного багатокутника в наборі необхідно [2x(кількість вершин) + 4] параметрів. Наприклад, для трикутника необхідно 10 параметрів, а для шестикутника – 16 параметрів. Тому набір трикутників буде представлений списком наступного формата, у котрому під кожним трикутником відведено 10 параметрів:

[x11, y11, x12, y, x13, y13, r1, g1, b1, alpha1, x21, y21, x22, y22, x23, y23, r2, g2, b2, alpha2, ...]

Щоб спростити це уявлення, буде використовуватись для кожного параметра числа з плаваючою точкою в діапазоні [0, 1]. Але перед тим як малювати багатокутник, необхідно привести кожен параметр до потрібного діапазону.

При такому поданні набір з 50 трикутників описуватиметься списком з 500 чисел з плаваючою точкою від 0 до 1, наприклад:

[0.1499488467959301, 0.3812631075049196, 0.0004394580562993053, 0.9988170920722447, 0.9975357316889601, 0.9997461395379549, 0.6338072268312986, 0.379170095245514, 0.29280945382368373, 0.20126488596803083,

...

0.4551462922205506, 0.9912529573649455, 0.7882252614083617, 0.01684396868069853, 0.2353587486989349, 0.003221988752732261, 0.9998952203500615, 0.48148512088979356, 0.11555604920908047, 0.08328550982740457]

Для оцінювання такого рішення слід розбити цей довгий список на частини, що відповідають окремим багатокутникам; у разі трикутників довжина кожної частини дорівнює 10. Потім створити нове зображення і намалювати на ньому всі багатокутники один за одним. І нарешті, обчислити ступінь відмінності між зображенням, що вийшло та оригіналом.

Клас, який буде інкапсулювати реконструкцію зображення має назву ImageTest. Для конструювання класу необхідні два параметри: шлях до файлу, який містить в собі зображення–приклад та кількість вершин багатокутників, завдяки яким складається зображення–реконструкція. Клас має наступний функціонал:

1) polygonDataToImage(): отримує список, який містить дані багатокутників, розбиває цей список на частини, що представляють окремі багатокутники та створює зображення, рисуючи ці багатокутники на чистому зображенні;

2) getDefference(): отримує дані багатокутників, створює з них зображення та вираховує різницю між ним та прикладом за допомогою методів вимірювання ступеня відмінності двох зображень;

3) plotImages(): для порівняння розміщує зразок та побудоване зображення на одному малюнку;

4) saveImage(): отримує дані багатокутників, створює з них зображення, створює зображення, яке містить зразок та побудоване зображення, потім зберігає цей малюнок до файлу.

У процесі виконання генетичного алгоритму метод saveImage() буде викликатись через кожні 100 поколінь, щоб можна було стежити за ходом реконструкції.

Для роботи з генетичним алгоритмом були створені цілий ряд каркасів на мові Python, наприклад GAFT, Pyevolve та PyGMO. Вивчаючи різноманітні варіанти, було прийняте рішення зупинитися на каркасі DEAP.

Distributed Evolutionary Algorithms in Python – розподілені еволюційні алгоритми на Python, підтримує швидку розробку рішень із застосуванням генетичних алгоритмів та інших методів еволюційних обчислень. DEAP пропонує різні структури даних та інструменти, необхідні для реалізації різних рішень на основі генетичних алгоритмів.

Для використання вбудованої реалізації генетичного алгоритма з елітизмом модуля DEAP необхідно передати методи зворотного виклику з певним ім'ям, які будуть йому зрозумілі. Для цього використовується каркас base.Toolbox модуля DEAP. Toolbox – контейнер для функцій, або операторів, і дозволяє створювати нові оператори шляхом призначення псевдонімів або налаштування функцій.

Далі буде розглянута реалізація генетичних операторів для реставрування зображення.

Для створення індивідуума у якості зображення використовувався модуль creator пакета DEAP. Creator – визначення індивідуумів, що утворюють популяцію в генетичному алгоритмі, в випадку цієї дипломної роботи індивідуум – це набір багатокутників на зображенні.

Функція пристосованості. Оскільки є ціль знайти мінімізовану різницю зображення–приклада та збудованого з багатокутників необхідно визначити єдину мету – мінімізовану стратегію пристосування, як показано у лістингу 3.3.

Лістинг 3.3 – Мінімізована стратегія пристосування

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
```

Далі йде створення індивідуума, як показано у лістингу 3.4.

Лістинг 3.4 – Створений індивідуум

```
creator.create("Individual", list, fitness=creator.FitnessMin)
```

Після створення індивідуума, необхідно задати оператор заповнення індивідуума, як показано на лістингу 3.5.

Лістинг 3.5 – Створення оператору заповнення індивідуума

```
def random_float(low, up):
          return [random.uniform(l, u) for l, u in zip([low] *
NUM_OF_PARAMS, [up] * NUM_OF_PARAMS)]

toolbox.register("attrFloat", random_float, BOUNDS_LOW,
BOUNDS_HIGH)

toolbox.register("individualCreator",
             tools.initIterate,
             creator.Individual,
             toolbox.attrFloat)
```

Далі йде створення оператору пристосованості, як показано в наступному лістингу 3.6.

Лістинг 3.6 – Створення оператору пристосованості

```
def getDiff(individual):
     return imageTest.getDifference(individual, "MSE"),
toolbox.register("evaluate", getDiff)
```

Метод getDifference() буде використовуватись кожен раз коли генетичному алгоритму потрібно буде дізнатися пристосованість індивідуума. Метод getDifference() звертається до класу ImageTest, щоб дізнатися різницю зображень за допомогою алгоритма MSE.

Попіксельна середньоквадратична помилка (MSE або Mean Square Error) – найпоширеніший спосіб оцінки подібності двох зображень – піксельне порівняння. Зрозуміло, для цього зображення мають бути однакового розміру.

Метрика СКО обчислюється в такий спосіб:

1) обчислити квадрат різниці між кожною парою відповідних пікселів зображень. Оскільки піксель представлений значеннями трьох компонентів –

червоної, зеленої та синьої, – то різниці вважаються за кожним виміром окремо;

2) скласти усі квадрати;

3) розділити суму на загальну кількість пікселів.

В даному програмному додатку зображення представлені засобами бібліотеки OpenCV та обчислення MSE виконуються наступним чином, як показано на лістингу 3.7.

Лістинг 3.7 – Формула обчислення подібності двох зображень за допомогою способу MSE

```
MSE = np.sum((cv2Image1.astype("float") -
cv2Image2.astype("float")) ** 2)/ float(numPixels)
```

Якщо обидва зображення однакові, то СКО дорівнює нулю. Тому як цільову функцію алгоритму можна прийняти мінімізацію цієї метрики.

Структурна подібність (SSIM або Structural Similarity Index Measure) – індекс структурної подібності використовується для передбачення якості зображення, створеного даним алгоритмом стиснення за допомогою його порівняння з вихідним. На відміну від MSE, метод SSIM обчислює не абсолютну величину помилки, а ґрунтується на сприйнятті та враховує зміни структурної інформації, а також таких властивостей зображення, як яскравість та текстура.

У модулі metrics з бібліотеки scikit-image є функція, яка обчислює індекс структурної подібності двох зображень. Через те, що обидва зображення будуть представлені у форматі бібліотеки OpenCV, цю функцію можна викликати безпосередньо, як показано у лістингу 3.8.

Лістинг 3.8 – Приклад виклику функції SSIM

```
SSIM = structural_similarity(cv2Image1, cv2Image2)
```

Функція structural_military() повертає число з плаваючою точкою в діапазоні [-1, 1], це і є індекс структурної подібності. Значення 1 означає, що обидва зображення однакові.

За промовчанням функція порівнює напівтонові зображення. Для порівняння кольорових зображень потрібно задати необов'язковий аргумент multi-channel рівним True.

Далі був створений оператор відбору. У якості оператору відбору був обраний турнірний оператор розміром 2, як показано на лістингу 3.9.

Лістинг 3.9 – Створення турнірного оператору відбору

```
toolbox.register("select", tools.selTournament, tournsize=2)
```

Така схема добре працює разом з елітичним підходом.

Для оператору схрещування був обраний спеціальний оператор каркаса DEAP, як показано на лістингу 3.10.

Лістинг 3.10 – Створення оператора схрещування

```
toolbox.register("mate", tools.cxSimulatedBinaryBounded,
low=BOUNDS_LOW, up=BOUNDS_HIGH, eta=CROWDING_FACTOR)
```

Імітований двійковий кросовер (SBX, або Simulated Binary Crossover) – перехресний оператор, потужність пошуку якого подібна до одноточкового кросовера. використовується в генетичних алгоритмах з двійковим кодом.

Для оператору мутації теж був обраний спеціальний оператор каркаса DEAP, як показано на лістингу 3.11.

Лістинг 3.11 – Створення оператору мутації

```
toolbox.register("mutate", tools.mutPolynomialBounded,
low=BOUNDS_LOW, up=BOUNDS_HIGH, eta=CROWDING_FACTOR, indpb=1.0 /
NUM_OF_PARAMS)
```

Усі оператори будуть передаватися до генетичного алгоритму с елітизмом, як показано в лістингу 3.12.

Лістинг 3.12 – Оператори які передаються до генетичного алгоритму

```
population, logbook = elitism.eaSimpleWithElitism(population,
toolbox, cxpb=P_CROSSOVER, mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, halloffame=hof, verbose=True)
```

3.5 Результати виконання генетичного алгоритма

Для тестування програми було обране відоме зображення персонажу Mario, яке показано на рисунку 3.2.

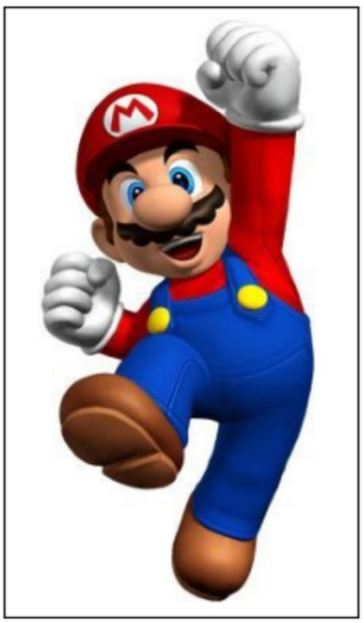

Рисунок 3.2 – Зображення оригінал для тестування програми

Для реконструкції зображення було взято 200 трикутників, розмір популяції 200 та 100 тисяч поколінь.

Результат алгоритму з використання попіксельної середньоквадратичної помилки наведені далі.

Перше покоління показано на рисунку 3.3.

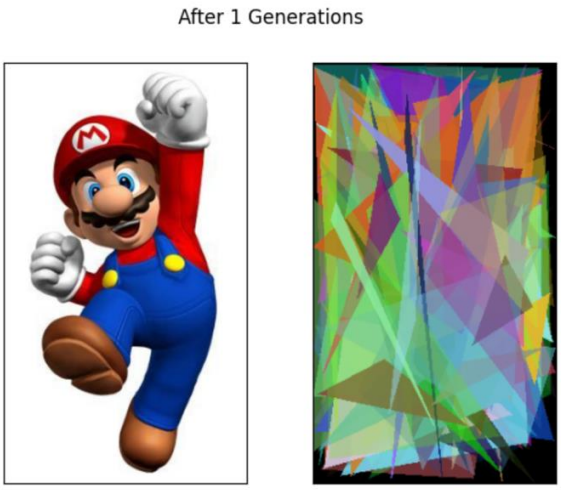

Рисунок 3.3 – Перше покоління еволюції

Результат еволюції 100 покоління показано на рисунку 3.4.

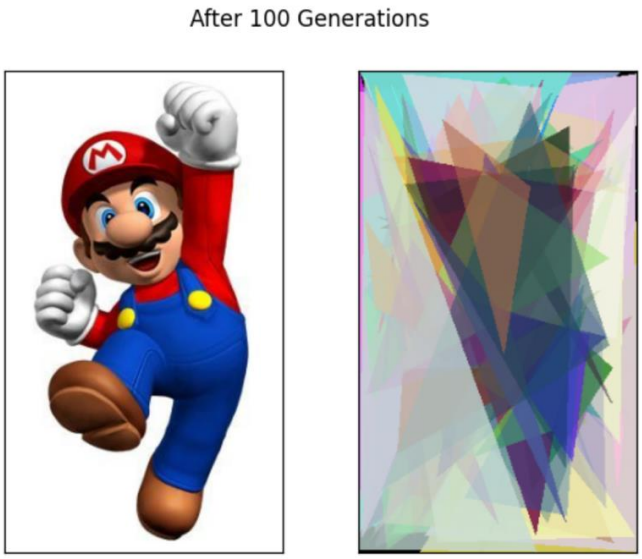

Рисунок 3.4 – Результат еволюції 100 поколінь

Результат еволюції 1500 покоління показано на рисунку 3.5.

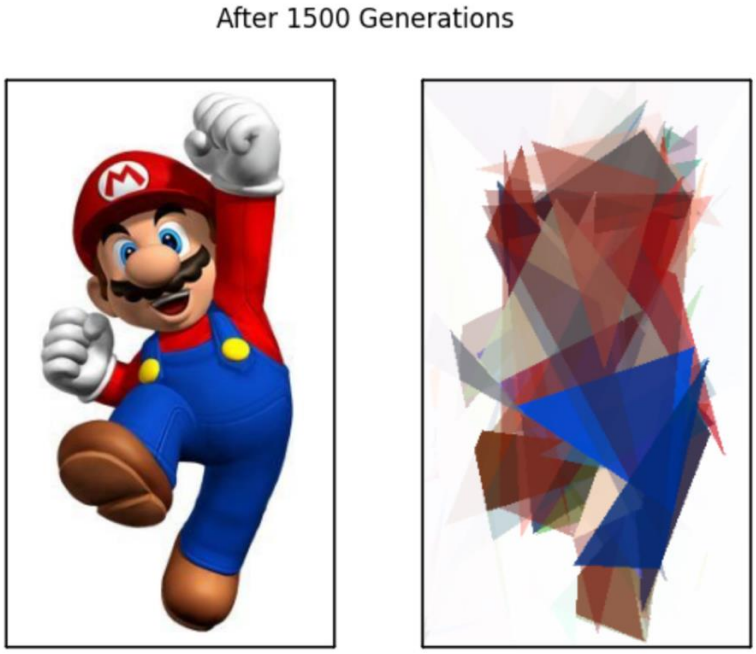

Рисунок 3.5 – Результат еволюції 1500 поколінь

Результат еволюції 5000 поколінь показано на рисунку 3.6.

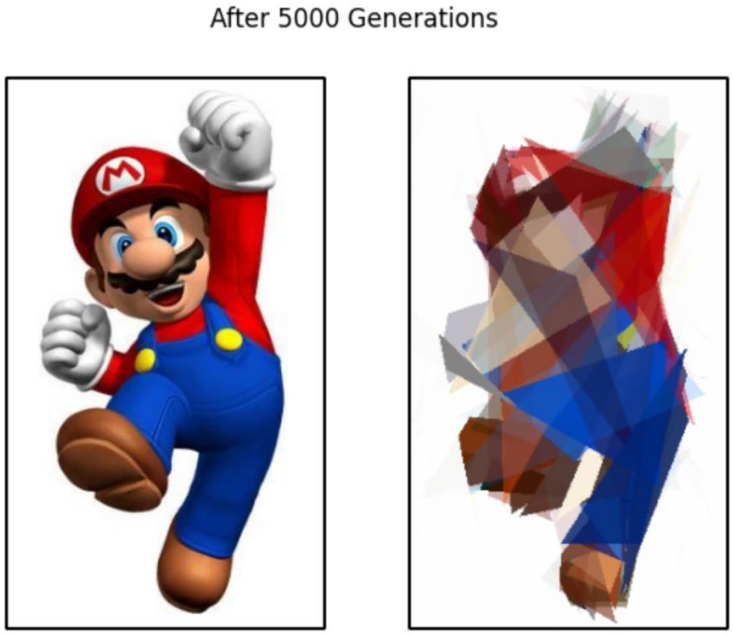

Рисунок 3.6 – Результат еволюції 5000 поколінь

Результат еволюції 25000 поколінь показано на рисунку 3.7.

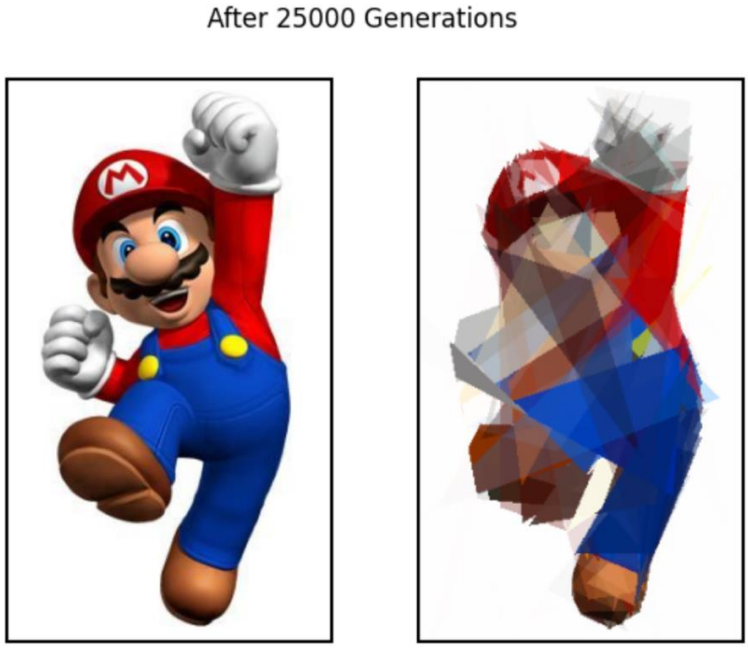

Рисунок 3.7 – Результат еволюції 25000 поколінь

Результат еволюції 50000 поколінь показано на рисунку 3.8.

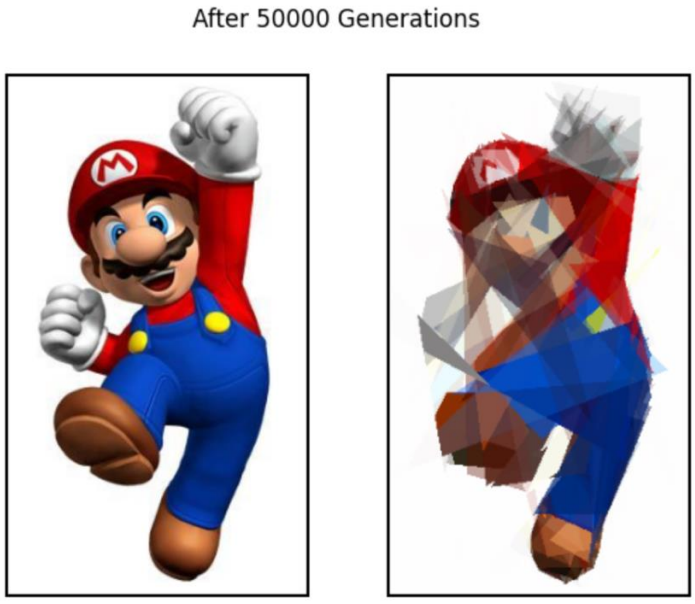

Рисунок 3.8 – Результат еволюції 50000 поколінь

Кінцевий результат еволюції 100000 поколінь показано на рисунку 3.9.

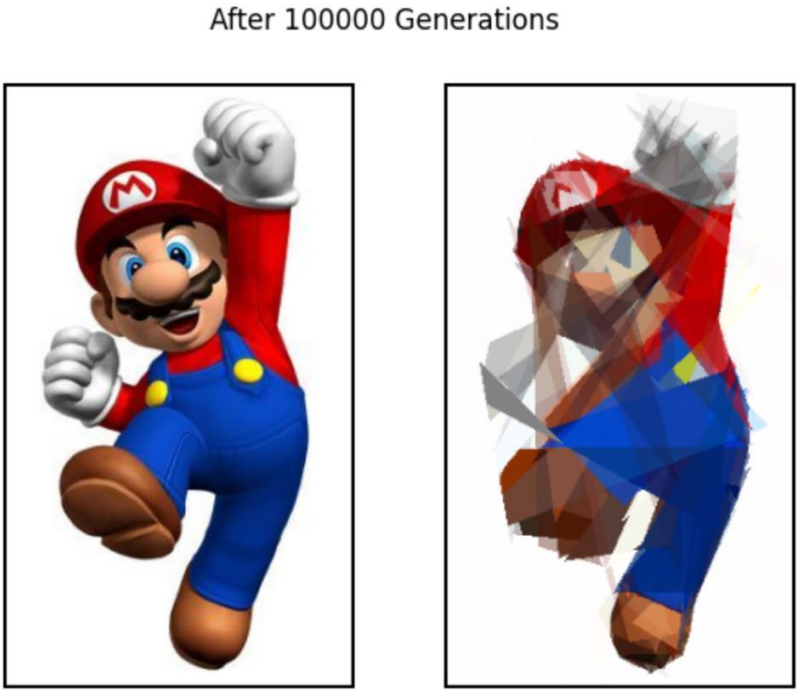

Рисунок 3.9 – Кінцевий результат еволюції 100000 поколінь

ВИСНОВКИ

Дана робота була присвячена розробці та дослідженню генетичних алгоритмів з реставрування зображень.

При проведенні аналізу існуючих програмних рішень, до відчутних недоліків можна віднести маленьку кількість існуючої інформацію та схожих приладів.

Кінцевий результат схожий на оригінал, хоча містить гострі кути та прямі лінії – але цього слід очікувати від зображення, складеного з багатокутників. Якщо дивитися на зображення, примруживши, то ці дефекти згладжуються.

До недоліків додатку можна віднести великий обсяг часу, що необхідно потратити для отримання значного результату роботи генетичного алгоритма.

Задачі, що були поставлені – виконані в повному обсязі. Подальший розвиток дослідження та проекта передбачає підтримку використання інших алгоритмів порівняння зображень, додавання клієнтського інтерфейсу та розпаралелювання процесу генетичного пошуку найкращого зображення, схожого на оригінал.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кукарцев, В. В. Порівняння систем контролю версій: Git, Mercurial, CVS і SVN [Текст] / В. В. Кукарцев, С. А. Бадарчи // Синергия наук. – 2018. – №19. − С. 538-548. 1. Кукарцев, В. В. Порівняння систем контролю версій: Git, Mercurial, CVS і SVN [Текст] / В. В. Кукарцев, С. А. Бадарчи // Синергия наук. – 2018. – №19. − С. 538-548.

2. Вирт, Н. Алгоритми та структури даних. Нова версія для Оберона [Текст] / Н. Вирт. – М.: ДМК Пресс, 2010. – 410 с.

3. Puntambekar, A. A. Software Engineering [Текст] / A. A. Puntambekar. Technical Publications, 2009. – 332 p.

4. Буч Г., Рамбо Д., Якобсон І. Мова UML. Інструкція користувача. 2-е вид. [Текст]: пер. з англ. Мухін Н. –. М.: ДМК Прес. – 496 с.

5. 7. Таненбаум Э., Остин Т. Т18 Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил. ISBN 978-5-496-00337-7.

6. Фаулер М. Рефакторинг: улучшение существующего кода. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 432 с., ил. ISBN 5-93286-045-6.

7. Мартин Р**.** Чистая архитектура. Искусство разработки программного обеспечения. — СПб.: Питер, 2018. — 352 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-0772-8.

8. Swaroop Chitlur A byte of Python.

9. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с., ил. ISBN 978-5-93286-159-2.

10. Clean Python: Elegant Coding in Python Sunil Kapil Sunnyvale, CA, USA ISBN-13 (pbk): 978-1-4842-4877-5.

11. Immunocomputing: principles and applications / Alexander O. Tarakanov, Victor A. Skormin, Svetlana P. Sokolova. ISBN 0-387-95533-X.

12. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour Proceedings of the AISB 2004 Symposium onThe Immune System and

Cognition ISBN 1 902956 38 2.

13. Wang Z., Bovik A. C., Sheikh H. R. & Simoncelli E. P. (2004) «Image quality assessment: From error visibility to structural similarity», *IEEE Transactions on Image Processing*, 13, 600-612:

14. Amita Kapoor, Amey Varangaonkar, N.Morris, K.Thakkar (2019) “Hands-On Artificail Intelligence for LoT”, Published by Packt Publishing Ltd. Livery Place 35 Livery Strret Birmingham B3 2PB, UK. ISBN 978-1-7883-606-7.

15. Alberto Artasanchez, Prateek Joshi “Artificial Intelligence with Python” Published by Packt Publishing Ltd. Livery Place35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-83921-953-5

16. Applied Reinforcement Learning with Python: With OpenAI Gym, Tensorf low, and Keras Taweh Beysolow II San Francisco, CA, USA ISBN-13 (pbk): 978-1-4842-5126-3 ISBN-13 (electronic): 978-1-4842-5127-0

17. Dmitry Zinoviev “Python Companion to Data Science” Printed in the United States of America. ISBN-13: 978-1-68050-184-1

18. Giuseppe Nicosia Vincenzo Cutello Peter J. Bentley Jon Timmis (Eds.) Artificial Immune Systems Third International Conference, ICARIS 2004 Catania, Sicily, Italy, September 13-16, 2004 Proceedings.

19. Hugues Bersini Jorge Carneiro (Eds.) Artificial Immune Systems 5th International Conference, ICARIS 2006 Oeiras, Portugal, September 4-6, 2006 Proceedings ISSN ISBN-10 ISBN-13 0302-9743 3-540-37749-2 Springer Berlin Heidelberg New York 978-3-540-37749-8 Springer Berlin Heidelberg New York.

20. Leandro Nunes de Castro Fernando José Von Zuben Helder Knidel (Eds.) Artificial Immune Systems 6th International Conference, ICARIS 2007 Santos, Brazil, August 26-29, 2007 Proceedings.

21. Peter J. Bentley Doheon Lee Sungwon Jung (Eds.) Artificial Immune Systems 7th International Conference, ICARIS 2008 Phuket, Thailand, August 10-13, 2008 Proceedings.

22. Paul S. Andrews Jon Timmis Nick D.L. Owens Uwe Aickelin Emma Hart Andrew Hone Andy M. Tyrrell (Eds.) Artificial Immune Systems 8th

International Conference, ICARIS 2009 York, UK, August 9-12, 2009 Proceedings.

23. Zhengshan L. (Ed.) (2003). The Study of Non-linearity Science and its Application in the Geosciences. Beijing, CN: Meteorological Press.

24. Yang, J., & Honavar, V. (1998). Feature Subset Selection Using a Genetic Algorithm. *IEEE Trans. on Intelligent Systems*, *13*, 44–49.

25. Yoo, J., & Hajela, P. (1999). Immune network simula- tions in multicriterion design. *Structural Optimization, 18*, 85-94.