Міністерство освіти і науки України
Державний ВНЗ «Національний гірничий університет»

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

напрям підготовки *122 Комп'ютерні науки*
(код і назва напряму підготовки)

спеціальність *Інформаційні управляючі системи та технології*
(код і назва спеціальності)

освітній рівень *магістр*
(назва освітнього рівня)

кваліфікація *інженер з комп'ютерних систем*
(назва кваліфікації)

на тему: *Дослідження оцінки ефективності нейронних мереж при обробці зображення*

Виконавець:
студент *2* **курсу , групи** *122М-16-1*

(підпис) *Колотилін Д.Е.* (прізвище та ініціали)

Керівники	**Посада, прізвище, ініціали**	**Оцінка**	**Підпис**
проекту	*д.т.н., проф. Алексєєв М.О.*		
розділів:			
Спеціальний			
Економічний	*к.е.н., доц. Касьяненко Л.В.*		
Рецензент			
Нормоконтроль	*к.т.н., доц. Коротенко Л.М.*		

Дніпро
2018

Міністерство освіти і науки України
Державний ВНЗ «Національний гірничий університет»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 18 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *122 «Комп'ютерні науки»*
(код і назва спеціальності)

студенту *122М-16-1* *Колотиліну Д.Є.*
(група) (прізвище та ініціали)

Тема дипломного проекту *Дослідження оцінки ефективності нейронних мереж при обробці зображення*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора Державного ВНЗ «НГУ» від *26. 12 .2017 р.* № *2127 -л*

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – функція розпізнання - функція оцінки, яка визначає міру відповідності.

Предмет досліджень – об'єкти робочого простору, полігони, точки, їх колірні властивості розташування протягом життєвого циклу програми.

Мета НДР – дослідження ефективності розпізнання за допомогою нейронних мереж.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів, що очікуються, полягає у визначенні необхідного методу.

Практична цінність результатів полягає у: визначенні необхідного методу.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати магістерської роботи повинні відповідати вимогам паспорту наукової спеціальності 05.13.06 – «Інформаційні технології».

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання методики розробки систем штучного інтелекту (нейронних мереж) та теорії перетворень. Згідно виробничих функцій та професійних задач магістра, повинна бути розроблена ефективна структура нелінійної моделі та алгоритм її адаптації при ідентифікації нелінійних нестаціонарних об'єктів.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок-кінець)**
1	2
Аналіз стану питання	10.09.2017 25.09.2017
Основи нейронних мереж	30.09.2017 18.10.2017
Конфігурація та результати моделювання	25.10.2017 23.11.2017

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки продажу ліцензії додатку на певний період.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки зниженню навантаження на дизайнерів на виконання цільового плану

робіт.

7 ДОДАТКОВІ ВИМОГИ

Завдання видав _____ *Алексєєв М.О.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Колотилін Д.Є.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *. .2017 р.*

Термін подання дипломного проекту до ДЕК *. .2018*

Реферат

Пояснительная записка:93 с., 16 рис., 97 источника.

Объект исследования: технологии генетической алгоритмизации программного обеспечения.

Цель магистерской работы: создание, исследование, наблюдение за поведением мутации и естественного отбора генов программного окружения информационных систем с помощью внедрения технологии Genetic Programming.

Методы исследования. При решении поставленной задачи использовались научные достижения в областях разработки информационных систем и программного обеспечения.

Научная новизна полученных результатов состоит в проведении анализа и выявлении недостатков поведения геномов в изолированной среде, скрещивание, мутацией объектов окружения, а также в использовании методики алгоритмизации информационных систем на основе использования технологии генетической алгоритмизации.

Практическое значение работы заключается векторизации растровой графики с использованием технологии генетической алгоритмизации и естественного отбора объектов, логирование всех этапов отсеивания сильных популяций.

Область применения. Разработанная информационная система может применяться для решения широкого спектра задач, в частности, для создания растровой графики, оптимизации файловых хранилищ.

Значение работы и выводы. Такого рода приложения позволяют решать проблемы с пикселизацией различного рода графических объектов, при модификации, масштабируемости, уход от растровых изображений в разных

сферах деятельности, что подтверждается разработанным программным продуктом в данной магистерской работе.

Прогнозы по развитию исследований. Разработать универсальные программные алгоритмы, оптимизация, которые могут быть использованы для перехода на векторную графику информационных систем из различных сфер рода деятельности. Разработать программное средство и пользовательский интерфейс для графического представления результатов, сравнительного анализа входных и выходных данных.

В разделе «Экономика» проведены расчеты трудоемкости разработки программного обеспечения, расходов на создание ПО и длительности его разработки.

Список ключевых слов: ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ, ПРЯМОЕ КОДИРОВАНИЕ, КОСВЕННОЕ КОДИРОВАНИЕ, СКРЕЩИВАНИЯ, МУТАЦИЯ, ЕСТЕСТВЕННЫЙ ОТБОР, ФУНКЦИЯ ПРИСПОСОБЛЕННОСТИ**,** ГЕНЕТИЧЕСКИЕ ОПЕРАЦИИ**,** ГЕНОТИП, ГЕНОМ, ФИТНЕСС-МЕТОД, SOLUTIONS**,** ПРОГОНЫ, ГЕНЕРАЦИЯ ПОКОЛЕНИЙ

Реферат

Пояснювальна записка : 93 стор., 16 мал., 97 джерел.

Об'єкт дослідження: технології генетичної алгоритмізації програмного забезпечення.

Мета магістерської роботи: створення, дослідження, спостереження за поведінкою мутації і природного добору генів програмного оточення інформаційних систем за допомогою впровадження технології Genetic Programming.

Методи дослідження. При вирішенні поставленого завдання використовувалися наукові досягнення в областях розробки інформаційних систем і програмного забезпечення.

Наукова новизна отриманих результатів полягає в проведенні аналізу та виявленні недоліків поведінки геномів в ізольованому середовищі, схрещування, мутацією об'єктів оточення, а також у використанні методики алгоритмізації інформаційних систем на основі використання технології генетичної алгоритмізації.

Практична цінність полягає векторизації растрової графіки з використанням технології генетичної алгоритмізації і природного відбору об'єктів, логирование всіх етапів відсіювання сильних популяцій.

Область застосування. Розроблена інформаційна система може застосовуватися для вирішення широкого спектра завдань, зокрема, для створення растрової графіки, оптимізації файлових сховищ.

Значення роботи та висновки. Такого роду програми дозволяють вирішувати проблеми з пікселізацією різного роду графічних об'єктів, при модифікації, масштабованості, відхід від растрових зображень в різних сферах діяльності, що підтверджується розробленим програмним продуктом в даній магістерській роботі.

Прогнози щодо розвитку досліджень. Розробити універсальні програмні алгоритми, оптимізація, які можуть бути використані для переходу на векторну графіку інформаційних систем з різних сфер роду діяльності. Розробити програмний засіб і призначений для користувача інтерфейс для графічного представлення результатів, порівняльного аналізу вхідних і вихідних даних.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПО і тривалості його розробки.

Список ключових слів: ГЕНЕТИЧНЕ ПРОГРАМУВАННЯ, ПРЯМЕ КОДИРОВАНИЕ, ОПОСЕРЕДКОВАНЕ КОДИРОВАНИЕ, СХРЕЩУВАННЯ, МУТАЦІЯ, ПРИРОДНИЙ ДОБІР, ФУНКЦІЯ ПРИСТОСУВАНЬ, ГЕНЕТИЧНІ ОПЕРАЦІЇ, ГЕНОТИП, ГЕНОМ, ФІТНЕС-МЕТОД, SOLUTIONS, ПРОГОНИ, ГЕНЕРАЦІЯ ПОКОЛІНЬ

Abstract

Explanatory note: 93 p., 16 fig, 97 sources.

Object of research: technology of genetic algorithmization of software.

The purpose of the degree project: creation, research, observation of mutation behavior and natural selection of the genes of the software environment of information systems through the introduction of Genetic Programming technology.

Methods of research. At the decision of the task in view, scientific achievements in the fields of development of information systems and software were used.

The scientific novelty the results obtained include analysis and identification of shortcomings in the behavior of genomes in an isolated environment, crossing, mutation of environmental objects, as well as using the algorithm of information systems based on the use of genetic algorithmization technology.

The practical value of work is the vectorization of raster graphics using the technology of genetic algorithmization and natural selection of objects, the logging of all stages of screening out strong populations.

The scope developed information system can be used to solve a wide range of tasks, in particular, for creating raster graphics, opt**imizing file storages.**

The value of the work and conclusions Such applications allow solving problems with pixelization of various kinds of graphic objects, with modification, scalability, leaving raster images in different fields of activity, which is confirmed by the developed software product in this master's work.

Projections on development research Develop universal software algorithms, optimization, which can be used to switch to vector graphics of information systems from different spheres of the activity. Develop a software tool and user interface for graphical representation of results, comparative analysis of input and output data.

In section "Economics Calculations of the complexity of software development, software development costs and the duration of its development were carried out.

List of keywords: GENETIC PROGRAMMING, DIRECT CODING, INDIRECT CODING, CROSSINGS, MUTATION, NATURAL SELECTION, FUNCTION OF ADMISSIBILITY, GENETIC OPERATIONS, GENOTYPE, GENOME, FITNESS-METHOD, SOLUTIONS, PROGONS, GENERATION OF GENERATIONS

Зміст

Вступ

Векторизацією називають процес отримання векторної моделі на основі растрового зображення. Суть проблеми полягає в тому, що в даний час не існує методу, який дозволяє повністю автоматизувати переклад в векторну форму інформації, представленої в графічному вигляді. Багато в чому це пов'язано з тим, що алгоритмічно не вирішена задача однозначного трактування графічних зображень. Під векторної формою далі будемо розуміти набір об'єктів, які задаються точками, ламаними або багатокутниками.

Однак є безліч областей, в яких існує необхідність у подібних перетвореннях. В першу чергу це геоінформаційні системи, де для створення закінчених продуктів необхідне перетворення традиційних джерел картографічної інформації - паперових носіїв - в електронну форму. Серед інших сфер застосування алгоритмів векторизації можна назвати САПР, дизайн і підготовку друкованих / електронних видань. У кожній з цих областей існують свої особливості і складності. Наприклад, в ГІС зазвичай мається на увазі, що вихідні растрові зображення карт використовують обмежену і досить невелике число кольорів, але при цьому необхідно обробляти дуже великі за розмірами зображення. Також в ГІС і системах проектування часто вихідні дані містять суміш лінійних (суцільних, пунктирних, штрихпунктирною і ін), майданних, символьних об'єктів, умовних знаків, заштрихованих областей і т.д. Що також сильно збільшує складність завдань розпізнавання і векторизації. Що стосується САПР-додатків, то там існує потреба виділення таких специфічних випадків, як прямі кути між лініями і дуги кіл.

Завдання, які існують в області дизайну та поліграфії, найменше формалізовані і визначені. У більшості випадків, в подібних завданнях растрові зображення мають менші розміри, ніж в ГІС і САПР, зазвичай мається на увазі, що символьно-текстова частина зображення не потребує розпізнаванні, тому що

легко може бути додана вручну (так само необхідно відзначити той факт, що для розпізнавання текстів розроблений цілий клас алгоритмів, досить сильно відрізняються від алгоритмів векторизації). З іншого боку, в задачах цього типу вихідні растрові зображення мають досить велику кількість кольорів, зазвичай перевищує межу, при якому відомі алгоритми дають хороші результати, тому додатково виникає проблема якісних алгоритмів попередньої обробки растра - зменшення числа кольорів і фільтрації від дрібних перешкод.

Метою даної роботи є реалізація алгоритмів векторизації для використання в сфері дизайну та поліграфії. Алгоритми реалізовані у вигляді додатку на PC.

У першому розділі проводитися огляд представлених в літературі методів векторизації, їх особливості, переваги та недоліки. У другому розділі детально описується пристрій алгоритму векторної скелетизації використаного при написанні програми, причини вибору генетичних алгоритмів їхні переваги перед відомими алгоритмами які використовують такі програми як Adobe Photoshop / Illustrator / Corel Draw, наведені описи і результати роботи онлайн перетворювачів. У третьому розділі розміщено короткий опис програмної реалізації написання програми.

Актуальність роботи: В даний час існує досить велика кількість методів класифікації, прогнозування, розпізнавання образів є важливою частиною інтелектуально-інформаційного аналізу в рамках широкого кола прикладних задач. Використання накопиченої інформації про різні об'єкти та явища передбачає визначення залежності між початковими і фінальними інформаціями, що і формулюється у вигляді задач класифікації та прогнозування.

Початкова інформація може бути неточною та неповною, предметна область недостатньо формалізованою, а вид залежності занадто складним і неявним для побудови адекватної моделі потрібної залежності. У таких випадках використовується т. Зв. навчання по прецедентах. В рамках даного підходу аналізується безліч пар виду (<початкова інформація »,« фінальна інформація>) і будується алгоритм, коректно відтворює задану відповідність на наявному кінцевому безлічі об'єктів.

Безліч прецедентів називають навчальною вибіркою, а початкові інформації, що описують окремі прецеденти, - об'єктами навчання. Так як інформація про предметну область відсутня, вибір залежно проводиться з деякого евристичного сімейства алгоритмів. Таке сімейство є в деякому сенсі універсальним, оскільки воно повинно бути застосовано для широкого класу задач.

Цілі і завдання дослідження. Метою даної магістерської роботи є створення і впровадження генетичних алгоритмів в сферу роботи програми для векторизації растрової графіки, так само опис роботи методик природного відбору геномів, представлених в оточенні мутації на різних етапах життєвого циклу популяції об'єктів програми.

Для досягнення поставленої мети потрібно випадковим чином створити початкову популяцію; навіть якщо вона виявиться абсолютно неконкурентоспроможною, ймовірно, що генетичний алгоритм все одно достатньо швидко переведе її в життєздатну популяцію. Таким чином, на першому етапі можна особливо не стараються зробити надто вже пристосованих особин, досить, щоб вони відповідали формату особин популяції, і на них можна було застосувати функцію пристосованості (Fitness). Підсумком першого кроку є популяція H, що складається з N особин.

Об'єкт дослідження функція пристосованості - функція оцінки, яка визначає міру пристосованості отриманого рішення, дозволяє оцінити ступінь пристосованості конкретних особин в популяції і вибрати з них найбільш пристосовані (тобто мають мінімальне значення функції пристосованості) відповідно до еволюційного принципом виживання "найсильніших" (найкраще пристосувалися).

Предмет дослідження – об'єкти робочого простору, полігони, точки, їх колірні властивості розташування, етапи мутації і схрещування протягом життєвого циклу програми.

Ідея роботи створення максимально подібної колірної гами за наявним зразком, запис етапів мутації, експорт отриманого результату в SVG.

Методи дослідження. При вирішенні поставленого завдання використовувалися наукові досягнення в областях генетичних алгоритмів при цьому до алгоритму ставляться такі вимоги: по-перше, він не повинен допускати помилок на прецедентах, і по-друге - задовольняти деяким додатковим обмеженням, що відображає властивості і особливості предметної області [13, 14]. В рамках алгебраїчного підходу було доведено можливість побудови коректних алгоритмів для широкого класу регулярних завдань

Наукова новизна отриманих результатів полягає у створеній методиці створення популяцій геометричних фігур, з власним геофоном, геноми якого формуються під час еволюції інших об'єктів.

Практичне значення отриманих результатів полягає в розробці програмного засобу, який дозволяє векторизувати растрову графіку і записувати кожну ітерацію природного відбору об'єктів дослідження.

Результати дипломної роботи можуть бути використані підприємствами, фірмами, розробниками для проектування інформаційних систем, створення програмного продукту

Особливий внесок магістра складається в:

- виборі методів досліджень і технологій реалізації;
- створення інформаційної системи, що реалізує механізми генно-орієнтованого підходу;
- розробці теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи розробки інформаційних систем;
- оцінці отриманих результатів.

Структура і обсяг роботи. Робота складається з вступу, чотирьох розділів і висновку. Містить 93 сторінки друкованого тексту, з 16 малюнками, 92 використаних джерел.

РОЗДІЛ 1

ОГЛЯД ЗАСОБІВ ДЛЯ ВЕКТОРИЗАЦІЇ ЗОБРАЖЕНЬ

1.1 Причини, передумови появи підходів

В даний час розроблено велика кількість різноманітних алгоритмів перекладу графічних зображень у векторну форму. Одним з класичних методів векторизації є розпізнавання на основі еталонних зображень. Однак цей підхід, який виправдовує себе в задачах розпізнавання друкованих і рукописних текстів, малоприйнятний до інших областей, зокрема до використання векторизації в ілюстративної графіку, через неможливість створення еталонів стосовно до даної задачі.

Для даної області більше підходять алгоритми, в яких робляться спроби виділити на растрі деякі конструкції загального вигляду. Такими конструкціями на найнижчому рівні зазвичай бувають ламані лінії. На наступних етапах можливе подальше уточнення результатів, їх подальше розпізнавання, наприклад, виділення прямих кутів, правильних багатокутників, дуг кіл та ін. (т.зв. об'єктна векторизация, см. [1]), або, стосовно завдань ілюстративної графіки, наприклад, апроксимація знайдених ламаних гладкими кривими.

В алгоритмах, що використовують даний підхід, завдання зазвичай «вирішується за наступною схемою: спочатку виділяються осьові лінії об'єктів у вигляді ланцюжків пікселів растра. Потім виконується лінійна апроксимація знайдених осьових ліній, в результаті чого формується модель зображення у вигляді наборів ламаних, стикуються в своїх кінцевих точках. Нарешті, виконується постобработка результату апроксимації, з метою підвищення якості. »[2]

Основні відмінності алгоритмів зазвичай укладені в першому етапі виділення осьових ліній. В [2] автор виділяє наступні сім груп алгоритмів: (1) засновані на утоньшній ліній, (2) засновані на зіставленні контурів, (3) засновані на графах об'єктних штрихів, (4) засновані на розбитті зображення регулярною сіткою, (5) засновані на розрідженому перегляді растра, (6) засновані на перетворенні Хафа, (7) засновані на апроксимації об'єктів растра

площадковими геометричними фігурами. Розглянемо докладніше деякі з цих підходів:

1. Методи, засновані на утоньшення ліній. Ці алгоритми також називають алгоритмами скелетизації або алгоритмами приведення до центральної осі [3]. В результаті їх роботи виходить сукупність ліній одиничної довжини, розташованих уздовж центральної осі вихідного об'єкта на растрі. Одним з визначень скелета є наступне (по Пфалцу і Розенфельду) - безліч всіх точок - центрів кіл, поміщених усередину об'єкта і мають максимально можливі радіуси.

Існує кілька груп алгоритмів, що реалізують знаходження скелета.

Алгоритми, засновані на стирання граничних шарів, вирішують задачу або шляхом ітеративного стирання пікселів на кордоні об'єкта до отримання лінії одинарної товщини, або шляхом знаходження меж об'єкта, апроксимації їх ламаними, які потім розглядаються як фронт поширення «всередину об'єкта» плоскої хвилі. Як результат роботи в цьому випадку розглядаються точки, в яких починають і закінчують перетин різні фронти поширення. Однак алгоритми даної групи, не дивлячись на достатню опрацьованість, мають досить високою трудомісткістю і зайвої точністю при моделюванні осьових ліній, якщо на растровому об'єкті є паразитні відгалуження, і дають неточності при моделюванні перетинів та відгалужень ліній на растрі (Рис 1).

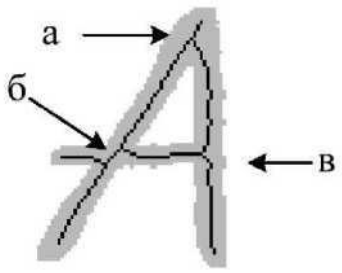

Рис. 1 – *а* - паразитні відгалуження скелета, *б* і *в* -дефекти при моделюванні розгалуження і перетину.

Для кардинального вирішення проблеми трудомісткості скелетизації різними авторами було запропоновано ряд неітеративному алгоритмів

побудови скелета, заснованих на визначенні скелета Пфалца і Розенфельда, які визначають його як безліч центрів максимальних кіл, поміщених усередину об'єкта. Для виявлення даних точок на растрі спочатку необхідно побудувати за бінарним растру растр такого ж розміру, пікселі якого, що знаходяться в точках об'єктних пікселів вихідного растру, мають значення, рівні відстані від них до найближчої фонової точки на вихідному растрі. Дане відстань, як правило, обчислюється з використанням деякої целочисленной метрики, наприклад, відстані від Манхеттену». Таке перетворення отримало назву перетворення відстані або діскантние перетворення. Потім потрібно переглянути отриманий растр і, точки, в яких спостерігається максимальне значення в локальній околиці, є точками, найближчими до скелетних (які можуть розташовуватися між пікселями, якщо товщина лінії - парна). Ці алгоритми мають трудомісткість строго пропорційну величині растра. Однак їм притаманні такі недоліки: не гарантовано зв'язність скелета, якщо вихідний об'єкт на растрі був зв'язковим, можливі скелетні лінії товщиною в два пікселя, що призводить до ускладнення процедур трасування скелетних ліній, дані алгоритми схильні до помилок в позиціонуванні скелетних точок в місцях стикування або перетину ліній растра. Ці алгоритми мають трудомісткість строго пропорційну величині растра. Однак їм притаманні такі недоліки: не гарантовано зв'язність скелета, якщо вихідний об'єкт на растрі був зв'язковим, можливі скелетні лінії товщиною в два пікселя, що призводить до ускладнення процедур трасування скелетних ліній, дані алгоритми схильні до помилок в позиціонуванні скелетних точок в місцях стикування або перетину ліній растра. Ці алгоритми мають трудомісткість строго пропорційну величині растра. Однак їм притаманні такі недоліки: не гарантовано зв'язність скелета, якщо вихідний об'єкт на растрі був зв'язковим, можливі скелетні лінії товщиною в два пікселя, що призводить до ускладнення процедур трасування скелетних ліній, дані алгоритми схильні до помилок в позиціонуванні скелетних точок в місцях стикування або перетину ліній растра.

Для подолання цих недоліків в літературі були запропоновані різні

алгоритми, що поліпшують якість скелета по Пфалцу-Розенфельду. Але слід зазначити, що подібні алгоритми корекції збільшують загальну трудомісткість, хоча саме для її зменшення в основному і був запропонований алгоритм Пфалца-Розенфельда.

2. Методи, засновані на зіставленні контурів. При роботі алгоритми даного класу припускають, що зображення містить в основному прямі лінії. Проводиться апроксимація відрізками кордонів між об'єктами. Потім для кожного відрізка робиться спроба знайти «другу сторону лінії» - інший майже паралельний йому відрізок, що обмежує область того ж кольору, що знаходиться на відстані менше заданого порогу (Масимально ширина лінії). Після знаходження двох сторін лінії, побудова осьової лінії є тривіальним завданням. Але алгоритми даного класу насилу справляються з випадками відгалуження ліній під малим кутом, або з випадками, коли одному граничному вектору можна зіставити кілька протилежних векторів, або коли зображення містить вигнуті лінії. Також недоліком даних алгоритмів є їх надмірна витонченість при обробці деяких окремих випадків.

3. Методи, засновані на графах об'єктних штрихів растру. Основна ідея даної методики - на основі послідовного перегляду растра і аналізу послідовностей об'єктних пікселів в рядках і стовпцях растра побудувати компактне топологічного векторне подання растрового зображення. Потім, переглядаючи це уявлення, можна досить ефективно побудувати детальну векторну модель растра.

Для цього вводиться поняття штриха як послідовності об'єктних пікселів, розташованих один за одним в одному стовпці або в одному рядку, і обмеженою з обох сторін фоновими пікселями. Ортогональної координатою штриха називається номер стовпчика, в якому розташований вертикальний штрих, або номер рядка в разі горизонтального штриха. Шляхом двократного перегляду растра знаходяться всі об'єктні штрихи.

Після цього проводиться пошук об'єктних ребер, як послідовностей суміжних штрихів. Знайдені об'єктні ребра приводяться до «структурному» виду - до ліній одинарної товщини.

Гідність даної методики - ефективність за часом. До недоліків даного методу відносяться неточна обробка точок розгалуження і неточна обробка кривих ліній.

4. Методи, засновані на розбитті зображення регулярною сіткою. Головною перевагою алгоритмів даного типу є їх висока тимчасова ефективність. Ідея, що лежить в їх основі така: все зображення, покривається регулярною сіткою і аналізуються лише пікселі растра на перетині з лініями сітки. Вміст комірки сітки відновлюється по конфігурації цих перетинів з використанням визначеного безлічі шаблонів (Рис. 2).

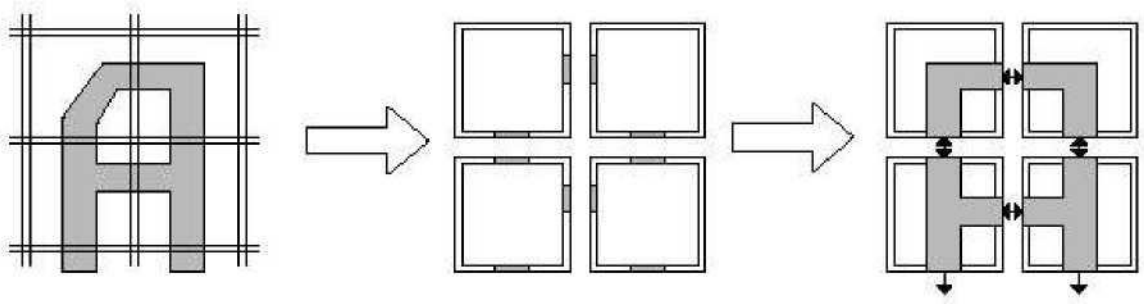

Рис. 2 – Початкове зображення, розбиття його на квадрати і заміна квадратів шаблонами.

При правильному виборі параметрів алгоритму більшість осередків буде визначено, що залишилися ж можуть бути оброблені або алгоритмом скелетизації, або подальшим рекурсивним діленням осередки. До недоліків даного алгоритму відносять складності при виборі оптимального розміру осередку, які можуть привести до некоректної обробки точок стикування і розгалуження, або до пропусків ліній, особливо штрихових і пунктирних. Також дані алгоритми не пристосовані для векторизації зображень з великим числом кривих.

5. Методи, засновані на розрідженому перегляді растра. Основною метою запропонованих алгоритмів є зменшення трудомісткості. Досягається це за рахунок відмови від перегляду всього растра, як це робиться в алгоритмах скелетизації, проводиться лише обробка об'єктних пікселів, при цьому по можливості не всіх, а лише необхідних для побудови коректної векторної моделі.

MIC-алгоритм (алгоритм «максимальних вписаних кіл», Maximal Inscribing Circles) орієнтований на пошук прямих ліній на растрі. В ідеалі пряма лінія представляється як якийсь прямокутник, що має цілком певні властивості, на яких і ґрунтується метод. При перегляді лише об'єктних пікселів алгоритм шукає пікселі, які є центрами вписаних кіл, таких що кордони окружності стосуються трьох сторін прямокутника (див. Рис. 3). При цьому мається на увазі, що растр є лінійчатим, ті. не містить зафарбованих майданних об'єктів, отже, кожен прямокутник представляє собою окремий сегмент лінії.

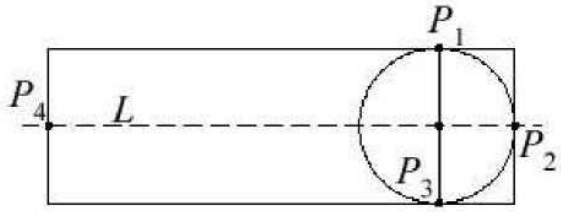

Рис. 3 – MIC – алгоритм.

Після знаходження такого центру, знайшовши дві діаметрально протилежні точки дотику кола з фоном (P1 і P3), визначаємо товщину лінії як довжину відрізка, що з'єднує ці точки. Потім проводиться пошук початку і кінця сегмента (P2 і P4) як максимально віддалених точок від знайденого центру вписаного кола в напрямку осі L знайденого сегмента.

Перевагою даного алгоритму є досить висока швидкість роботи на розріджених лінійчатих растрах, в основному складаються з відрізків прямих ліній, але при щільно заповнених растрах, особливо що складаються з дрібних

відрізків або плавних кривих, відзначається різке зниження швидкості роботи.

Інший алгоритм даного сімейства - OZZ-алгоритм відновлює середню лінію об'єкта слідуючи траєкторії пучка світла, що відбивається від внутрішніх кордонів об'єкта і переміщається уздовж об'єкта. Середня лінія відновлюється як ламана з вузлами на серединах траєкторій пучка між відображеннями.

6. Методи, засновані на площадковому моделюванні об'єктів.

Усевищевикладені алгоритми некоректно працюють на нелінійчатих растрах. Частковим вирішенням проблеми є сегментація зображення на тонкі лінії і майданні об'єкти з подальшою «зшивкою» результатів векторизації, але при цьому не завжди можливо поєднання отриманих векторних моделей через похибки, що виникають в процесі оцифровки.

Для вирішення даної проблеми було запропоновано алгоритми, засновані на наступній ідеї. Всі об'єкти, зображувані на вихідному растрі, аппроксимируются наборами багатокутників, мають просту

геометричну структуру, наприклад, квадратами, трапеціями і т.п. Потім, в залежності від характеристик побудованих багатокутників, частина з них належить апроксимуючої майданні об'єкти, а частина, що залишилася - лінійні. Так як багатокутники мають просту структуру, то досить просто побудувати векторні моделі кордонів майданних об'єктів і осьових ліній лінійних об'єктів. Через те, що для моделювання об'єктів використовується одне безліч багатокутників, об'єднання даних векторних моделей не становить труднощів.

Всі алгоритми цього типу мають трудомісткість більшу, ніж лінійна, щодо площі растра.

«Порівляльне вивчення алгоритмів, що реалізують вищеописані методики, дозволило зробити наступні висновки. Методи, засновані на площадковому моделюванні об'єктів, на відміну від методів зіставлення контурів, розбиття зображення регулярною сіткою, розрідженого перегляду растра, а також методів, заснованих на графах об'єктних штрихів і методів,

заснованих на перетворенні Хафа, так як дозволяють без спеціальних хитрувань отримувати осьові лінії не тільки прямих ліній, а й довільних лінійчатих растрів. »[2]

В моделюванні об'єктів притаманні такі негативні риси, як досить велика трудомісткість і орієнтованість на бінарні (двокольорові) растри. Теоретично можливо приведення повнокольорового растра до набору з одного або більше бінарних зображень таким чином, що векторизация отриманого набору дає вірну загальну векторну модель оригіналу. Але реалізація цього підходу, по-перше, вельми і вельми нетривіальна і, звичайно, збільшує сумарну трудомісткість; по-друге, призводить до проблем поєднання векторних моделей окремих бінарних шарів.

Іншим обмежувальним чинником є те, що моделювання, як і більшість вищеописаних алгоритмів, не дозволяє отримати правильні результати при наявності на растрі, так як в цьому випадку векторних поданням майданних об'єктів будуть якісь лінії, отримані «утоньшение» їх до послідовностей пікселів одиничної товщини. Природно для користувача результати такої векторизації марні (див. Рис 4).

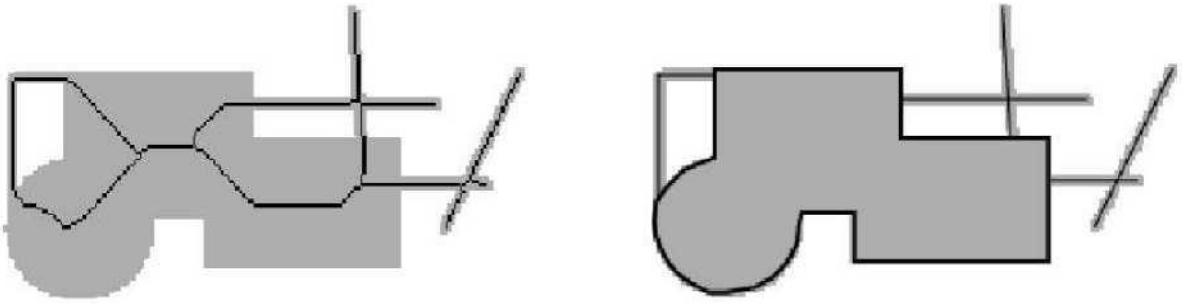

Рис. 4 – Зліва - результат застосування скелетизації до растру, який містить майданні об'єкти. Праворуч – вірна з точки зору користувача векторна модель растра.

Єдиним виходом з даної ситуації є попередня сегментація растра на частини, що містять об'єкти одного типу. При цьому можливе застосування спеціалізованих алгоритмів, наприклад, сегментація тексту - графіки, лінійних

- майданних об'єктів, тонких - товстими та ін. [1] Для нашої предметної області актуальна лише сегментація лінійних і майданних об'єктів. Для цього в літературі пропонується підхід, заснований на застосуванні операторів математичної морфології, суть якого полягає в наступному: задається граничне значення товщини, рівне n. Далі до зображення застосовується перетворення, стирає до n шарів граничних пікселів. Отриманий растр містить лише майданні об'єкти, його лише треба відновити за допомогою передбачених процедурою методів.

«Недоліком такого алгоритму є те, що йому необхідно виконати n + 1 ітерацію завершення роботи, таким чином, його трудомісткість пропорційна n • S, де S - площа вихідного зображення» [2]. Крім цього передбачається, що вихідний растр є бінарним, що також обмежує застосування цього алгоритму.

Інший негативною стороною сегментації є те, що в результаті виходять по суті дві векторні моделі, які потребують процесі суміщення, в результаті чого може постраждати точність векторизації.

Для подолання цих недоліків в роботах [2] і [4] був запропонований алгоритм векторної генетичної векторизації, який дозволяє одночасно класифікувати об'єкти на лінійні і майданні і отримувати єдину векторну модель, яка потребує додаткових зусиль по «зшивці» об'єктів. Ще одним його перевагою є можливість застосування при векторизації багатобарвних растрів. Даний алгоритм заснований на використанні тріангуляції в якості базової структури представлення вмісту растра. Він і був використаний, з невеликими змінами, для реалізації функцій векторизації, з породженням об'єктів природного відбору. Детальний його опис наведено в наступних розділах 1.2, 1.3.

1.2 Природна еволюція

Як відомо, у Природи є свій метод створення кращих організмів. Дарвін назвав його Еволюцією внаслідок природного відбору. Еволюція має на увазі під собою послідовний розвиток організмів - безперервну послідовність

батьків і їх дітей, коли діти багато успадковують від своїх батьків, але де в чому від них відрізняються.

Природний відбір - це безперервне бій за життя між усіма. "Виживає сильніший" - ось життєве кредо Природи, якщо нагороджувати титулом "сильний" самого підходящого, самого пристосованого для життя. Якщо підходити до опису еволюції більш формально, то спочатку необхідно зазначити, що об'єктом розвитку (тобто еволюції) є не самі організми, а види в цілому. Вид - це сукупність організмів, подібних за будовою та іншими ознаками. Користуючись термінологією об'єктно-орієнтованого програмування, вид - це клас, а належать виду індивіди - об'єкти цього класу. Сукупність індивідів одного виду назвемо популяцією. Щоб еволюція взагалі була можлива, організми повинні відповідати 3 найважливішим властивостям:

1. Кожен індивід у популяції здатний до розмноження. Відмінності індивідів друг від друга впливають на ймовірність їх виживання.

2. Кожен нащадок успадковує риси свого батька (подібне походить від подібного). Ресурси для підтримки життєдіяльності і розмноження обмежені, що породжує конкуренцію і боротьбу за них. Всі процеси в живих організмах працюють за рахунок складних молекул - білків.

3. Кожен білок являє собою маленький біологічний автомат. Молекула білка складається з послідовності амінокислот. Сукупність інформації і будова всіх білків в організмі визначає його початкову структуру (розвиток організму відбувається також і під дією зовнішнього середовища). Вся ця інформація називається генетичною інформацією, або генотипом. Процес побудови, розвитку організму за інформацією з генотипу називається онтогенезом. А будова, якості та властивості організму - фенотип. Оскільки зовнішнє середовище впливає на організм в цілому, то можна сказати, що ймовірність виживання організму визначається фенотипом. Генетична інформація в клітці зберігається в спеціальних молекулах - нуклеїнові кислоти. Нуклеїнова кислота являє собою полімер, тобто молекулу, яка була послідовність із

сполучених між собою невеликих молекул - мономерів. Мономерами нуклеїнових кислот є нуклеотиди.

Для кодування інформації використовується 4 види нуклеотидів, які охоплюють за назвами входять до них азотистих основ - A, T, G, C. Таким чином, алфавіт кодування складається з 4 букв. При сексуальному розмноженні нащадку передається інформація про будову батьків шляхом передачі ДНК. Генетична інформація в клітці зберігається в спеціальних молекулах - нуклеїнові кислоти. Нуклеїнова кислота являє собою полімер, тобто молекулу, яка була послідовність із сполучених між собою невеликих молекул - мономерів. Мономерами нуклеїнових кислот є нуклеотиди. Для кодування інформації використовується 4 види нуклеотидів, які охоплюють за назвами входять до них азотистих основ - A, T, G, C. Таким чином, алфавіт кодування складається з 4 букв. При сексуальному розмноженні нащадку передається інформація про будову батьків шляхом передачі ДНК. Генетична інформація в клітці зберігається в спеціальних молекулах - нуклеїнові кислоти. Нуклеїнова кислота являє собою полімер, тобто молекулу, яка була послідовність із сполучених між собою невеликих молекул - мономерів. Мономерами нуклеїнових кислот є нуклеотиди. Для кодування інформації використовується 4 види нуклеотидів, які охоплюють за назвами входять до них азотистих основ - A, T, G, C. Таким чином, алфавіт кодування складається з 4 букв. При сексуальному розмноженні нащадку передається інформація про будову батьків шляхом передачі ДНК. Для кодування інформації використовується 4 види нуклеотидів, які охоплюють за назвами входять до них азотистих основ - A, T, G, C. Таким чином, алфавіт кодування складається з 4 букв. При сексуальному розмноженні нащадку передається інформація про будову батьків шляхом передачі ДНК. Для кодування інформації використовується 4 види нуклеотидів, які охоплюють за назвами входять до них азотистих основ - A, T, G, C. Таким чином, алфавіт кодування складається з 4 букв. При сексуальному розмноженні нащадку передається інформація про

будову батьків шляхом передачі ДНК.

При цьому, для побудови ДНК нащадка, батьківські ДНК міняються своїми ділянками. Це процес називається схрещуванням (кросовер crossover). При цьому новий ген являє собою комбінацію інформації з батьківських ДНК (рекомбінація спадкової інформації).

Розрахунок вектора цільових значень.

Крок еволюції - побудова нового покоління.

Перевірка критерію завершення, якщо не виконано - перехід на 2.

Крок еволюції можна розділити на наступні етапи:

1. Обчислення вектора пристосованості.
2. Відбір кандидатів на схрещування (Відбір - Selection).
3. Схрещування, тобто породження кожною парою відібраних кандидатів нових індивідів, шляхом геномів.
4. Мутація геномів.
5. Обчислення вектора цільових значень і побудова нової популяції (нового покоління).

1.3 Цільова функція і кодування

Серед компонентів, що утворюють генетичний алгоритм, в більшості випадків тільки два безпосередньо визначаються конкретним завданням - це кодування завдання (відображення простору пошуку на простір бітових рядків) і цільова функція. Розглянемо задачу параметричної оптимізації, яка полягає у визначенні набору змінних, мініімізірующіх деяку величину (мета). У більш традиційних термінах, завдання полягає в пошуку мінімуму деякої функції F (X1, X2, ..., XM).

На першому етапі зазвичай робиться припущення, що змінні, що представляють параметри, можуть бути представлені бітовими рядками. Це означає, що змінні попередньо дискретизируются деяким чином і що область дискретних значень є певною мірою 2. Наприклад, з 10 бітами на параметр ми

отримуємо область з 210 = 1024 дискретних значень. Якщо параметри неперервні, то проблема дискретизації не заслуговує на особливу увагу. Зрозуміло, передбачається, що дискретизація забезпечує достатнє розширення, щоб зробити можливим регулювання отримання результату з бажаним рівнем точності. Передбачається також, що дискретизація в деякому сенсі представляє основну функцію.

Бітову рядок довжини N можна розглядати як ціле двійкове число I, якому відповідав би якийсь речовий значення r з заданого діапазону [Min, Max]. Це відповідність встановлюється формулою

$$r = \text{Min} + (\text{Max} - \text{Min})\, I / (2\text{-}1)_N \qquad 1.0$$

За винятком проблеми кодування, цільова функція зазвичай дається як частина постановки завдання. З іншого боку, розробка цільової функції іноді може безпосередньо входити в розробку алгоритму оптимізації або пошуку рішення. В інших випадках значення цільової функції може залежати від реалізації і давати тільки наближений або приватний результат.

1.4 Загальна структура генетичного алгоритму

У природі особини в популяції конкурують один з одним за різні ресурси, такі, наприклад, як їжа або вода. Крім того, члени популяції одного виду часто конкурують за залучення шлюбного партнера. Ті особини, які найбільш пристосовані до навколишніх умов, матимуть відносно більше шансів на відтворення нащадків. Слабо пристосовані особини або зовсім не справлять потомства, або їх потомство буде дуже нечисленним. Це означає, що гени від високо адаптованих або пристосованих особин будуть поширюватися в збільшенні кількості нащадків на кожному наступному поколінні. Таким чином, вид розвивається, все краще пристосовуючись до середовища проживання.

Генетичні алгоритми використовують пряму аналогію з таким механізмом. Вони працюють з сукупністю "особин" - популяцією, кожна з яких представляє можливе рішення даної проблеми. Кожна особина

оцінюється мірою її "пристосованості" згідно з тим, наскільки "добре" відповідне їй рішення задачі. Наприклад, мірою пристосованості могло б бути ставлення сили / ваги для даного проекту моста. (В природі це еквівалентно оцінці того, наскільки ефективний організм при конкуренції за ресурси.) Найбільш пристосовані особини отримують можливість "відтворювати" потомство за допомогою "перехресного схрещування" з іншими особинами популяції.

Це призводить до появи нових особин, які поєднують в собі деякі характеристики, успадковані ними від батьків. Найменш пристосовані особини з меншою ймовірністю зможуть відтворити нащадків, так що ті властивості, якими вони володіли, будуть поступово зникати з популяції в процесі еволюції.

Таким чином, відтворюється вся нова популяція допустимих рішень, вибираючи кращих представників попереднього покоління, схрещуючи їх і отримуючи безліч нових особин. Це нове покоління містить більш високе співвідношення характеристик, якими володіють хороші члени попереднього покоління. Таким чином, з покоління в покоління хороші характеристики поширюються по всій популяції.

Схрещування найбільш пристосованих особин призводить до того, що досліджуються найбільш перспективні ділянки простору пошуку. В кінцевому підсумку популяція буде сходитися до оптимального рішення задачі. Саме тому виникла потреба в створенні програмного продукту використовують генетичні алгоритми для перетворення растрового зображення у векторне.

РОЗДІЛ 2

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

2.1.1 Предметна область

У всіх методах побудови алгоритмічних композицій використовується локальна оптимізація. Вона проводиться тому, що завдання одночасної

оптимізації всіх базових алгоритмів і коректує операції надзвичайно складна. Вона є багатопараметричної, багатоекстремального, і в загальному випадку не зводиться до готових процедур навчання базових алгоритмів.

При практичному вирішенні складних, погано формалізованих, многопараметрических завдань несподівано гарні рішення можуть знаходити еволюційні алгоритми. Одними з найбільших і поширених серед них є генетичні алгоритми, запропоновані Холландом в 1975 році [53]. Алгоритми цього класу засновані на дарвінівський еволюційних принципах, а саме забезпечення генетичної різноманітності і природного відбору. Так, в рамках генетичного алгоритму формується популяція потенційних рішень завдання, звані инди- видами. У просторі індивідів визначаються бінарна операція рекомбінації і унарна - мутації (можливі і інші «генетичні» операції). Далі запускається ітераційний процес, на кожному кроці якого поточна популяція піддається рекомбінації і мутації, після чого проводиться селекція кращих варіантів вирішення для наступного покоління. Генетичні алгоритми використовувалися для широкого кола завдань, в тому числі і в розпізнаванні [41]. Так, в [91] генетичний алгоритм використовується для відбору найбільш інформативних ознак, в [40] - для синтезу логічних прищепив класифікації.

Відомі також методи побудови вирішальних дерев, коли рекомбінація реалізується як обмін піддеревами у пари вирішальних дерев. У даній роботі використовується особливий вид генетичного алгоритму, званий кооперативної коеволюція (Cooperative Coevolution) [64, 63]. Дана модель використовує кілька популяцій, кожна з яких відповідає рішенню частини завдання або її подзадаче. Популяції взаємодіють один з одним за принципом симбіозу, вибираючи алгоритми, найкращим чином кооперирующих з іншими. Фактично, він використовується як метод настройки конкретної моделі алгоритмів. У даній роботі пропонується новий підхід, при якому генетичний алгоритм виступає в ролі надбудови над стандартними методами навчання базових алгоритмів, не втручаючись в їхню внутрішню структуру. відповідає одному алгоритму композиції, а корекція проводиться тільки при

оцінці пристосованості індивідів. В процесі еволюції кожен базовий алгоритм отримує можливість спеціалізуватися в певній, тільки йому притаманною ролі. В рамках даного підходу розроблено і детально описаний в даній роботі метод побудови локальних базисів на основі кооперативної коеволюції.

Даний метод, на відміну від проблемно-орієнтованих методів алгебраїчного підходу, не є жадібним і дозволяє будувати композиції мінімальній потужності. Кооперативна коеволюция вперше використовується для побудови алгоритмічних композицій з довільними базовими родинами алгоритмів і коригувальних операцій. Метод реалізований для лінійних коригувальних операцій і протестований з правилом Байеса в якості базового сімейства алгоритмів на ряді модельних і реальних завдань.

2.1.2 Постановка задачі

Мета роботи: розробити генетичний алгоритм для постоенія дерева коеволюції використовує кілька створюваних популяцій для генерації полігонів пристосованих фітнес функцією спираючись на задане зображення користувача, запис генеруються поколінь.

У процесі досягнення поставлених цілей виникають наступні завдання:

1. Швидкий і надійний спосіб знаходження сполучних точок між зображеннями.

2. Застосування фітнес методу для відстеження якості генеруються нащадків.

3. Мутація поточних полігонів, схрещування.

4. Введені обмеження:

5. Функцією пристосування буде сума похибок кольорів всіх пікселів.

2.1.3. Аналіз завдання

Генетичні алгоритми є найбільш поширеними і найбільш вивченими еволюційними алгоритмами. Їх відмінною рисою є уявлення індивідів у

вигляді бінарних векторів, певним чином кодують рішення поставленого завдання. Генетичний алгоритм зазвичай містить дві генетичні операції - схрещування (рекомбінації, кросинговеру) і мутації. Формально для конвертації в векторні зображення досить скористатися наступною набором параметрів:

$$GA = (P0, n, ni, m, S, C, M, f, t) \quad (2.0)$$

$P0 = (a1,..., An) \in I\ n$, $I = \{0, 1\}$

m - початкова популяція;

n - розмір популяції;

ni - розмір проміжної популяції, з якої виробляється селекція найкращих нащадків;

m - довжина бінарного уявлення індивіда;

$S: I\ ni \rightarrow I\ n$ - оператор селекції;

$C: I \times I \rightarrow I$ - оператор схрещування;

$M: I \rightarrow I$ - оператор мутації;

$f: I \rightarrow R$ - функція відповідності;

$t: I\ n \rightarrow \{0, 1\}$ - критерій зупинки;

2.1.6 Опис і структура алгоритму

Алгоритм виконується наступним чином: на першому етапі створюється початкова популяція P0 довічних векторів довжини m, що складається з n індивідів - вершини полігонів. Далі на кожній ітерації алгоритму з поточної популяції Pk за допомогою операторів схрещування та мутації формується проміжна (intermediate) популяція Pik розміру ni, з якої оператор селекції S відбирає n найбільш пристосованих для «розмноження» на наступній ітерації в залежності від отриманих даних по RGB каналах з вихідного зображення. Пристосованість, або адаптивність, індивіда, визначається функцією відповідності f (фітнес метод), що служить сполучною ланкою між громадянами та предметною областю. Індикатором зупинки алгоритму служить критерій t.

В силу згаданих вище причин існує безліч способів завдання кожного з перерахованих параметрів, кожний з яких не може бути, взагалі кажучи,

визнаний гірше або краще іншого.

2.2 Опис мов і засобів програмування для розробки програми

Для експериментів генетичних алгоритмів була обрана середовище розробки Java 8 по скільки вона більш сприятлива в плані очищення пам'яті після виконання селекції Java є мовою програмування і платформу обчислень, яка була вперше випущена Sun Microsystems в 1995 р Існує безліч додатків і веб-сайтів, які не працюють при відсутності встановленої Java, і з кожним днем число таких веб-сайтів і додатків збільшується. Java відрізняється швидкістю, високим рівнем захисту і надійністю. Від портативних комп'ютерів до центрів даних, від ігрових консолей до комп'ютерів, які використовуються для наукових розробок, від стільникових телефонів до мережі Інтернет - Java всюди!Java - об'єктно-орієнтована мова програмування, що розробляється компанією Sun Microsystems з 1991 року і офіційно випущений 23 травня 1995 року. Спочатку нову мову програмування називався Oak (James Gosling) і розроблявся для побутової електроніки, але згодом був перейменований в Java і став використовуватися для написання аплетів, додатків і серверного програмного забезпечення.

Програми на Java можуть бути трансльовані в байт-код, що виконується на віртуальній java-машині (JVM) - програмою, обробній байт-код і передавальної інструкції обладнанню, як інтерпретатор, але з тією відмінністю, що байт-код, на відміну від тексту, обробляється значно швидше.

Мова Java зародився як частина проекту створення передового програмного забезпечення для різних побутових приладів. Реалізація проекту була почата на мові C ++, Але незабаром виник ряд проблем, найкращим засобом боротьби з якими була зміна самого інструмента - мови програмування. Стало очевидним, що необхідний платформо-незалежний мова програмування, що дозволяє створювати програми, які не доводилося б

компілювати окремо для кожної архітектури і можна було б використовувати на різних процесорах під різними операційними системами.

Мова Java потрібен для створення інтерактивних продуктів для мережі Internet. Фактично, більшість архітектурних рішень, прийнятих при створенні Java, було продиктовано бажанням надати синтаксис, схожий з C і C ++. В Java використовуються практично ідентичні угоди для оголошення змінних, передачі параметрів, операторів і для управління потоком виконанням коду. В Java додані всі хороші риси C ++.

Три ключові елементи об'єдналися в технології мови Java

1. Java надає для широкого використання свої аплети (applets) - невеликі, надійні, динамічні, які не залежать від платформи активні мережеві додатки, що вбудовуються в сторінки Web. Аплети Java можуть налаштовуватися і поширюватися споживачам з такою ж легкістю, як будь-які документи HTML

2. Java вивільняє міць об'єктно-орієнтованої розробки додатків, поєднуючи простий і знайомий синтаксис з надійним і зручним в роботі середовищем розробки. Це дозволяє широкому колу програмістів швидко створювати нові програми і нові аплети

3. Java надає програмісту багатий набір класів об'єктів для ясного абстрагування багатьох системних функцій, використовуваних при роботі з вікнами, мережею і для введення-виведення. Ключова риса цих класів полягає в тому, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектра системних інтерфейсів

Для розробки інтерфейсу була обрана технологія JavaFX 2.0, нова версія платформи функціонально-багатих клієнтських Java-додатків, спочатку розробленої в Sun Microsystems, до кінця весни 2011 року перейде в стадію відкритого бета-тестування. Фінальна версія платформи повинна з'явитися пізніше в течениии року.

Основними нововведеннями JavaFX 2.0 стануть набір API для роботи з середовищем на Java, високопродуктивне графічне ядро, підтримка відтворення мультимедіа, можливість впроваджувати HTML-контент в

JavaFX-додатки, нові елементи користувацьких інтерфейсів і спрощена інсталяція.

Набір API пропонується замість мови JavaFX Script, раніше використовувався в платформі. Розробники, як і раніше охочі застосовувати мову скриптів, можуть скористатися проектом Visage, який забезпечує можливість створення користувацьких інтерфейсів за допомогою декларативного мови програмування.

Платформа JavaFX змагається з іншими технологіями створення функціонально-багатих клієнтів: HTML5, Microsoft Silverlight і Adobe Flash.

Нові інтерфейси дозволять:

1. застосовувати потужні функціональні можливості Java, такі як параметризрвані типи, анотації і багатопоточність;

2. спростити використання web-розробниками JavaFX в інших популярних динамічних мовах програмування, таких як JRuby, Groovy and JavaScript.

Повнофункціональні клієнтські програми на базі JavaFX 2.0 цілком розроблені на Java. Застосовуючи Java і в серверній, і в клієнтської частини додатків, розробники можуть значно знизити ризики, спростивши бізнес-рішення, вважають в Oracle. JavaFX 2.0 надає веб-компонент, який працює на базі движка веб-рендеринга Webkit і дозволяє розробникам ефективно комбінувати і поєднувати традиційні можливості Java і динамічні можливості веб-технологій.

Розробники можуть використовувати існуючі бібліотеки Java, отримувати доступ до вбудованих можливостям систем або ефективно підключатися до серверних сполучною додатків на базі Java Platform, Enterprise Edition (Java EE) з додатків JavaFX. Існуючі додатки Java Swing можуть бути легко модернізовані за допомогою нових функцій JavaFX, таких як

повнофункціональний графічний програмний інтерфейс, відтворення мультимедіа і вбудований веб-контент, йдеться в повідомленні Oracle.

З другою версією JavaFX замовникам і партнерам буде простіше створювати сучасні, виразні графічні інтерфейси користувача і візуальні представлення даних, спираючись на наявні знання і інвестиції в технології Java, - заявила Нандини Рамані (Nandini Ramani), віце-президент по розробці, Java Client Group , Oracle. - Платформа JavaFX 2.0 є новітньою розробкою, яка враховує вимоги сумісності з усіма новими характеристиками і функціональними можливостями набору Java Development Kit (JDK), в тому числі тими, які з'являться у версії JDK 8, наприклад, модульність і `private` методами>.

JavaFX 2.0 дає розробникам гнучкі можливості для створення додатків з використанням бажаних інструментів і мов програмування. Так, в JavaFX 2.0 вводиться FXML, мова з підтримкою сценаріїв, заснований на мові розмітки XML і призначений для опису користувацьких інтерфейсів. Розробники, знайомі з веб-технологіями або іншими мовами розмітки на базі XML, легко освоять FXML і його можливості для створення різноманітних додатків, таких як візуалізація даних і бізнес-додатки на базі форм.

JavaFX 2.0 також дозволяє застосовувати популярні мови сценаріїв, підтримувані віртуальної машиною Java (JVM), такі як Groovy, JRuby і Scala, і об'єднує простоту динамічних мов з потужною функціональністю платформи Java.

Крім того, при створенні додатків JavaFX розробники зможуть використовувати бажані інструменти Java-розробки, наприклад, NetBeans і Eclipse. Додатки JavaFX реалізуються або як настільні додатки, або в захищеному режимі через браузер з використанням модуля Java.

Додатково Oracle представила нові проекти і програми, пов'язані з платформою JavaFX: загальнодоступну бета-версію JavaFX 2.0 for Mac OS X; закриту програму ознайомчого доступу з JavaFX Scene Builder, інструментом візуального побудови інтерфейсу для платформи JavaFX (дозволяє

проектувати екран шляхом перетягування з палітри і розміщення на сцені компонентів для користувача інтерфейсу). Загальнодоступне бета-тестування Scene Builder заплановано на початок 2012 р

У зв'язку зі зміною внутрішньої архітектури JavaFX 2.0 корпорація Oracle оголосила в лютому 2012 року про майбутнє припинення підтримки старіших версій цієї платформи функціонально-багатих інтернет-додатків. Додатки, засновані на JavaFX 1.2 і 1.3, до кінця цього року необхідно оновити шляхом перекладу на версію 2.0, вказують в Oracle.

До JavaFX 2.0 робота з платформою відбувалася за допомогою мови JavaFX Script. У 2010 році на конференції JavaOne компанія Oracle оголосила, що відмовиться від JavaFX Script і реалізує виклики всіх функцій JavaFX 2.0 в API Java. Середовище виконання JavaFX 2.0 для Windows була випущена в жовтні минулого року, а пізніше з'явилися попередні релізи для Mac OS X і Linux.

Можливість завантажити JavaFX 1.2 і 1.3 на сайті Oracle збережеться до 20 грудня. Нагадаємо, компанія Sun Microsystems представила JavaFX в 2007 році в якості розширення Java, що дозволяє створювати багаті графікою мультиплатформенні додатки. Oracle після покупки Sun продовжила розвивати JavaFX, але ця технологія менш популярна, ніж інші платформи аналогічного призначення, - Flash і Silverlight.

Версія JavaFX 8 випущена як частина реалізації Oracle Java Development Kit (JDK) 8 і буде служити базовим набором інструментів розробки призначеного для користувача інтерфейсу для Java SE 8 Embedded, надаючи єдину узгоджену середу програмування додатків як для вбудованих, так і для настільних систем. JavaFX 8, включає:

1. Загальнодоступний API-інтерфейс Public UI Control API, який дозволить іншим розробникам, що створює керуючі елементи призначеного

для користувача інтерфейсу, забезпечувати висхідну сумісність з майбутніми версіями JavaFX;

2. Підтримку додаткових тегів HTML5, реалізовану в компоненті WebView, що підвищує рівень сумісності між Java і HTML5;

3. Розширену підтримку 3D для кращої візуалізації даних і інноваційних користувальницьких інтерфейсів;

4. Підтримку мов з двонаправленим письмом (в яких слова зазвичай пишуть справа наліво, а цифри - зліва направо) і складних наборів символів, що забезпечить повну інтернаціоналізацію платформи.

5. JavaFX SceneBuilder 2.0 буде також підтримувати функції JavaFX 8 і, крім того, полегшить взаємодію з NetBeans Integrated Development Environment (IDE) і іншими інтегрованими середовищами розробки Java;

6. Oracle має намір стандартизувати відповідні частини JavaFX в рамках процесу JCP в період розробки Java SE 9.

Складальник проекту Gradle - система автоматичного складання, побудована на принципах Apache Ant і Apache Maven, але надає DSL на мові Groovy замість традиційної XML-подібної форми подання конфігурації проекту.

На відміну від Apache Maven, заснованого на концепції життєвого циклу проекту, і Apache Ant, в якому порядок виконання завдань (targets) визначається відносинами залежності (depends-on), Gradle використовує спрямований ациклічний граф для визначення порядку виконання завдань.

Gradle був розроблений для розширюваних многопроектной збірок, і підтримує інкрементальні збірки, визначаючи, які компоненти дерева збірки не змінилися і які завдання, залежні від цих частин, не вимагають перезапуску.

Основні плагіни призначені для розробки і розгортання Java, Groovy і Scala додатків, але готуються плагіни і для інших мов програмування.

Для досягнення кращий цілей в тестуванні і налагодження програми обрано IDE IntelliJ IDEA, завдяки технології evolution expression - дозволяє вести спостереження і мутувати необхідні дані ефектів у програмному забезпеченні,

з використанням задаються умов дебаг режиму. IDEA - інтегроване середовище розробки програмного забезпечення на багатьох мовах програмування, зокрема Java, JavaScript, Python, розроблена компанією JetBrains.

Перша версія з'явилася в січні 2001 року і швидко стала популярною, як перша середа для Java з широким набором інтегрованих інструментів для рефакторінга [2], які дозволяли програмістам швидко реорганізовувати вихідні тексти програм. Дизайн середовища орієнтований на продуктивність роботи програмістів, дозволяючи сконцентруватися на функціональних завданнях, в той час як IntelliJ IDEA бере на себе виконання рутинних операцій.

Починаючи з шостої версії продукту IntelliJ IDEA надає інтегрований інструментарій для розробки графічного інтерфейсу користувача. Серед інших можливостей, середа добре сумісна з багатьма популярними вільними інструментами розробників, такими як CVS, Subversion, Apache Ant, Maven і JUnit. У лютому 2007 року розробники IntelliJ анонсували ранню версію плагіна для підтримки програмування на мові Ruby [3].

Починаючи з версії 9.0, середа доступна в двох редакціях: Community Edition і Ultimate Edition. Community Edition є повністю вільною версією, доступною під ліцензією Apache 2.0, в ній реалізована повна підтримка Java SE, Groovy, Scala, а також інтеграція з найбільш популярними системами управління версіями. В редакції Ultimate Edition, доступною під комерційною ліцензією, реалізована підтримка Java EE, UML-діаграм, підрахунок покриття коду, а також підтримка інших систем управління версіями, мов та фреймворків [4].

2.4 Обгрунтування вибору технічних і програмних засобів

Оскільки структура додатки з використанням обраних технологій, представляє собою windows-додаток, написаний на мові Java 8, із застосуванням фреймворка JavaFX. Здійснено робота на ОС Windows 10, так

само тестувалася робота додатки на Linux Mint.

Технічні мінімальні вимоги такі:

- Процесор з тактовою частоті 2000 mhz (наприклад, Intel Celeron або краще),
- 1024 MB оперативної пам'яті,
- VGA-сумісна графічна карта з роздільною здатністю 1024 × 768,
- 5 GB вільного дискового простору.

Однак, для більш-менш комфортної роботи потрібен комп'ютер з характеристиками, кращими ніж:

- CPU 3 ghz,
- 4096 MB оперативної пам'яті,
- 8 GB вільного дискового простору.

РОЗДІЛ 3

ПРОЕКТУВАННЯ РОБОТИ ПРОГРАМИ

Людина може порівняти зображення і виділяти на них об'єкти візуально, на інтуїтивному рівні. Однак, для машини зображення - всього лише ні про що не говорить набір даних. У загальних рисах можна описати два підходи до того, щоб якось «наділити» машину цим людським умінням. Є певні методи для порівняння зображень, засновані на зіставленні знань про зображення в цілому. У загальному випадку це виглядає наступним чином: для кожної точки зображення обчислюється значення певної функції, на підставі цих значень можна приписати зображенню певну характеристику, тоді задача порівняння зображень зводиться до задачі порівняння таких характеристик. За великим рахунком, ці методи настільки ж погані, наскільки прості, і працюють прийнятно, практично, тільки в ідеальних ситуаціях. Причин цього більш ніж достатньо: поява нових об'єктів на зображенні, перекриття одних об'єктів іншими, шуми, зміни масштабу, положення об'єкта на зображенні, положення камери в тривимірному просторі, висвітлення, аффінниє перетворення і т.д. Власне, погані якості цих методів обумовлені їх основною ідеєю, тобто

Остання фраза відразу наштовхує на думки обходу таких проблем: треба або якось вибирати точки, що вносять вклад в характеристику, або, ще краще, виділяти деякі особливі (ключові) точки і порівнювати їх. На цьому ми і підійшли до ідеї зіставлення зображень по ключових точках. Можна сказати, що ми замінюємо зображення деякої моделлю - набором його ключових точок. Відразу відзначимо, що особливої буде називатися така точка зображеного об'єкта, яка з великою часткою ймовірності буде знайдена на іншому зображенні цього ж об'єкта. Фітнес-метод служить для зіставлення полігонів, що генеруються програмної і заданої користувачем вихідної картинки. Він повинен забезпечувати інваріантність знаходження одних і тих же особливих точок щодо перетворень зображень.

Єдино, що залишається незрозумілим - яким чином визначати яка

ключова точка одного зображення відповідає ключовій точці іншого зображення. Адже після застосування фітнес методу можна визначити тільки координати особливих точок, а не рівень перекриття по каналу A (прозорості) вони на кожному зображенні різні. Тут в справу і вступають мутації. Мутатор - метод дозволяє змінити поведінки вже існуючих полігонів, колір, прозорість, а так само відповідні його гени

У підсумку виходить наступна схема рішення задачі зіставлення зображень:

1. На зображеннях генеруються за заданими критеріями точки, які утворюють граф.

2. За збігом результатів фітнес методу програмою приймається рішення необхідно залишити поточний полігон, чи придатний даний нащадок.

3. На основі набору сильних поколінь будується модель перетворення зображень, за допомогою якого з одного зображення можна отримати інше.

На кожному етапі є свої проблеми і різні методи їх вирішення, що вносить певний свавілля в рішення початкової задачі. Далі будемо розглядати першу частину рішення, а саме генерацію особливих точок і їх реакцію фітнес-методом В основному буде описаний алгоритм методу генерації потомства, а не те, чому цей алгоритм працює, і виглядає саме так.

На останок перерахуємо деякі перетворення, щодо яких ми б хотіли отримати інваріантність:

1) зміщення

2) поворот

3) масштаб (один і той же об'єкт може бути різних розмірів на різних зображеннях)

4) зміни яскравості

5) зміни положення камери

Инвариантность щодо трьох останніх перетворень в повній мірі

отримати не вдасться, але хоча б частково зробити це вже було б не погано.

3.1 Апроксимація геометричних перетворень зображень за допомогою многочленів

Типовою завданням в області цифрової обробки зображень є геометричне перетворення зображень [5], наприклад, обертання або масштабування. Геометричні перетворення також застосовуються при виправленні дисторсии (геометричних спотворень) камери [6, p. 396], побудові панорамних зображень [7] та ін. Геометричні перетворення визначають залежність між точками на вихідному і перетвореному зображенні і найчастіше визначаються у вигляді функцій прямого або зворотного відображення:

$$u = M(u')$$
$$u' = M\text{-}1(u), \qquad 3.0$$

де u '= [u', v '] - координати вихідного зображення, а u = [u, v] - координати перетвореного зображення.

Використання функції зворотного відображення (3.0) краще, так як дозволяє уникати накладень і дірок, які можуть виникнути при застосуванні функції прямого відображення, а також полегшує завдання інтерполяції

Розглянемо задачу виправлення дисторсії камери, як найбільш часто зустрічається на практиці при використанні ширококутної оптики. Модель спотворень, що вносяться об'єктивом камери, визначається характеристиками об'єктива. Наприклад, модель спотворень може бути визначена трьома коефіцієнтами радіальних спотворень k1, k2, k3 і двома коефіцієнтами тангенціальних спотворень p1 і p2 наступним чином:

$$\tilde{u} = (u - u_0) / f_u$$
$$\tilde{v} = (v - v_0) / f_v$$
$$r^2 = \tilde{u}^2 + \tilde{v}^2$$
$$\tilde{u}' = \tilde{u}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2 p_1 \tilde{u}\tilde{v} + p_2(r^2 + 2\tilde{u})$$
$$\tilde{v}' = \tilde{v}(1 + k_1 r^2 + k_2 r^2 + k_3 r^6) + 2 p_2 \tilde{u}\tilde{v} + p_1(r^2 + 2\tilde{v})$$

$$u' = f'_u \tilde{u}' + u'_0$$
$$v' = f'_v \tilde{v}' + v'_0 \qquad 3.1$$

де [u0, v0] і [fu, fv] координати центру зображення (точка перетину оптичної осі і площини) і фокусні відстані по обох осях для вихідного зображення; параметри $[u'_0, v'_0]$ І $[f'_u, f'_v]$. Визначені аналогічно для вихідного зображення. Обчислення виразу для однієї точки вимагає виконання 14 умножений за умови зберігання результатів проміжних обчислень, що є вельми витратним при обробці відео в режимі реального часу. Поширена рішення в цьому випадку: попередньо розрахувати значення функції зворотного відображення для всіх можливих точок перетвореного зображення. Для вихідного зображення розміру w × h будуть потрібні дві таблиці U (v, u) і V (v, u), де u = 0, 1, ..., w - 1; v = 0, 1, ..., h - 1. Такий підхід вимагає постійного зберігання в пам'яті таблиці заздалегідь розрахованих значень функції і, отже, часто переглядаôте цій галузі пам'яті, що може представляти труднощі в разі реалізації алгоритму виправлення геометричних спотворень на програмованих логічних інтегральних схемах (ПЛІС).

$$f(X) = a0 + a1\ x + a1\ x2 + ... + an_{-1}\ xn^{-1} + An\ xn. \qquad 3.2$$

Обчислення значень такого многочлена ефективно реалізується за допомогою схеми Горнера, при цьому многочлен записується в такій формі:

$$f(X) = a0 + x\ (a1 + x\ (a2 + ...\ x\ (an_{-1} + An\ x)\ ...)). \qquad 3.3$$

Незважаючи на простоту, такий підхід дозволяє обчислити значення многочлена за n операцій додавання і n операцій множення. Інший спосіб обчислення значень полінома, що дозволяє обійтися виключно операціями

додавання, що в багатьох випадках дозволяє підвищити швидкодію, відомий як м*етод кінцевих різниць* [3]. для многочлена розглянемо наступні різниці:

$$\Delta 0\ (x) = P\ (x),$$
$$\Delta i + 1\ (x) = \Delta i + 1\ (x + 1) - \Delta i\ (x), \qquad 3.4$$

де i = 0, 1, ..., n - 1. випливає, що

$$P(X + 1) = \Delta 0\ (x + 1),$$
$$\Delta i + 1\ (x + 1) = \Delta i\ (x) - \Delta i + 1\ (x). \qquad 3.5$$

Можна, можливо довести, що кінцева різницю n-го порядку Δn (x) буде однаковою при будь-яких x. Отримані ітераційні співвідношення [6] дозволяють обчислити значення многочлена в точці x + 1 при відомих значеннях многочлена P і різниць Δj (x) для j = 0, 1, ..., n в точці x. Розкриваючи виразу [5], отримуємо слідующие початкові значення різниць Δ*j* для многочлена ступеня n = 5 при x = 0:

$$\Delta 0\ (0) = a0,$$
$$\Delta 1\ (0) = a1 + a2 + a3 + a4 + a5,$$
$$\Delta 2\ (0) = 2a2 + 6a3 + 14a4 + 30a5, \quad (7)$$
$$\Delta 3\ (0) = 6a3 + 36a4 + 150a5,$$
$$\Delta 4\ (0) = 24a4 + 240a5,$$
$$\Delta_5(0) = 120a5. \qquad 3.6$$

Таким чином, алгоритм обчислення значень многочлена за допомогою методу скінченних різниць наступний:

1. Отримати початкові значення різниць Δj в точці x = 0 за формулами (3.6). Значення многочлена в цій точці P (0) = Δ0.

2. Для отримання кожного наступного значення многочлена необхідно відповідно до формул (3.6) ітераційно оновити значення Δj, а потім запам'ятати Δ0 як значення многочлена.

Перший крок алгоритму можна опустити, якщо в якості вхідних даних використовувати не коефіцієнти полінома aj, а попередньо розраховані різниці

Δj. Таким чином, для обчислення одного значення многочлена потрібно всього n - 1 операцій додавання.

Розглянемо тепер задачу апроксимації деякого набору точок (xi, yi), i = 0, 1, ..., m за допомогою многочлена заданої ступеня n. підставляючи xi і yi в (3), отримуємо наступну систему рівнянь:

$$\begin{bmatrix} x_0^n & x_0^{n-1} & \cdots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_m^n & x_m^{n-1} & \cdots & x_m & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_m \end{bmatrix} \quad 3.7$$

Перший множник системи рівнянь, званий матрицею Вандермонда, позначимо як X. Векторами a = (a_0, a_1, ..., an) I y = (y_0, y_1, ..., ym).

Позначимо коефіцієнти полінома і апроксимується значення. Таким чином, система рівнянь може бути представлена в короткій записи: Xa = y.

при $m>n$ система є перевизначення і не має точного рішення. У тому випадку, якщо потрібно апроксимувати дані, що не містять некоректних значень (викидів), розумно застосувати метод найменших квадратів. Позначимо різницю вихідних і апроксимованих значень, також звану залишками, як

$$e=y-Xa. \quad 3.8$$

При вирішенні системи (3.7) за методом найменших квадратів знаходяться такі коеффіціент a многочлена, що сума квадратів різниць залишків e є мінімальною:

$$a = \text{argmin}\sum(P\ (x_i) - yi) \quad 3.9$$

Сума квадратів залишків також може бути записана в наступному вигляді:

$$\sum(P\ (xi)-yi)\ 2= Et\ e = (y-Xa)\ t\ (y\ -Xa) \quad 3.10$$

диференціюючи за a і прирівнюючи похідну до 0 (у цьому випадку помилка буде мінімальною), отримуємо:

$$(X^TX)a = Xy, \quad 3.11$$

Потім знаходимо a:

$$a = (X^TX)\ -1X^Ty = X^+y, \quad 3.12$$

тут $X^+ = (X^TX)\ -1X^T$ є *псевдо зворотною матрицею*[71], яка може бути знайдена за допомогою сингулярного розкладання [2].

Вище була розглянута апроксимація многочленами одновимірних функцій, а таблиці функції зворотного відображення $U(u, v)$ і $V(u, v)$ Двовимірні. Для переходу до двовимірному нагоди пропонується наступний підхід. Кожна v-й рядок таблиці (нехай для визначеності це буде таблиця U) апроксимується многочленом ступеня n через підрядник (тому що найчастіше зображення зберігається в пам'яті через підрядник):

$$U(v, u) \approx Rv(u) = r0(v) + r1(v)\,u + r2(v)\,u2 + ... + rn(v)\,un. \qquad 3.13$$

Далі за формулами, подібним [7], здійснюється перехід від коефіцієнтів rj (v), j = 0, 1, ..., n багаточленів до початкових значень кінцевих різниць Δj (v). Отримані коефіцієнти Δj (v) розглядаються як функції від номера рядка v. Якщо таблиця U є гладкою функцією, то і різниці Δj (v) також будуть гладкими, і, отже, їх також можливо апроксимувати многочленом ступеня m:

$$\Delta j(v) \approx Cj(v) = cj0 + cj1v + cj2v2 + ... + cjmvm. \qquad 3.14$$

Нарешті, ці многочлени представляються у вигляді набору різниць δji, i = 0, 1, ..., m, отриманих за формулами з коефіцієнтів cji.

В результаті вихідна таблиця U (v, u) розміру w × h упаковується в таблицю коефіцієнтів δj і розміру (m + 1) × (n + 1), i = 0, 1, ..., n; j = 0, 1, ..., m. Аналогічним чином упаковується таблиця V (v, u). При апроксимації многочленами необхідно підібрати ступеня поліномів m і n таким чином, щоб з одного боку забезпечити достатню точність апроксимації, а з іншого боку уникнути небажаних осциляцій апроксимуючої функції, які можуть виникнути при її високих порядках (подібних феномену Рунге). Розпакування двовимірної таблиці різниць Δ ji може бути здійснена за наступним алгоритмом:

1. Прийняти, що номер рядка v = 0;
2. Копіювати Δj ← δj 0, для j = 0, 1, ..., n;
3. Прийняти, що номер стовпця u = 0;
4. Записати в вихідну таблицю значення U (u, v) = Δ0;

5. Якщо поточний елемент - останній в рядку (u = w-1), перейти до 8;

6. Оновити значення $\Delta j \leftarrow \Delta j + \Delta j + 1$, для j = 0, 1, ..., n – 1;

7. Оновити номер стовпчика $u \leftarrow u + 1$ і перейти до 4;

8. Якщо поточна рядок остання (v = h-1), завершити роботу алгоритму;

9. Для всіх j = 0, 1, ..., n і i = 0, 1, ..., m - 1 оновити коефіцієнти $\delta ji \leftarrow \delta ji + \delta j\ i + 1$;

10. Оновити номер рядка $v \leftarrow v + 1$ і перейти до 3.

В якості ілюстрації розглянемо застосування описаного способу апроксимації таблиць U (v, u) і V (v, u) для стиснення таблиць виправлення геометричних спотворень. На рисунку 3.1 наведено разностное зображення вихідних і апроксимованих таблиць перетворень. Темні тони відповідають максимальної абсолютної різниці, світлі - мінімальної. Середньоквадратичне відхилення значень, апроксимованих від значень вихідних таблиць, склало близько 0,014 пікселя, а максимальне абсолютне - близько 0,08 пікселя, що значно перевершує точність калібрування камери (в конкретному випадку близько 0,5 пікселя).

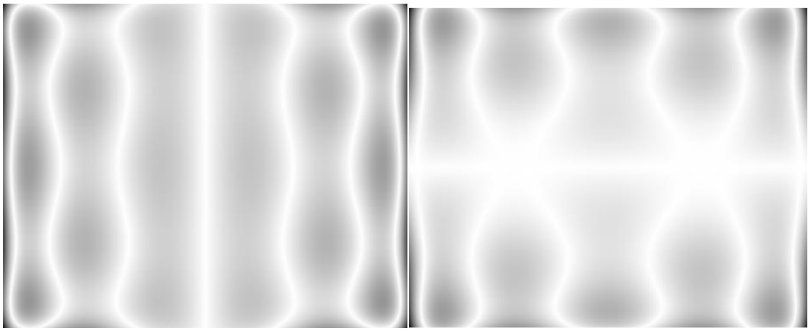

Рис 3.1 – Апроксимація таблиць зображення.

3.2 Опис алгоритму роботи

Для вирішення описаних в попередньому розділі алгоритмічних конструкцій, була складена загальна блок-схема, представлена на рис 3.2.

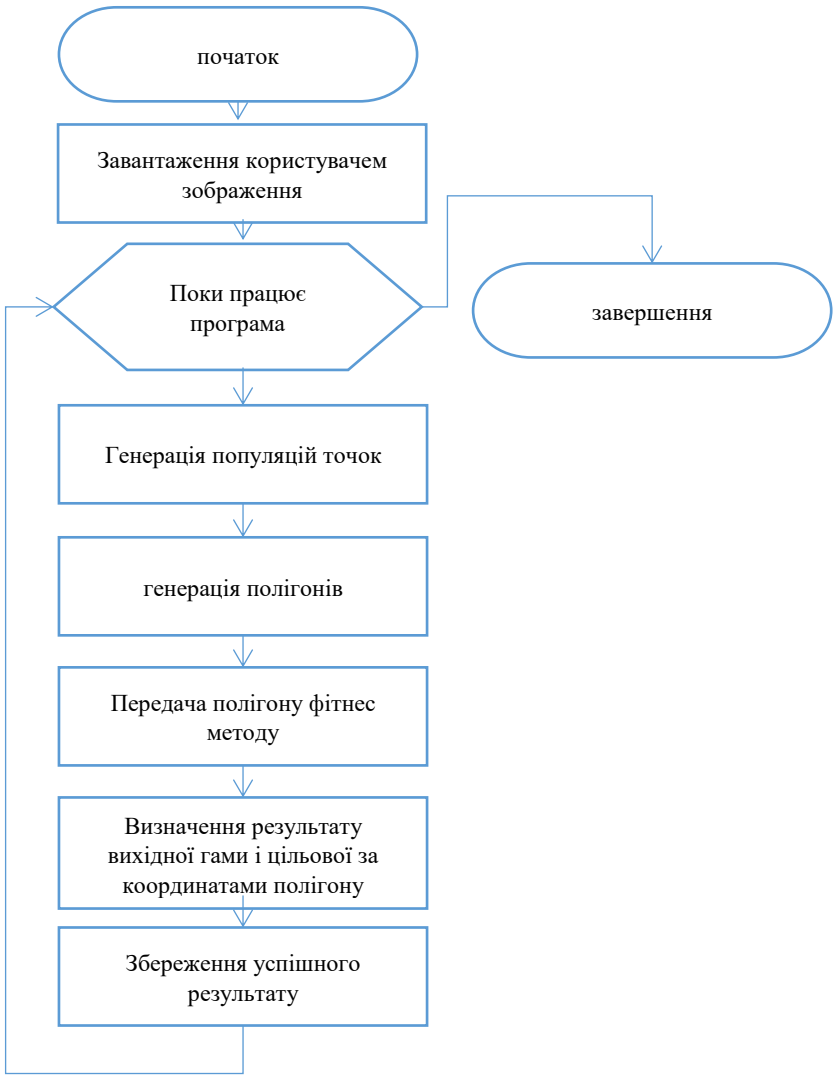

Рис. 3.2 – Загальна блок-схема роботи програми.

Гени в нашому додатку будуть координати точок багатокутника (полігоном) і його колір. Схрещування ми використовувати не будемо, тільки мутацію (це цілком допустимо). Популяція, як не дивно, буде складатися з одного індивіда, і це буде картинка з промальованим на ній полігонами.

Функцією пристосування буде сума похибок кольорів всіх пікселів. Похибка для пікселя визначаємо, як $R_i * R_п + G_i * G_п + B_i * B_п$, де R - різниця між червоною складовою пікселя вихідної картинки і отриманої, G і B - зеленого і синього. Чим менше значення функції пристосованості, тим менше

похибка і тим кращу картинку ми отримали.

Точка має координати X і Y. Процедура SetRandom задає точці випадкові координати, відповідно до розмірів нашої картинки. Процедура Mutate ділиться на три частини, кожна з яких виконується відповідно до своєї заданої ймовірності випадання.

Перша дає шанс точці переміститися в будь-яке місце картинки.
Друга частина переміщує точку на середню відстань, до 20 пікселів.
Третя частина дає нагоду піднятися на невелику відстань, у мене це до 3 пікселів. Для кращого уявлення приводжу вставку коду програми:

```
public class Point implements Cloneable {
    public int X;
    public int Y;

    public Point(int x, int y) {
        X = x;
        Y = y;
    }

    public Point() {
    }

    public void SetRandom() {
        X = ThreadLocalRandom.current().nextInt(0, Helper.Width);
        Y = ThreadLocalRandom.current().nextInt(0, Helper.Height);
    }

    public void Mutate(Workarea workarea) {

        if (rDouble(1.0) <= Helper.MutationPointMoveMaxChance)
        {
            SetRandom();
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationPointMoveMiddleChance)
        {
            X = Math.min(Math.max(0, X + Helper.rInt(-Helper.MiddleRange,
Helper.MiddleRange + 1)), Helper.Width);
            Y = Math.min(Math.max(0, Y + Helper.rInt(-Helper.MiddleRange,
Helper.MiddleRange + 1)), Helper.Height);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationPointMoveNearChance)
        {
            X = Math.min(Math.max(0, X + Helper.rInt(-Helper.NearRange,
Helper.NearRange + 1)), Helper.Width);
            Y = Math.min(Math.max(0, Y + Helper.rInt(-Helper.NearRange,
Helper.NearRange + 1)), Helper.Height);
            workarea.IsChange = true;
        }
    }}
```

Кисть: У кисті є A, R, G, B складові кольору. SetRandom ініціалізує їх випадковими значеннями. У процедурі Mutate кожна зі складових може випадково змінитися відповідно до своєї ймовірності випадання мутації

опущені перевизначені методи:

```
public class Brush implements Cloneable, Serializable {

    public int A;
    public int R;
    public int G;
    public int B;

    public Brush(int a, int r, int g, int b) {
        this.A = a;
        this.R = r;
        this.G = g;
        this.B = b;
    }

    public Brush() {
    }

    public void SetRandom()
    {
        A = rInt(30, 61); // Is need add +1 to bound?
        R = rInt(0, 256);
        G = rInt(0, 256);
        B = rInt(0, 256);
    }


    public void Mutate(Workarea workarea) {
        if (rDouble(1.0) <= Helper.MutationBrushAChance)
        {
            A = rInt(30, 61);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationBrushRChance)
        {
            R = rInt(0, 256);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationBrushGChance)
        {
            G = rInt(0, 256);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationBrushBChance)
        {
            B = rInt(0, 256);
            workarea.IsChange = true;
        }
    }}
```

Полігон включає в себе масив точок і кисть (колір). Процедура SetRandom инициализирует кисть, а також будує мінімальну кількість точок навколо випадково обраної. Мутація полігону являє собою мутацію кисті і кожної точки полігону, а також з певною ймовірністю може відбутися додавання або видалення якоїсь точки. Процедура Draw відмальовує наш полігон:

```
public class Polygon implements Cloneable {

    private List<Point> Points;
    private Brush Brush;

    void SetRandom()
    {
        Points = new ArrayList<>();

        Point center = new Point();
        center.SetRandom();

        for (int i = 0; i < Helper.MinPointsPerPolygon; i++)
        {
            Point point = new Point();
            point.X = Math.min(Math.max(0, center.X + Helper.rInt(-3, 4)),
Helper.Width);
            point.Y = Math.min(Math.max(0, center.X + Helper.rInt(-3, 4)),
Helper.Height);
            Points.add(point);
        }

        Brush = new Brush();
        Brush.SetRandom();
    }

    void Mutate(Workarea workarea)
    {
        if (Helper.rDouble(1.0) <= Helper.MutationPolygonAddPointChance)
        {
            AddPoint();
            workarea.IsChange = true;
        }

        if (Helper.rDouble(1.0)  <= Helper.MutationPolygonDelPointChance)
        {
            DelPoint();
            workarea.IsChange = true;
        }

        Brush.Mutate(workarea);
        Points.forEach((Point p) ->p.Mutate(workarea));
    }

    private void AddPoint()
    {
        if (Points.size() < Helper.MaxPointsPerPolygon)
        {
            Point point = new Point();
            int index = Helper.rInt(1, Points.size() - 1);
            Point p1 = Points.get(index - 1);
            Point p2 = Points.get(index);
            point.X = (p1.X + p2.X) / 2;
            point.Y = (p1.Y + p2.Y) / 2;
            Points.add(index, point);
        }
    }

    private void DelPoint()
    {
        if (Points.size() > Helper.MinPointsPerPolygon)
        {
            int index = Helper.rInt(0, Points.size());
            Points.remove(index);
        }
    }

    void Draw(Graphics g)
    {
            int npoints = Points.size();
            int xpoints[] = new int[npoints];
            int ypoints[] = new int[npoints];
```

```
            for (int i = 0; i<npoints; i++) {
                xpoints[i] = Points.get(i).X;
                ypoints[i] = Points.get(i).Y;
            }

            java.awt.Polygon p = new java.awt.Polygon(xpoints,ypoints,npoints);

            g.setColor(new Color (Brush.R, Brush.G, Brush.B,Brush.A));
            g.fillPolygon(p);

    }
```

Заключний основний клас являє собою робочу область містить наші багатокутники. Істинність властивості IsChange показує, що десь сталася мутація. SetRandom ініціює мінімальне початкове число полігонів. Mutate запускає мутацію для кожного полігону, а також із заданою вірогідністю може статися додавання або видалення полігону. Fitness-method обчислює функцію пристосування, їй передається масив кольорів оригінальної картинки. Ну і функція Draw повертає нашу відмальовану картинку:

```
public class Workarea implements Cloneable, Serializable {
    List<Polygon> Polygons;
    boolean IsChange;
    void SetRandom() {
        Polygons = new ArrayList<>();

        for (int i = 0; i < Helper.MinPolygons; i++)
        {
            AddPolygon();
        }

        IsChange = true;
    }
    void Mutate()
    {
        if (Helper.rDouble(1.0) <= Helper.MutationWorkareaAddPolygonChance)
        {
            AddPolygon();
            IsChange = true;
        }

        if (Helper.rDouble(1.0) <= Helper.MutationWorkareaDelPolygonChance)
        {
            DelPolygon();
            IsChange = true;
        }

        Polygons.forEach(p -> p.Mutate(this));
    }

    private void AddPolygon()
    {
        if (Polygons.size() < Helper.MaxPolygons)
        {
            Polygon polygon = new Polygon();
            polygon.SetRandom();
            Polygons.add(polygon);
        }
    }

    private void DelPolygon()
```

```
    {
        if (Polygons.size() > Helper.MinPolygons)
        {
            int index = Helper.rInt(0, Polygons.size());
            Polygons.remove(index);
        }
    }

    double Fitness(Color[][] colors)
    {
        double fitness = 0;
        BufferedImage img = Draw();


        for (int i = 0; i < Helper.Width; i++)
        {
            for (int j = 0; j < Helper.Height; j++)
            {

                int clr =  img.getRGB(i,j);
                int  red   = (clr & 0x00ff0000) >> 16;
                int  green = (clr & 0x0000ff00) >> 8;
                int  blue  =  clr & 0x000000ff;

                Color c1 = new Color(red,green,blue);
                Color c2 = colors[i][j];

                int r = c1.getRed() - c2.getRed();
                int g = c1.getGreen() - c2.getGreen();
                int b = c1.getBlue() - c2.getBlue();

                fitness += r * r + g * g + b * b;
            }
        }

        return fitness;
    }

    BufferedImage Draw()
    {
        BufferedImage img = new BufferedImage(Helper.Width, Helper.Height,
TYPE_INT_ARGB);

        Graphics g = img.createGraphics();
        for(Polygon p :Polygons) {
            p.Draw(g);
        }
        return img;
    }


    @Override
    protected synchronized Object clone() {
        Workarea newarea = new Workarea();
        newarea.Polygons = new ArrayList<Polygon>();
        Polygons.forEach((p) -> newarea.Polygons.add((Polygon) p.clone()));
        return newarea;
    }
```

Ініціалізируем початкову популяцію, далі робимо копію і мутуючи її, обчислюємо функцію пристосування для нової популяції і, якщо вона менше поточної (якщо нас, чим менше значення функції, тим краще), то Перезаписуємо поточну популяцію нової, більш кращою.

Так само залишився клас налаштувань Helper:

```
class Helper {
    static int Width = 0;
    static int Height = 0;

    static int MinPointsPerPolygon = 3;
    static int MaxPointsPerPolygon = 10;

    static int MinPolygons = 0;
    static int MaxPolygons = 100;

    static int NearRange = 3;
    static int MiddleRange = 20;

    static double MutationPointMoveMaxChance = 0.0007;
    static double MutationPointMoveMiddleChance = 0.0007;
    static double MutationPointMoveNearChance = 0.0007;

    static double MutationBrushAChance = 0.0007;
    static double MutationBrushRChance = 0.0007;
    static double MutationBrushGChance = 0.0007;
    static double MutationBrushBChance = 0.0007;

    static double MutationPolygonAddPointChance = 0.0007;
    static double MutationPolygonDelPointChance = 0.0007;

    static double MutationWorkareaAddPolygonChance = 0.0014;
    static double MutationWorkareaDelPolygonChance = 0.0007;

    private Helper() {
    }

    static int rInt(int start, int end) {
        return ThreadLocalRandom.current().nextInt(start, end);
    }

    static double rDouble(double max) {
        return ThreadLocalRandom.current().nextDouble(max);
    }
}
```

3.3 Виклик і завантаження

Фрагмент автоматизованої інформаційної системи для перетворення растрової графіки у векторну. Основним призначенням розробленого фрагмента є аналіз і відсіювання генетично сильну популяцію з генів на генерується інформацією.

Після запуску програми користувачеві програми відображається вікно, представлене на малюнку 3.5 В даний час програмний продукт перебуває в стані бета-тестування, і оптимізації.

Основними елементами, присутніми на формі є: вікно з різними компонентами завантаження збереження стану, моніторинг результатів фітнес методу, в режимі реального часу відображаються результати відсіювання

популяцій.

3.4 Опис призначеного для користувача інтерфейсу

Розробляється утиліта покликана полегшити роботу користувача і призначена для роботи в автоматичному режимі. Тому інтерфейс програми, що розробляється мінімалістичний і передбачає тільки завантаження файлу, а також вказівку для вивантаження отриманої карти.

Оскільки на даний момент програма проходить бета-тестування, також передбачено виведення допоміжної інформації по кадрам, представлений на рис 3.5-3.13.

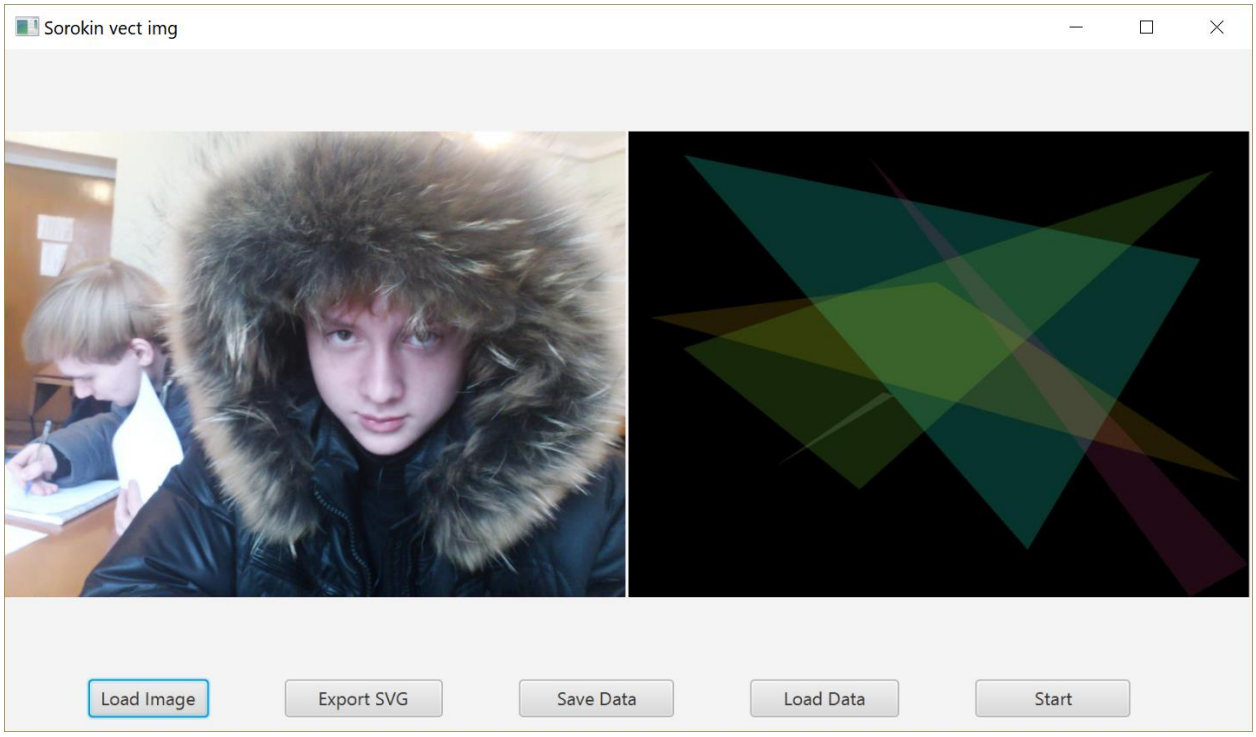

Рис. 3.5 - Головне вікно програми, 15-а популяція.

При натисканні на кнопку «Load Image »відкриється інтерфейс вибору зображення. В якому необхідно вибрати цільову картинку, потім натиснути кнопку «Start», після чого спостерігається перерисовка зображення різними

багатокутниками в стилі RGBA:

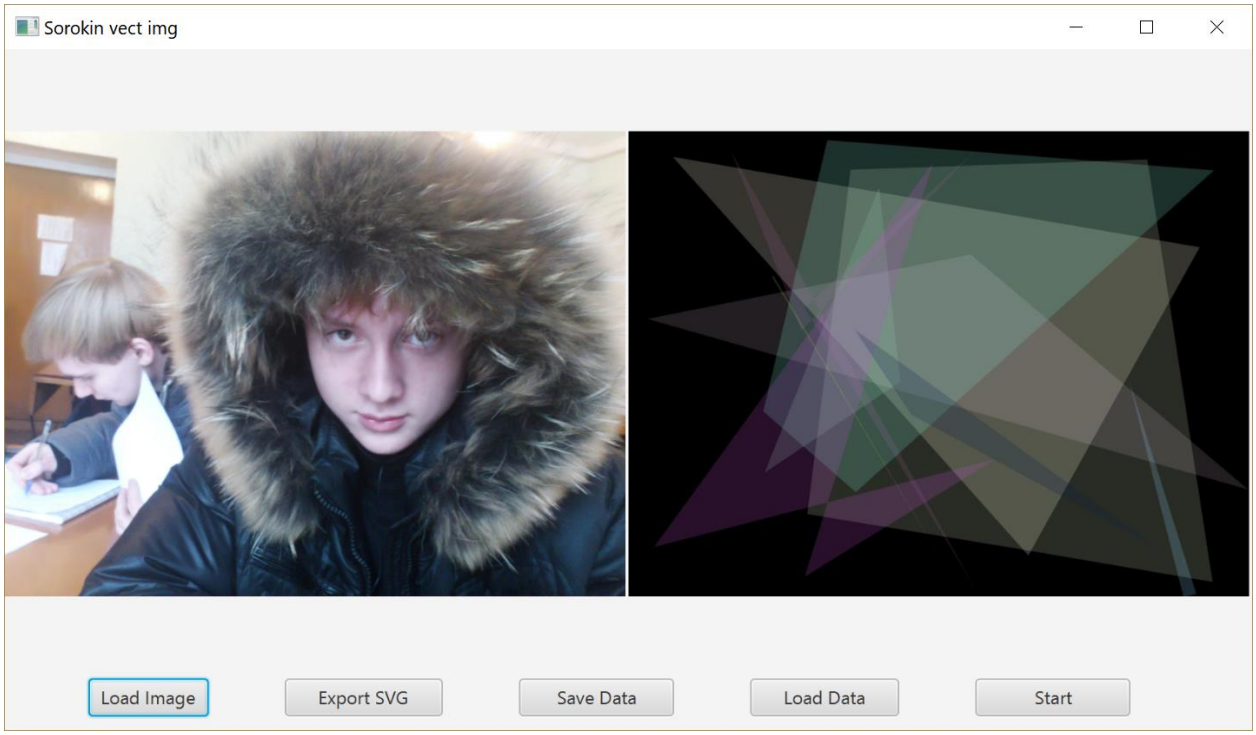

Рис. 3.6 - Нарощування популяцій, 50-а популяція

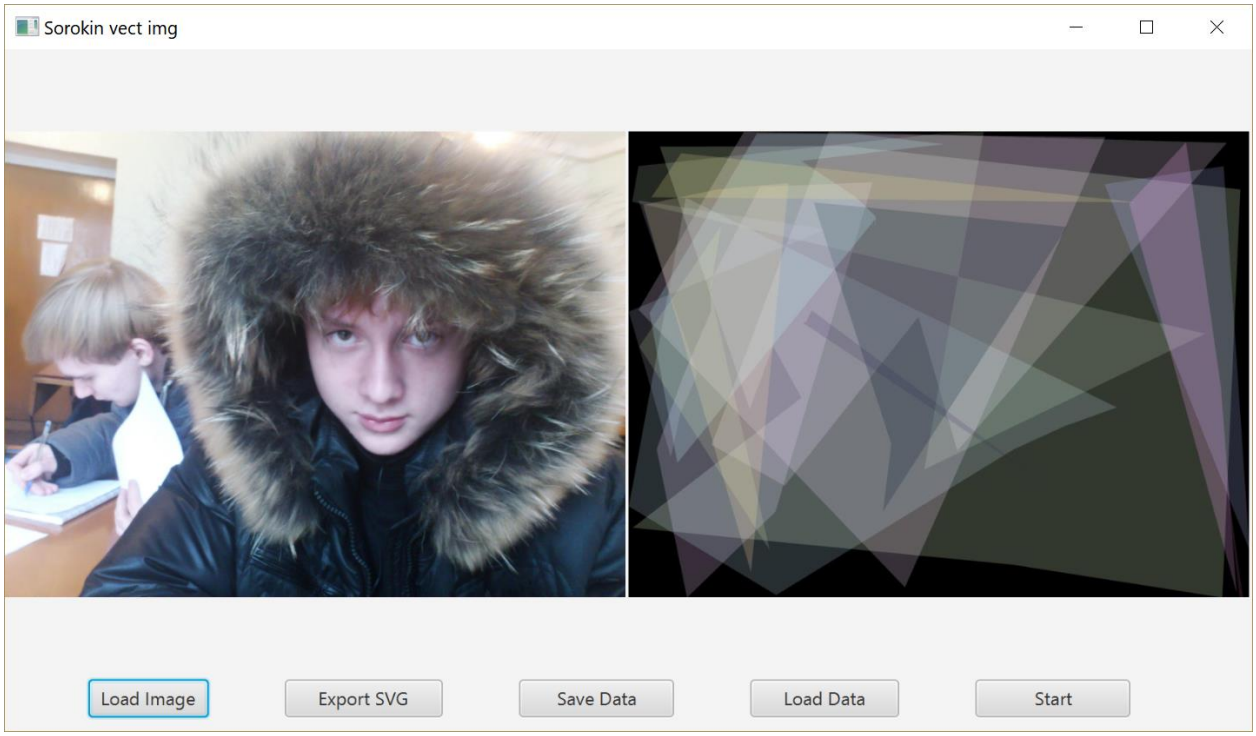

Рис. 3.7 - Нарощування популяцій, 75-а популяція

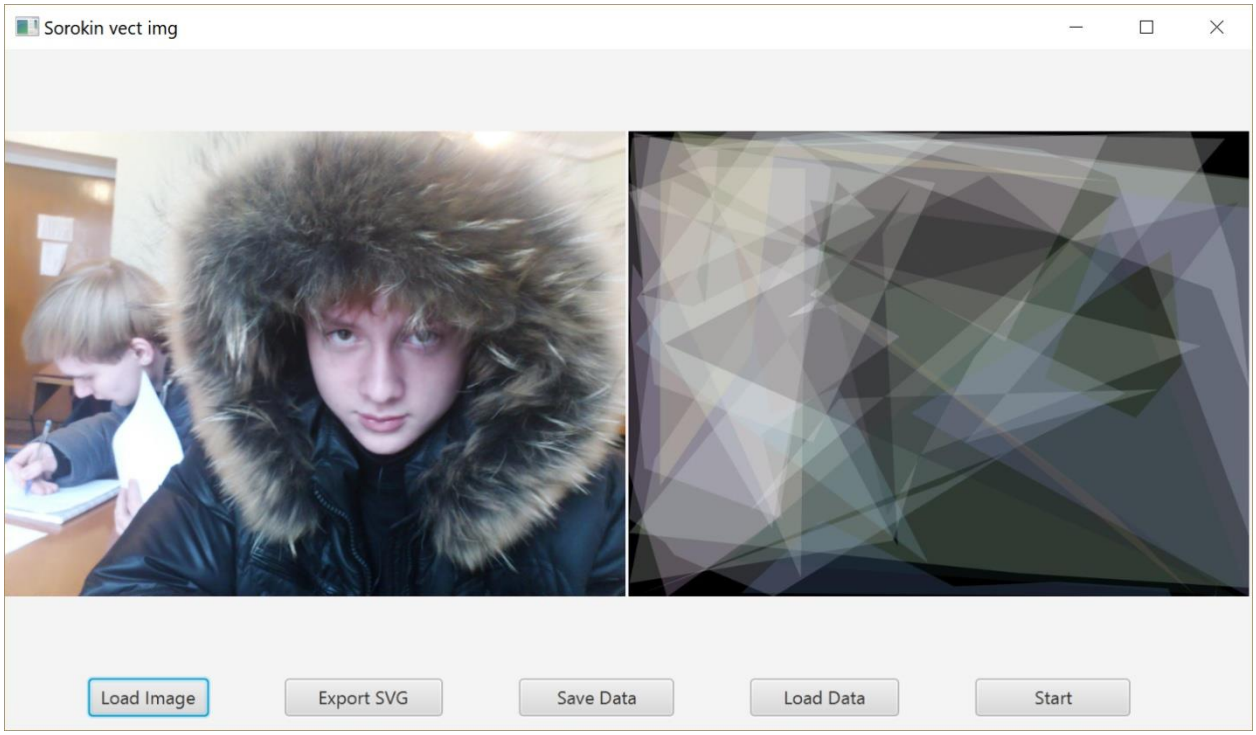

Рис. 3.8 - Нарощування популяцій, 280-ая популяція

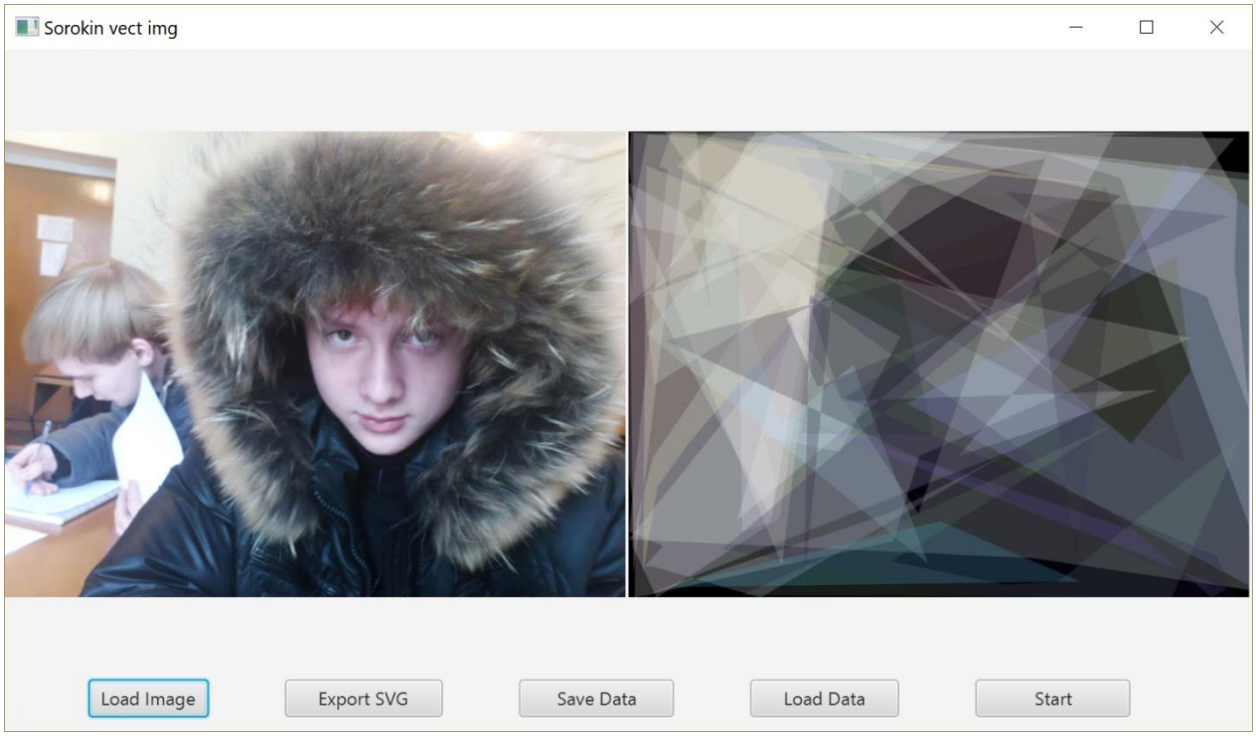

Рис. 3.9 - Нарощування популяцій, 400-ая популяція

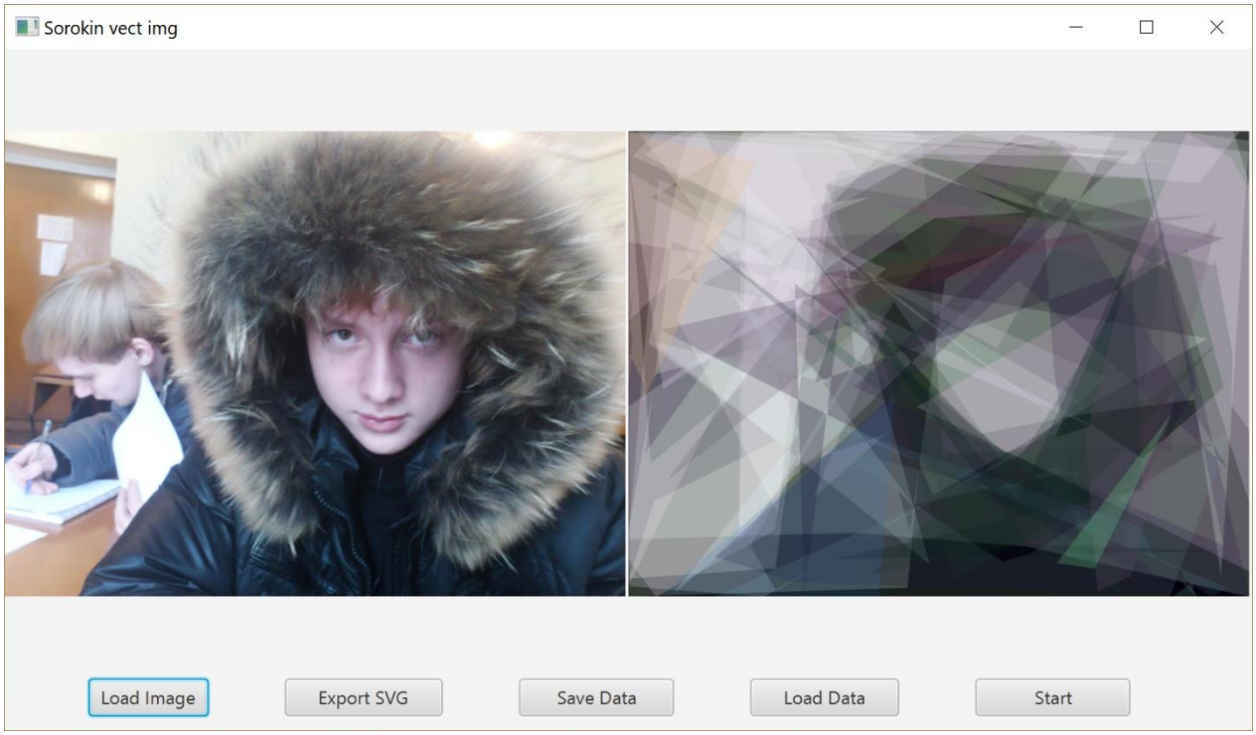

Рис. 3.10 - Нарощування популяцій, 800-ая популяція

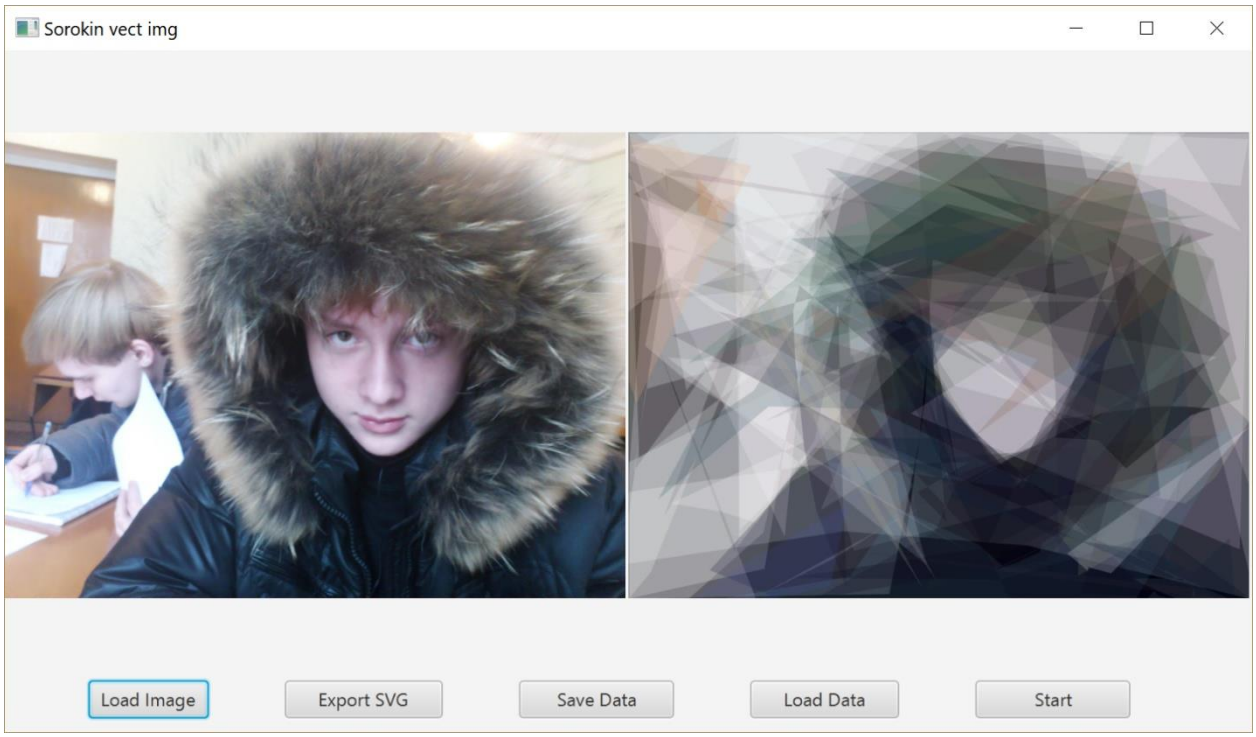

Рис. 3.11 - Нарощування популяцій, 1500-ая популяція

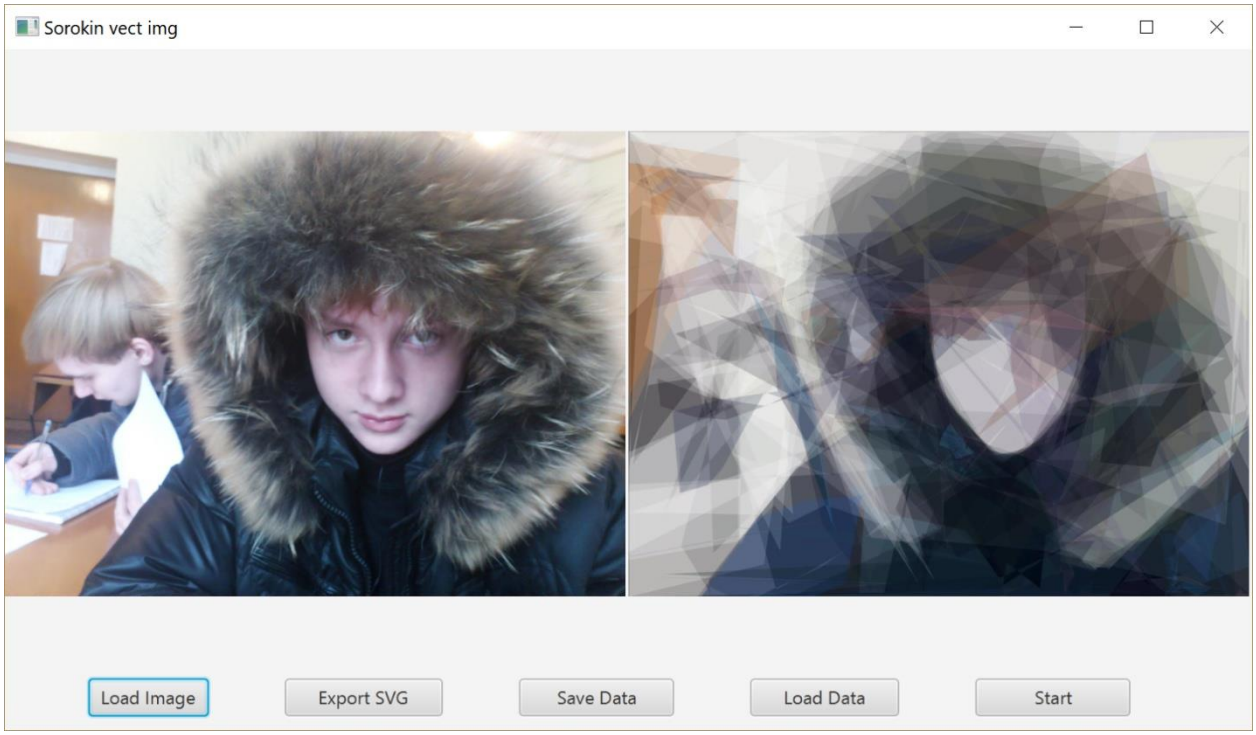

Рис. 3.12 - Нарощування популяцій, 3500-а популяція

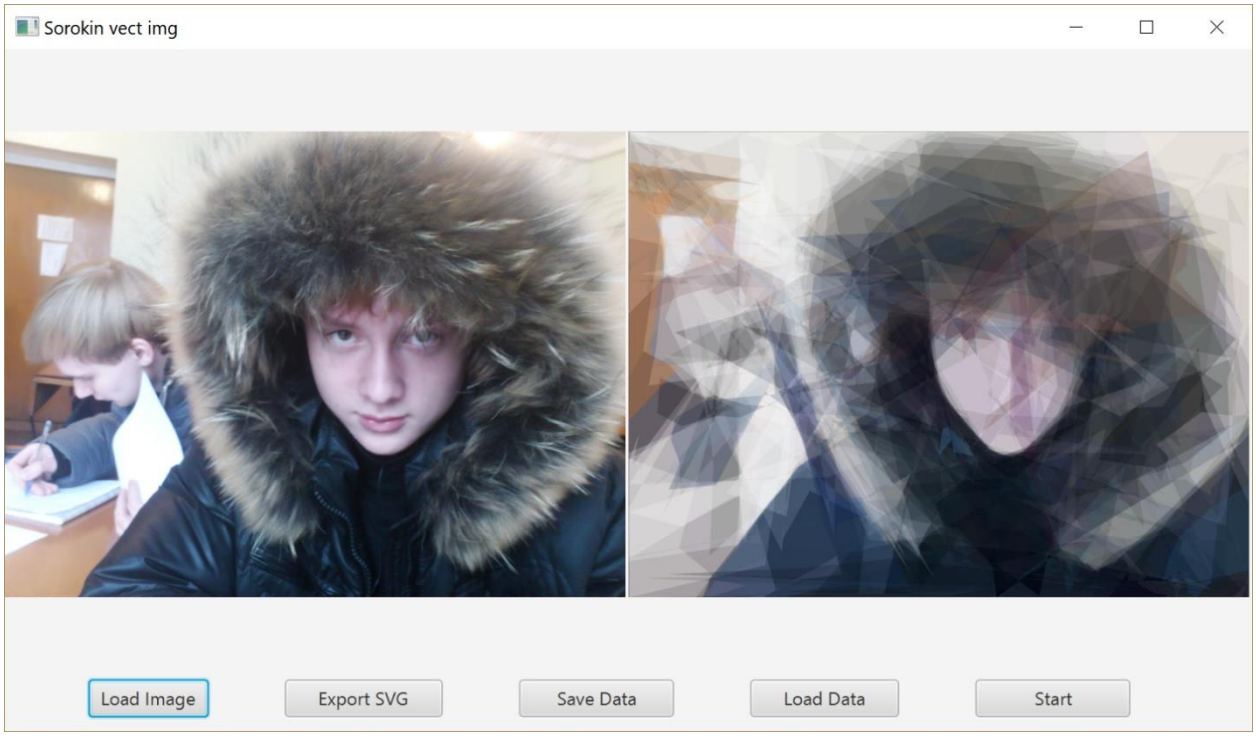

Рис. 3.13 - Нарощування популяцій, 5000-ая популяція

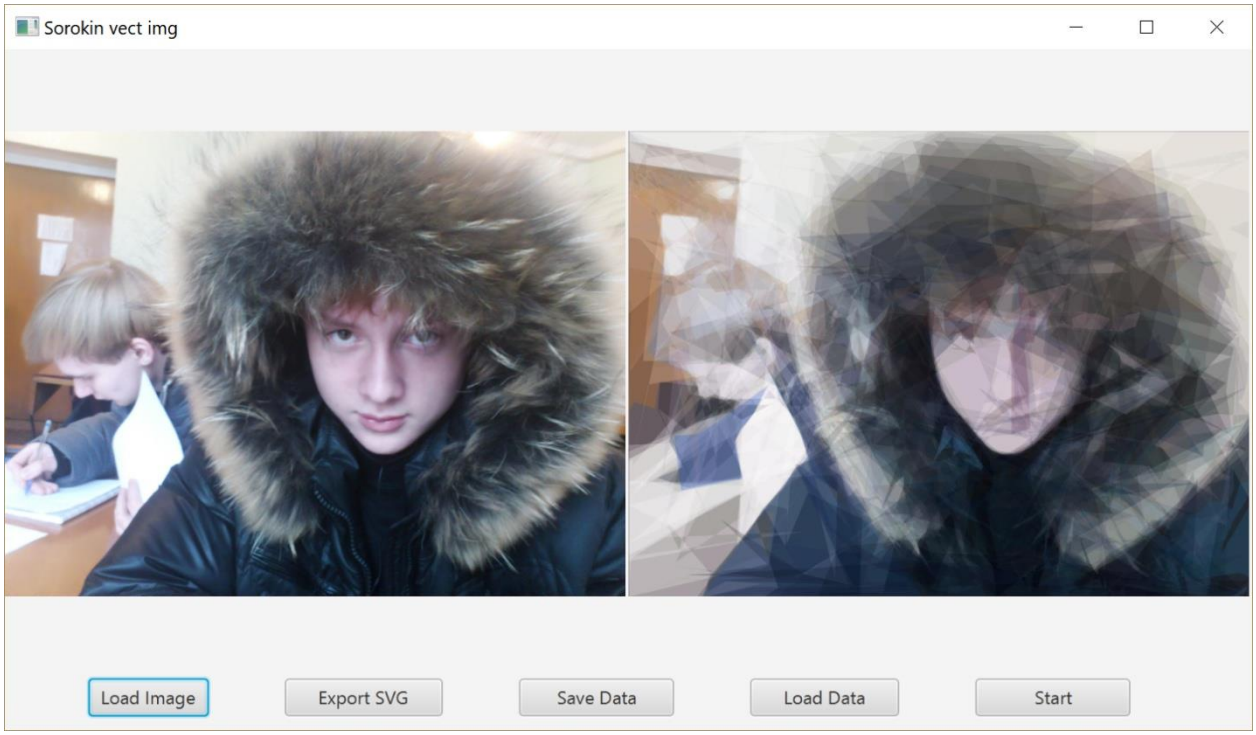

Рис. 3.13 - Нарощування популяцій, 10000-ая популяція

3.5 Обробка помилок і виняткових ситуацій

Виняткова ситуація(Exception) - це переривання нормального ходу роботи програми через неможливість правильно виконати наступні дії. Обробка винятків відома як метод нейтралізації помилок. Об'єкт виключення забезпечує інформацію про тип виникла помилки і змушує потік виконання програми тимчасово призупинитися при генерації або порушення виключення. Примірники об'єктів винятків автоматично створюються при порушенні виключення і руйнуються після його обробки.

Розділ 4

ЕКОНОМІЧНА ЧАСТИНА

При розробці програмного забезпечення важливими етапами є визначення трудомісткості розробки ПЗ, розрахунок витрат на створення програмного продукту і аналіз ринку збуту розробленого програмного забезпечення.

4.1 Визначення трудомісткості розробки програмного забезпечення

Початкові дані:

1) передбачуване число операторів - 720;

2) коефіцієнт складності програми - 1,6;

3) коефіцієнт корекції програми в ході її розробки - 0,4;

4) годинна заробітна плата програміста, грн / год - 50;

5) вартість машино-години ЕОМ, грн / год - 5;

6) коефіцієнт кваліфікації програміста - 1.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста, тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість можна розрахувати за формулою:

$t = t_o + t_u + t_a + t_n + t_{отт} + t_д$ людино-годин,

Де to витрати праці на підготовку і опис поставленого завдання

(приймається = 50);

T_u витрати праці на дослідження алгоритму розв'язання задачі;

T_a витрати праці на розробку блок-схеми алгоритму;

$T_п$ витрати праці на програмування по готовій блок-схемі;

$T_{отл}$ витрати праці на налагодження програми на ЕОМ;

$T_д$ витрати праці на підготовку документації.

Складові витрати праці визначаються виходячи з умовного числа операторів в розробляється ПО.

Умовне число операторів:

$$Q = q \cdot C \cdot (1 + p) \qquad 4.1$$

Q передбачуване число операторів (q = 720).

C коефіцієнт складності програми. Коефіцієнт складності завдання З характеризує відносну складність програми по відношенню до так званої типової задачі, що реалізує стандартні методи рішення, складність якої прийнята рівною одиниці (величина С лежить в межах від 1,25 до 2). Для даного програмного продукту, з урахуванням великої кількості і різноманітності оброблюваної інформації і складності складання звітів, коефіцієнт складності завдання візьмемо 1,6.

P коефіцієнт корекції програми в ході її розробки. Коефіцієнт корекції програми р - збільшення обсягу робіт за рахунок внесення змін до алгоритму або програму за результатами уточнення постановок. В даному випадку програма вимагала численних доробок. З урахуванням цього візьмемо коефіцієнт рівний 0,4.

$$Q = 720 \cdot 1{,}6 \cdot (1 + 0{,}3) = 1479 \qquad 4.2$$

Витрати праці на вивчення опису завдання визначається з урахуванням уточнення опису і кваліфікації програміста.

$$t_u = \frac{Q \cdot B}{(75\ldots85)K} \qquad 4.3$$

В коефіцієнт збільшення витрат праці внаслідок недостатнього опису завдання. Коефіцієнт збільшення витрат праці в залежності від складності завдання приймається від 1,25 до 2, внаслідок недостатнього опису рішення задачі приймемо В = 1,8.

К коефіцієнт кваліфікації програміста, який визначається від стажу роботи за даною спеціальністю. Коефіцієнт становить: для працюючих до двох років 0,8; від двох до трьох років 1,0; від трьох до п'яти років 1,1 1,2; від п'яти до семи 1,3 1,4; понад сім років 1,5 1,6. Тому приймемо К = 1.

$$t_u = \frac{1613}{80} = 36{,}288 \text{ люд.-годин}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20\ldots25)K} \qquad 4.4$$

$$t_a = \frac{1497}{20} = 29{,}94 \text{ люд.-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25)K} \qquad 4.5$$

$$t_n = \frac{1497}{25} = 59{,}9 \text{ люд.-годин.}$$

Витрати праці на налагодження програми за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)K} \qquad 4.6$$

$$t_{отл} = \frac{1497}{5} = 299{,}52 \text{ люд.-годин.}$$

Витрати праці на налагодження програми за умови комплексного

налагодження завдання:

$$T^{к}_{отл} = 1{,}5 \cdot t_{отл}, \quad 4.7$$

$$T^{к}_{отл} = 1{,}5 \cdot 299{,}52 = 449{,}28 \text{ люд.-годин.}$$

Витрати на підготовку документації:

$$t_{д} = t_{др} + t_{до}, \quad 4.8$$

Де tдр трудомісткість підготовки матеріалів і рукописи;

Тдо трудомісткість редагування, друку і оформлення документації.

Трудомісткість підготовки матеріалів і рукописи визначається за формулою:

$$t_{др} = \frac{Q}{(15..20)K} \quad 4.9$$

$$t_{др} = \frac{1497}{18} = 83{,}2 \text{ люд.-годин.}$$

Трудомісткість редагування, друку і оформлення документації:

$$t_{до} = 0{,}75 t_{др} \quad 4.10$$

Т до = 0,75 * 83,2 = 62,4 люд.-годин.

Витрати на підготовку документації складуть:

Тд = 83,2+ 62,4 = 145,6 люд.-годин.

Отримуємо трудомісткість розробки ПЗ:

Т = 50 + 29,952 + 29,94 + 59,904 + 299,52 + 145 = 614,316 люд.-годин

Таким чином, трудомісткість розробки програмного забезпечення становить 614,316 люд.-годин.

4.2 Витрати на створення програмного забезпечення

Витрати на створення ПЗ (Кпо) включають витрати на заробітну плату розробників програми (Зз / п), яка визначається множенням сумарної трудомісткості розробки ПЗ (t) на середню заробітну плату програміста з

нарахуваннями та вартості машинного часу на налагодження.

$$K_{ПО} = З_{ЗП} + З_{МВ} \qquad 4.11$$

Заробітна плата розробників визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \qquad 4.12$$

Де t загальна трудомісткість, люд.-годин.

Спр середня годинна заробітна плата програміста, грн / год.

Спр = 20 грн. / Год.

З зп = 710,8 · 20 = 14216 грн.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} \cdot C_{мч}, \qquad 4.13$$

Де tотл трудомісткість налагодження програми на ЕОМ, ч.

СМЧ вартість машинного часу ЕОМ грн / год.

З мв = 299,52 · 5 = 1497,6 грн.

Витрати на створення програмного забезпечення складуть:

Кпо = 14216 + 1497,6 = 15713,6 грн.

Певні таким чином витрати на створення програмного забезпечення є одноразовими капітальними витратами на створення АС.

Очікуваний період створення ПО:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.}, \qquad 4.14$$

Де Bk число розробників;

Fp місячний фонд робочого часу (при 40 годинному робочому тижні Fp = 176 годин).

$$T = \frac{710,8}{1 \cdot 176} = 4 \text{ міс.}$$

Таким чином, очікувана тривалість розробки складе близько 4 місяців, а витрати на створення програмного забезпечення - 15713,6 грн.

4.3 Маркетингові дослідження ринку збуту розробленого програмного продукту

Маркетингові дослідження розробленого програмного забезпечення необхідно провести для вивчення ринку збуту і потенційних покупців, з метою його реалізації не тільки підприємству-замовнику. З метою зменшення грошових витрат, підприємства широко використовують вітчизняні програмні продукти в зв'язку з їх дешевизною, адаптованість до місцевих умов господарювання і простотою сервісного обслуговування, тому розроблено програмне забезпечення буде мати переваги в ціновій політиці і зручності використання в порівнянні з зарубіжними аналогами.

Основними покупцями розробленого програмного продукту будуть фізичні та юридичні особи, які мають справу зі створенням зображення, для публікації, або в рекламних цілях. Головною метою покупки буде отримання послуг, пов'язаних з відтворенням цільового зображення на векторній основі.

Головними конкурентами розробленого програмного забезпечення будуть додатки, призначені для роботи з растровою і векторною графікою, які мають функції перетворення в вектор. Найвідомішими з таких програм є Adobe Photoshop, Corel Draw, які вимагають наявності дизайнера для проведення подібних конвертацій.

Перевага розробленого в даному дипломному проекті програмного забезпечення полягає в комбінації з оригінальним алгоритму конвертації зображення з растрової у векторну графіку, використовуючи генетичні алгоритми, а також в оптимальній вартості ліцензії на відповідний період. Для даного програмного засобу, його алгоритмів, аналогів не існує тому через це не можна порівняти з конкурентами.

4.4 Оцінка економічної ефективності впровадження програмного забезпечення

Дане ПО дозволяє скоротити люд.-годинни, витрачених дизайнерами на конвертування зображення у векторну графіку. Через відсутність аналогів на ринку не можна порахувати в якому обсязі необхідні інвестиції, термін окупаємості, так, як і очікуваний прибуток. Економічний ефект після впровадження прорахувати неможливо.

Висновки

В ході проведеної роботи були вивчені різні алгоритми зображень з метою отримати уявлення про сильні та слабкі сторони різних підходів і вибрати найбільш підходящий алгоритм для заданої сфери застосування - області дизайну і ілюстративної графіки. У процесі вивчення були запропоновані та протестовані нові способи вирішення задач сегментації растра на фігури і лінійні об'єктів, апроксимації полігонів, і граничних ліній в контексті використання алгоритму, засновані на площадковому моделюванні об'єктів запропонованого в роботах [25] та [48].

В результаті був розроблений генетичний алгоритм, реалізує функції векторизації повнокольорових растрових зображень з автоматичним виділенням ліній і полігонів, аналізу апроксимації даних, що генерують при нарощуванні поколінь генів.

У роботі над проектом було проведено аналіз існуючих механізмів їх структури і компонентів, що відповідають за мутацію, схрещування об'єктів, так само аналіз в досягненні кращого результату генерації і порівняння популяцій полігонів. Крім того, було проведено огляд і опис методів розпізнавання і аналізу графічної інформації, розкладання зображення на спектри канали, докладний опис порівняння вихідних даних з досягнутим результатом.

При впровадженні даної системи в механізм векторизації було помічено гранично високий час роботи програми при зображеннях понад 3264x2448px, в порівнянні з 3072 × 1620px - зросла в 1,4 рази.

Список використаних джерел

1. Воронцов К. В. Про проблемно-орієнтованої оптимізації базисів завдань розпізнавання - http://www.ccas.ru/frc/papers/voron98jvm.pdf.

2. Воронцов К. В. Локальні базиси в алгебраїчному підході до проблеми розпізнавання. - Дисертація на здобуття наукового ступеня к.ф.-м.н., М .: ВЦ РАН. - 1999. -http://www.ccas.ru/frc/thesis/VoronCanDisser.pdf.

3. Воронцов К. В. Оптимізаційні методи лінійної та монотонної корекції в алгебраїчному підході до проблеми розпізнавання - 2000.

4. Воронцов К. В. Проблемно-орієнтовані методи алгебраїчного підходу. - 2002. http://www.ccas.ru/frc/papers/voron02po4.pdf.

5. Воронцов К. В. Огляд сучасних досліджень з проблеми якості навчання алгоритмів - 2004.

6. Журавльов Ю. І. Коректні алгебри над множинами некоректних (евристичних) алгоритмів. частина I // Кібернетика. - 2002.

7. Журавльов Ю. І. Коректні алгебри над множинами некоректних (евристичних) алгоритмів. частина II // Кібернетика. - 2002. - № 6. - С. 21-27.

8. Журавльов Ю. І. Коректні алгебри над множинами некоректних (евристичних) алгоритмів. частина III // Кібернетика. - 2004. - № 2. - С. 35-43.

9. Журавльов Ю. І. Про алгебраїчному підході до вирішення завдань розпізнавання або класифікації // Проблеми кібернетики. - 1978. - Т. 33. - С. 5-68.

10.Рудаков К. В. Про деякі універсальних обмеження для алгоритмів класифікації // ЖВМіМФ. - 1986. - Т. 26, № 11. - С. 1719-1730.http://www.ccas.ru/frc/papers/rudakov86universal.pdf.

11.Рудаков К. В. Про симетричних і функціональних обмеженнях для алгоритмів класифікації // ДАН СРСР. - 1987. - Т. 297, № 1. - С. 43-46.

12.Рудаков К. В. Повнота і універсальні обмеження в проблемі корекції евристичних алгоритмів класифікації // Кібернетика. - 1987. - № 3. - С. 106-109.

13.Рудаков К. В. Універсальні і локальні обмеження в проблемі корекції евристичних алгоритмів // Кібернетика. - 1987. - № 2. - С. 30-35.

14.Рудаков К. В. Про алгебраїчної теорії універсальних і локальних обмежень для задач класифікації // Розпізнавання, Класифікація, Прогноз. - 1988. - Т. 1. - С. 176-200.

15.Рудаков К. В. Алгебраїчна теорія універсальних і локальних обмежень для алгоритмів розпізнавання. - Дисертація на здобуття наукового ступеня д.ф.-м.н., М .: ВЦ РАН. - 1 992.

16.Рудаков К. В., Воронцов К. В. Про методи оптимізації та монотонної корекції в алгебраїчному підході до проблеми розпізнавання // Докл. РАН. - 1999. - Т. 367, № 3. - С. 314-317.

17.Рудаков К. В., Трофимов С. В. Алгоритм синтезу коректних процедур розпізнавання для задач з непересічними класами // ЖВМіМФ. - 1988. - Т. 28, № 9. - С. 1431-1434.

18. Angeline PJ Adaptive and self-adaptive evolutionary computations // Computational Intelligence: A Dynamic Systems Perspective / Ed. by M. Palaniswami, Y. Attikiouzel. - IEEE Press, 1995. - Pp. 152-163.

19.Angeline PJ, Pollack JB Competitive environments evolve better solutions for complex tasks // Proceedings of the 5th International Conference on Genetic Algorithms (GA-93). - 1993. - Pp. 264-270.

20. Back T. Self-adaptation in genetic algorithms.

21.Back T. Optimization by means of genetic algorithms // 36th International Scientific Colloquium / Ed. by E. Kohler. - Technical University of Ilmenau: 1991. - Pp. 163-169.

22. Back T., Hoffmeister F. Global optimization by means of evolutionary algorithms.

23.Back T., Hoffmeister F., Schwefel H. A survey of evolution strategies. - 1991.

24.Baker JE Adaptive Selection Methods for Genetic Algorithms / Ed. by

P. of an International Conference on Genetic Algorithms, e. their Applications (JJ Grefenstette. - Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

25. Bauer E., Kohavi R. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants // Machine Learning. - 1999. - Vol. 36, no. 1-2. - Pp. 105-139.

26. Bhanu B., Krawiec K. Coevolutionary construction of features for transformation of representation in machine learning.

27. Blake C., Merz C. UCI repository of machine learning databases: Tech. rep .: Department of Information and Computer Science, University of California, Irvine, CA, 1998..

28. Breiman L. Bagging predictors // Machine Learning. - 1996. - Vol. 24, no. 2. - Pp. 123-140.

29. Breiman L. Bias, variance, and arcing classifiers: Tech. Rep. 460: Statistics Department, University of California, 1996..

30. Breiman L. Arcing classifiers // The Annals of Statistics. - 1998. - Vol. 26, no. 3. - Pp. 801-849.

31. Bucci A., Pollack J. Order-theoretic analysis of coevolution problems: Coevolutionary statics. - 2002.

32. Bucci A., Pollack J. A mathematical framework for the study of coevolution. - 2003.

33. Busetti F. Genetic algorithms overview.

34. Dietterich TG An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization // Machine Learning. - 2000. - Vol. 40, no. 2. - Pp. 139-157.

35. Domingos P. Why does bagging work? a bayesian account and its implications // Knowledge Discovery and Data Mining. - 1997. - Pp. 155-158.

36. Dymitr Ruta BG Analysis of the correlation between majority voting error and the diversity.

37. Ficici SG, Melnik O., Pollack JB A game-theoretic investigation of selection methods used in evolutionary algorithms // Proceedings of the 2000

Congress on Evolutionary Computation CEC00. - La Jolla Marriott Hotel La Jolla, California, USA: IEEE Press, 6-9 2000. - P. 880.

38. Ficici SG, Pollack JB A game-theoretic approach to the simple coevolutionary algorithm // Parallel Problem Solving from Nature - PPSN VI 6th International Conference / Ed. by H.-PS Marc Schoenauer, Kalyanmoy Deb, G¨unter Rudolph,

39. Xin Yao, Evelyne Lutton, Juan Julian Merelo. - Paris, France: Springer Verlag, 2000..

40. Ficici SG, Pollack JB Pareto optimality in coevolutionary learning // Lecture Notes in Computer Science. - 2001. - Vol. 2159. - Pp. 316+.

41. Fidelis MV, Lopes HS, Freitas AA Discovering comprehensible classification rules a genetic algorithm // Proceedings of the 2000 Congress on

Evolutionary Computation CEC00. - La Jolla Marriott Hotel La Jolla, California, USA: IEEE Press, 6-9 2000. - Pp. 805-810.

42. Freitas A. A survey of evolutionary algorithms for data mining and knowledge discovery. - 2001.

43. Freund Y. Boosting a weak learning algorithm by majority // COLT: Proceedings of the Workshop on Computational Learning Theory. - Morgan Kaufmann Publishers, 1990..

44. Freund Y. An adaptive version of the boost by majority algorithm // COLT: Proceedings of the Workshop on Computational Learning Theory. - Morgan Kaufmann Publishers, 1999..

45. Freund Y., Schapire RE A decision-theoretic generalization of on-line learning and an application to boosting // European Conference on Computational Learning Theory. - 1995. - Pp. 23-37.

46. Freund Y., Schapire RE Experiments with a new boosting algorithm // International Conference on Machine Learning. - 1996. - Pp. 148-156.

47. Freund Y., Schapire RE A short introduction to boosting // J. Japan. Soc. For Artif. Intel. - 1999. - Vol. 14, no. 5. - Pp. 771-780.

48. Giordana A., Neri F. Search-intensive concept induction // Evolutionary Computation. - 1995. - Vol. 3, no. 4. - Pp. 375-419.

49. Grove AJ, Schuurmans D. Boosting in the limit: Maximizing the margin of learned ensembles // AAAI / IAAI. - 1998. - Pp. 692-699.

50. Hancock PJB An empirical comparison of selection methods in evolutionary algorithms // Evolutionary Computing, AISB Workshop. - 1994. - Pp. 80-94.

51. Hinterding R., Gielewski H., Peachey TC The nature of mutation in genetic algorithms // Proceedings of the Sixth International Conference on Genetic

Algorithms / Ed. by L. Eshelman. - San Francisco, CA: Morgan Kaufmann, 1995. - Pp. 65-72.

52. Ho TK The random subspace method for constructing decision forests // IEEE Transactions on Pattern Analysis and Machine Intelligence. - 1998. - Vol. 20, no. 8. - Pp. 832-844.

53. Horn J. Finite Markov Chain Analysis of Genetic Algorithms with Niching // Proceedings of the Fifth International Conference on Genetic Algorithms / Ed.

By S. Forrest. - San Mateo, California: Morgan Kaufmann Publishers, 1993. - Pp. 110-117.

54. JHH Adaptation in Natural and Artificial Systems. - University of Michigan Press, 1975.

55. Jin R., Liu Y., Si L., Carbonell J., Hauptmann A. A new boosting algorithm using input-dependent regularizer // The 20-th International Conference on Machine Learning. - 2003.

56. Kuncheva L., Skurichina M., Duin R. An experimental study on diversity for bagging and boosting with linear classifiers. - 2002.

57. Kuncheva L., Whitaker C. Measures of diversity in classifier ensembles. - 2000.

58. Luke S., Wiegand RP When coevolutionary algorithms exhibit evolutionary dynamics // In Workshop Proceedings of the 2003 Genetic and Evolutionary Computation Conference. - 2002.

59. Maclin R., Opitz D. An empirical evaluation of bagging and boosting // AAAI / IAAI. - 1997. - Pp. 546-551.

60. Mason L. Margins and Combined Classifiers: Ph.D. thesis / Australian National University. - 1 999.

61. Paredis J. Coevolutionary computation // Artificial Life. - 1995. - Vol. 2, no. 4. - Pp. 355-375.

62. Pena-Reyes CA Coevolutionary fuzzy modeling.

63. Pena-Reyes CA Island fuzzy coco: Island model-based coevolution of fuzzy systems.

64. Potter MA The Design and Analysis of a Computational Model of Cooperative Coevolution: Ph.D. thesis. - 1997.

65. Potter MA, De Jong K. A cooperative coevolutionary approach to function optimization // Parallel Problem Solving from Nature - PPSN III / Ed. by Y. Davidor, H.-P. Schwefel, R. M¨anner. - Berlin: Springer, 1994. - Pp. 249-257.

66. Potter MA, De Jong K. Evolving neural networks with collaborative species // Proc. of the 1995 Summer Computer Simulation Conf. - The Society of Computer Simulation, 1995. - Pp. 340-345.

67. Potter MA, De Jong KA Cooperative coevolution: An architecture for evolving coadapted subcomponents // Evolutionary Computation. - 2000. - Vol. 8, no. 1. - Pp. 1-29.

68. Radcliffe NJ The algebra of genetic algorithms: Tech. Rep. TR92-11: 1992.

69. Ratsch G., Onoda T., Muller KR An improvement of adaboost to avoid overfitting.

70. Ratsch G., Onoda T., Muller K.-R. Soft margins for AdaBoost // Machine Learning. - 2001. - Vol. 42, no. 3. - Pp. 287-320.

71. Rosin CD, Belew RK New methods for competitive coevolution // Evolutionary Computation. - 1997. - Vol. 5, no. 1. - Pp. 1-29. Ruta D., Gabrys B. Classifier selection for majority voting.

72. Saharon Rosset JZ, Hastie T. Margin maximizing loss functions.

73. Schapire R. The boosting approach to machine learning: An overview // MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA. - 2001.

74. Schapire RE Theoretical views of boosting and applications // Algorithmic Learning Theory, 10th International Conference, ALT 99, Tokyo, Japan, December 1999 року, Proceedings. - Vol. 1720. - Springer, 1999. - Pp. 13-25.

75. Schapire RE, Freund Y., Lee WS, Bartlett P. Boosting the margin: a new explanation for the effectiveness of voting methods // Annals of Statistics. - 1998. - Vol. 26, no. 5. - Pp. 1651-1686.

76. Schapire RE, Singer Y. Improved boosting using confidence-rated predictions // Machine Learning. - 1999. - Vol. 37, no. 3. - Pp. 297-336.

77. Skurichina M., Duin RPW Limited bagging, boosting and the random subspace method for linear classifiers.

78. Skurichina M., Kuncheva L., Duin R. Bagging and boosting for the nearest mean classifier: Effects of sample size on diversity and accuracy // Multiple Classifier Systems (Proc. Third International Workshop MCS, Cagliari, Italy) / Ed. by JKF Roli. - Vol. 2364. - Springer, Berlin, 2002. - Pp. 62-71.

79. Sloman A. The semantics of evolution: Trajectories and trade-offs in design space and niche space // Lecture Notes in Computer Science. - тисяча дев'ятсот дев'яносто вісім.

80. Tsymbal A., Puuronen S. Ensemble feature selection with the simple Bayesian classification in medical diagnostics.

81. Twycross J., Cayzer S. An immune-based approach to document classification.

82. Whitley D. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best // Proceedings of the Third

International Conference on Genetic Algorithms / Ed. by JD Schaffer. - San Mateo, Kaufman, 1989.

83. Whitley D. An overview of evolutionary algorithms: practical issues and common pitfalls // Information and Software Technology. - 2001. - Vol. 43, no. 14.

84. Whitley D., Rana SB, Heckendorn RB Island model genetic algorithms and linearly separable problems // Evolutionary Computing, AISB Workshop. - 1997. - Pp. 109-125.

85. Wiegand R., Jansen T. The cooperative coevolutionary (1 + 1) ea // MIT Press. - 2003.

86. Wiegand R., Liles W., De Jong K. Analyzing cooperative coevolution with evolutionary game theory. - 2002.

87. Wiegand RP An Analysis of Cooperative Coevolutionary Algorithms: Ph.D. thesis / George Mason University, Fairfax, VA. - 2004.

88. Wiegand RP, Liles WC, De Jong KA An empirical analysis of collaboration methods in cooperative coevolutionary algorithms // In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). - 2001.

89. Wright AH Genetic algorithms for real parameter optimization // Foundations of genetic algorithms / Ed. by GJ Rawlins. - San Mateo, CA: Morgan Kaufmann, 1991. - Pp. 205-218.

90. Wright AH, Zhao Y. Markov chain models of genetic algorithms Proceedings of the Genetic and Evolutionary Computation Conference / Ed. by W. Banzhaf J. Daida, AE Eiben, MH Garzon, V. Honavar, M. Jakiela, RE Smith.

91. Orlando, Florida, USA: Morgan Kaufmann, 13-17 1999. - Pp. 734-741.

92. Yang J., Honavar V. Feature subset selection using A genetic algorithm // Genetic Programming 1997: Додати Proceedings of the Second Annual Conference / Ed. by JR

93. Koza, K. Deb, M. Dorigo, DB Fogel, M. Garzon, H. Iba, RL Riolo. - Stanford University, CA, USA: Morgan Kaufmann, 13-16 1997. - P. 380.

94. Zheng Z., Webb G. Multiple boosting: A combination of boosting and bagging.

95. Zheng Z., Webb G., Ting K. Integrating boosting and stochastic attribute selection committees for further improving the performance of decision tree learning.

96. Zheng Z., Webb GI Stochastic attribute selection committees with multiple boosting: Learning more accurate and more stable classifer committees // PacificAsia Conference on Knowledge Discovery and Data Mining. - 1999. - Pp. 123-132.

97. Zheng Z., Webb GI Multi strategy ensemble learning: Reducing error by combining ensemble learning techniques. - 2003.

Додаток А (текст програми)

Main.java:

```
public class Main extends Application {
    public static Scene mainView;

    @Override
    public void start(Stage primaryStage) throws
Exception{
        Parent root =
FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Sorokin genetic transform
to vector graphic");
        mainView = new Scene(root);
        primaryStage.setScene(mainView);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

sample.fxml:

```
<FlowPane prefHeight="448.0" prefWidth="820.0"
GridPane.rowIndex="9"
xmlns="http://javafx.com/javafx/8.0.121"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="org.max.Controller">
    <children>
        <GridPane prefHeight="414.0" prefWidth="820.0">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
            </columnConstraints>
            <rowConstraints>
                <RowConstraints minHeight="10.0"
prefHeight="30.0" vgrow="SOMETIMES" />
            </rowConstraints>
            <children>
                <ImageView fx:id="originImage"
fitHeight="408.0" fitWidth="408.0" pickOnBounds="true"
preserveRatio="true" />
            <ImageView fx:id="generatedimage"
fitHeight="408.0" fitWidth="408.0"
nodeOrientation="INHERIT" pickOnBounds="true"
preserveRatio="true" GridPane.columnIndex="1" />
            </children>
```

```
        </GridPane>
        <Button fx:id="loadImage" mnemonicParsing="false"
onAction="#loadImage" text="Load Image">
            <FlowPane.margin>
                <Insets left="55.0" right="40.0" />
            </FlowPane.margin>
        </Button>
        <Button fx:id="exportSvg" mnemonicParsing="false"
onAction="#exportSvg" prefHeight="25.0" prefWidth="104.0"
text="Export SVG">
            <FlowPane.margin>
                <Insets left="10.0" right="40.0" />
            </FlowPane.margin>
        </Button>
        <Button fx:id="saveData" mnemonicParsing="false"
onAction="#saveData" prefHeight="25.0" prefWidth="102.0"
text="Save Data">
            <FlowPane.margin>
                <Insets left="10.0" right="40.0" />
            </FlowPane.margin>
        </Button>
        <Button fx:id="loadData" mnemonicParsing="false"
onAction="#loadData" prefHeight="25.0" prefWidth="98.0"
text="Load Data">
            <FlowPane.margin>
                <Insets left="10.0" right="40.0" />
            </FlowPane.margin>
        </Button>
        <Button fx:id="start" mnemonicParsing="false"
onAction="#start" prefHeight="25.0" prefWidth="102.0"
text="Start">
            <FlowPane.margin>
                <Insets left="10.0" right="40.0" />
            </FlowPane.margin>
        </Button>
    </children>
</FlowPane>
Controller.java:
public class Controller {
   File selectedFile;
   Worker w;
   boolean dialog = true;

   @FXML
   private void loadImage(ActionEvent event) {
      System.out.println("LoadImage event: ");
      FileChooser fileChooser = new FileChooser();
      fileChooser.setTitle("Open Resource File");
      fileChooser.getExtensionFilters().addAll(
```

```
            new FileChooser.ExtensionFilter("Image Files",
"*.png", "*.jpg", "*.gif"));
      selectedFile = fileChooser.showOpenDialog(null);
      if (selectedFile != null) {
         try {
            BufferedImage bufferedImage =
ImageIO.read(selectedFile);
            Image image =
SwingFXUtils.toFXImage(bufferedImage, null);
            ((ImageView)
Main.mainView.lookup("#originImage")).setImage(image);
         } catch (IOException ignored) {
         }
      }
   }


   @FXML
   private void exportSvg(ActionEvent event) {
      if (!dialog) {
         FileChooser fileChooser = new FileChooser();
         fileChooser.setTitle("Save Image");
         fileChooser.getExtensionFilters().addAll(
               new FileChooser.ExtensionFilter("Save
data", "*.svg")
         );
         File file = fileChooser.showSaveDialog(null);
         try (BufferedWriter bw = new BufferedWriter(new
FileWriter(file))) {
            bw.write(w.toSvg());
         } catch (IOException e) {
            e.printStackTrace();
         }
      } else {
         Alert alert = new
Alert(Alert.AlertType.INFORMATION);
         alert.setTitle("Information Dialog");
         alert.setHeaderText(null);
         alert.setContentText("Save must be after start
process!");
         alert.showAndWait();
      }
   }

   @FXML
   private void saveData(ActionEvent event) {
      if (!dialog) {
         try {
            FileChooser fileChooser = new FileChooser();
```

```
            fileChooser.setTitle("Save Data");
            fileChooser.getExtensionFilters().addAll(
                  new FileChooser.ExtensionFilter("Save
data", "*.ser")
            );
            File file = fileChooser.showSaveDialog(null);
            this.serialize(w.getWorkarea(),
file.getPath());
         } catch (IOException e) {
            e.printStackTrace();
         }
      } else {
         Alert alert = new
Alert(Alert.AlertType.INFORMATION);
         alert.setTitle("Information Dialog");
         alert.setHeaderText(null);
         alert.setContentText("Save must be after start
process!");
         alert.showAndWait();
      }
   }

   @FXML
   private void loadData(ActionEvent event) {
      try {
         FileChooser fileChooser = new FileChooser();
         fileChooser.setTitle("Open Resource File");
         fileChooser.getExtensionFilters().addAll(
               new FileChooser.ExtensionFilter("Load saved
data", "*.ser"));
         Object wo =
deserialize(fileChooser.showOpenDialog(null).getPath());
         if (wo instanceof Workarea) {
            if (w == null) {
               w = new Worker();
            }
            w.setMainFrame(Main.mainView);
            w.setWorkarea((Workarea) wo);
         }

      } catch (IOException e) {
         e.printStackTrace();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }

   @FXML
   private void start(ActionEvent event) {
```

```
      if (selectedFile != null && selectedFile.canRead())
{
         Task<Boolean> task = new Task<Boolean>() {
            @Override
            public Boolean call() throws IOException {
               if (w == null) {
                  w = new Worker();
               }
               w.setMainFrame(Main.mainView);
               w.doAction(selectedFile);
               return true;
            }
         };
         dialog = false;
         new Thread(task).start();
      } else {
         Alert alert = new
Alert(Alert.AlertType.INFORMATION);
         alert.setTitle("Information Dialog");
         alert.setHeaderText(null);
         alert.setContentText("At first you need select
image!");
         alert.showAndWait();
      }

   }

   private Object deserialize(String fileName) throws
IOException,
         ClassNotFoundException {
      FileInputStream fis = new FileInputStream(fileName);
      ObjectInputStream ois = new ObjectInputStream(fis);
      Object obj = ois.readObject();
      ois.close();
      return obj;
   }

   private void serialize(Object obj, String fileName)
         throws IOException {
      FileOutputStream fos = new
FileOutputStream(fileName);
      ObjectOutputStream oos = new
ObjectOutputStream(fos);
      oos.writeObject(obj);
      fos.close();
   }
}
```

Workarea.java:

```
public class Workarea implements Cloneable, Serializable {
   public List<Polygon> Polygons;

   public boolean IsChange;

   public void SetRandom() {
      Polygons = new ArrayList<>();

      for (int i = 0; i < Helper.MinPolygons; i++) {
         AddPolygon();
      }

      IsChange = true;
   }

   public void Mutate() {
      if (Helper.rDouble(1.0) <=
Helper.MutationWorkareaAddPolygonChance) {
         AddPolygon();
         IsChange = true;
      }

      if (Helper.rDouble(1.0) <=
Helper.MutationWorkareaDelPolygonChance) {
         DelPolygon();
         IsChange = true;
      }

      Polygons.forEach(p -> p.Mutate(this));
   }

   private void AddPolygon() {
      if (Polygons.size() < Helper.MaxPolygons) {
         Polygon polygon = new Polygon();
         polygon.SetRandom();
         Polygons.add(polygon);
      }
   }

   private void DelPolygon() {
      if (Polygons.size() > Helper.MinPolygons) {
         int index = Helper.rInt(0, Polygons.size());
         Polygons.remove(index);
      }
   }

   public double Fitness(Color[][] colors) {
      double fitness = 0;
      BufferedImage img = Draw();
```

```
      for (int i = 0; i < Helper.Width; i++) {
         for (int j = 0; j < Helper.Height; j++) {

            int clr = img.getRGB(i, j);
            int red = (clr & 0x00ff0000) >> 16;
            int green = (clr & 0x0000ff00) >> 8;
            int blue = clr & 0x000000ff;

            Color c1 = new Color(red, green, blue);
            Color c2 = colors[i][j];

            int r = c1.getRed() - c2.getRed();
            int g = c1.getGreen() - c2.getGreen();
            int b = c1.getBlue() - c2.getBlue();

            fitness += r * r + g * g + b * b;
         }
      }

      return fitness;
   }

   public BufferedImage Draw() {
      BufferedImage img = new BufferedImage(Helper.Width,
Helper.Height, TYPE_INT_ARGB);

      Graphics g = img.createGraphics();
      for (Polygon p : Polygons) {
         p.Draw(g);
      }
      return img;
   }


   @Override
   public synchronized Object clone() {
      Workarea newarea = new Workarea();
      newarea.Polygons = new ArrayList<Polygon>();
      Polygons.forEach((p) ->
newarea.Polygons.add((Polygon) p.clone()));
      return newarea;
   }

   @Override
   public String toString() {
      return "Workarea{" +
            "Polygons=" + Polygons +
```

```
            ", IsChange=" + IsChange +
            '}';
    }

    public String toSvg() {
        String svg = "";
        String header = "<?xml version=\"1.0\"
encoding=\"iso-8859-1\"?>\n" +
            "<svg version=\"1.1\" id=\"Layer_1\"
xmlns=\"http://www.w3.org/2000/svg\"
xmlns:xlink=\"http://www.w3.org/1999/xlink\" height=\"" +
Draw().getHeight() + "\" width=\"" + Draw().getWidth() +
"\">";
        ArrayList<String> poly = new ArrayList<>();
        Polygons.forEach((Polygon p) ->
poly.add(p.toSvg()));
        String footer = "</svg>";
        svg += header;
        svg += String.join(" ", poly);
        svg += footer;
        return svg;
    }
}
Polygon.java:
public class Polygon implements Cloneable, Serializable {

    public List<Point> Points;
    public Brush Brush;

    public void SetRandom() {
        Points = new ArrayList<Point>();

        Point center = new Point();
        center.SetRandom();

        for (int i = 0; i < Helper.MinPointsPerPolygon; i++)
{
            Point point = new Point();
            point.X = Math.min(Math.max(0, center.X +
Helper.rInt(-3, 4)), Helper.Width);
            point.Y = Math.min(Math.max(0, center.X +
Helper.rInt(-3, 4)), Helper.Height);
            Points.add(point);
        }

        Brush = new Brush();
        Brush.SetRandom();
    }
```

```
   public void Mutate(Workarea workarea) {
      if (Helper.rDouble(1.0) <=
Helper.MutationPolygonAddPointChance) {
         AddPoint();
         workarea.IsChange = true;
      }

      if (Helper.rDouble(1.0) <=
Helper.MutationPolygonDelPointChance) {
         DelPoint();
         workarea.IsChange = true;
      }

      Brush.Mutate(workarea);
      Points.forEach((Point p) -> p.Mutate(workarea));
   }

   private void AddPoint() {
      if (Points.size() < Helper.MaxPointsPerPolygon) {
         Point point = new Point();
         int index = Helper.rInt(1, Points.size() - 1);
         Point p1 = Points.get(index - 1);
         Point p2 = Points.get(index);
         point.X = (p1.X + p2.X) / 2;
         point.Y = (p1.Y + p2.Y) / 2;
         Points.add(index, point);
      }
   }

   private void DelPoint() {
      if (Points.size() > Helper.MinPointsPerPolygon) {
         int index = Helper.rInt(0, Points.size());
         Points.remove(index);
      }
   }

   public void Draw(Graphics g) {
      int npoints = Points.size();
      int xpoints[] = new int[npoints];
      int ypoints[] = new int[npoints];

      for (int i = 0; i < npoints; i++) {
         xpoints[i] = Points.get(i).X;
         ypoints[i] = Points.get(i).Y;
      }

      java.awt.Polygon p = new java.awt.Polygon(xpoints,
ypoints, npoints);
```

```
      g.setColor(new Color(Brush.R, Brush.G, Brush.B,
Brush.A));
      //  g.drawPolygon(p);
      g.fillPolygon(p);
      //  g.dispose();

   }

   @Override
   protected Object clone() {
      Polygon newpolygon = new Polygon();
      newpolygon.Brush = (org.max.gen.Brush)
Brush.clone();
      newpolygon.Points = new ArrayList<>();
      for (Point p : Points) {
         newpolygon.Points.add((Point) p.clone());
      }
      // Points.forEach((Point p) ->
newpolygon.Points.add((Point) p.clone()));
      return newpolygon;
   }

   @Override
   public String toString() {
      return "Polygon{" +
            "Points=" + Points +
            ", Brush=" + Brush +
            '}';
   }

   public String toSvg() {
      ArrayList<String> points = new ArrayList<>();
      Points.forEach((Point p) -> points.add(p.X + "," +
p.Y));
      String arrayPoints = String.join(" ", points);
      String color = "style=\"fill:rgba(" + Brush.R + ","
+ Brush.G + "," + Brush.B + "," + ((double) Brush.A / 255)
+ ")\"";
      return " <polygon points=\"" + arrayPoints + "\" " +
color + " />";
   }
}
Point.java:
public class Point implements Cloneable, Serializable {
   public int X;
   public int Y;

   public Point(int x, int y) {
      X = x;
```

```
      Y = y;
   }

   public Point() {
   }

   public void SetRandom() {
      X = ThreadLocalRandom.current().nextInt(0,
Helper.Width);
      Y = ThreadLocalRandom.current().nextInt(0,
Helper.Height);
   }

   public void Mutate(Workarea workarea) {

      if (rDouble(1.0) <=
Helper.MutationPointMoveMaxChance) {
         SetRandom();
         workarea.IsChange = true;
      }

      if (rDouble(1.0) <=
Helper.MutationPointMoveMiddleChance) {
         X = Math.min(Math.max(0, X + Helper.rInt(-
Helper.MiddleRange, Helper.MiddleRange + 1)),
Helper.Width);
         Y = Math.min(Math.max(0, Y + Helper.rInt(-
Helper.MiddleRange, Helper.MiddleRange + 1)),
Helper.Height);
         workarea.IsChange = true;
      }

      if (rDouble(1.0) <=
Helper.MutationPointMoveNearChance) {
         X = Math.min(Math.max(0, X + Helper.rInt(-
Helper.NearRange, Helper.NearRange + 1)), Helper.Width);
         Y = Math.min(Math.max(0, Y + Helper.rInt(-
Helper.NearRange, Helper.NearRange + 1)), Helper.Height);
         workarea.IsChange = true;
      }
   }

   @Override
   protected Object clone() {
      return new Point(X, Y);
   }

   @Override
   public String toString() {
```

```
      return "Point{" +
            "X=" + X +
            ", Y=" + Y +
            '}';
   }
}
Helper.java :
public class Helper {
   public static int Width = 0;
   public static int Height = 0;

   public static int MinPointsPerPolygon = 3;
   public static int MaxPointsPerPolygon = 40;

   public static int MinPolygons = 0;
   public static int MaxPolygons = 100000;

   public static int NearRange = 3;
   public static int MiddleRange = 20;

   public static double MutationPointMoveMaxChance =
0.0007;
   public static double MutationPointMoveMiddleChance =
0.0007;
   public static double MutationPointMoveNearChance =
0.0007;

   public static double MutationBrushAChance = 0.0007;
   public static double MutationBrushRChance = 0.0007;
   public static double MutationBrushGChance = 0.0007;
   public static double MutationBrushBChance = 0.0007;

   public static double MutationPolygonAddPointChance =
0.0007;
   public static double MutationPolygonDelPointChance =
0.0007;

   public static double MutationWorkareaAddPolygonChance =
0.0014;
   public static double MutationWorkareaDelPolygonChance =
0.0007;

   private Helper() {
   }

   public static int rInt(int start, int end) {
      return ThreadLocalRandom.current().nextInt(start,
end);
   }
```

```
    public static double rDouble(double max) {
        return ThreadLocalRandom.current().nextDouble(max);
    }
}
Brash.java:
public class Brush implements Cloneable, Serializable {

    public int A;
    public int R;
    public int G;
    public int B;

    public Brush(int a, int r, int g, int b) {
        this.A = a;
        this.R = r;
        this.G = g;
        this.B = b;
    }

    public Brush() {
    }

    public void SetRandom() {
        A = rInt(30, 61); // Is need add +1 to bound?
        R = rInt(0, 256);
        G = rInt(0, 256);
        B = rInt(0, 256);
    }

    public void Mutate(Workarea workarea) {
        if (rDouble(1.0) <= Helper.MutationBrushAChance) {
            A = rInt(30, 61);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationBrushRChance) {
            R = rInt(0, 256);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationBrushGChance) {
            G = rInt(0, 256);
            workarea.IsChange = true;
        }

        if (rDouble(1.0) <= Helper.MutationBrushBChance) {
            B = rInt(0, 256);
            workarea.IsChange = true;
```

```
        }
    }

    @Override
    protected synchronized Object clone() {
        return new Brush(A, R, G, B);
    }

    @Override
    public String toString() {
        return "Brush{" +
                "A=" + A +
                ", R=" + R +
                ", G=" + G +
                ", B=" + B +
                '}';
    }
}
```

Більше можна знайти в доданому до роботи Диску.

Додаток Б

ВІДГУК
на дипломний проект студента групи 122м-16-1 Колотилина Д.Е.
на тему: «Дослідження оцінки ефективності нейронних мереж при обробці зображення»

Тема дипломного проекту - конвертація растрової графіки у векторну з використанням генетичнх алгоритмі.

Актуальність поставленого завдання обумовлюється широким попитом на різного роду інформаційних систем в різних сферах діяльності.

Тема дипломного проекту прямо пов'язана з об'єктом діяльності 122. Програмна інженерія магістр з інформаційних технологій.

Зміст роботи включає створення, дослідження, спостереження за поведінкою мутації і природного добору генів програмного оточення інформаційних систем за допомогою впровадження технології Genetic Programming. При вирішенні поставленого завдання використовувалися наукові досягнення в областях розробки інформаційних систем і програмного забезпечення. Наукова новизна отриманих результатів полягає в проведенні аналізу та виявленні недоліків поведінки геномів в ізольованому середовищі, схрещування, мутацією об'єктів оточення, а також у використанні методики алгоритмізації інформаційних систем на основі використання технології генетичної алгоритмізації.

Практична цінність полягає у векторизації растрової графіки з використанням технології генетичної алгоритмізації і природного відбору об'єктів, логирование всіх етапів відсіювання сильних популяцій. Це дає можливість застосувати нові технології використання генетичних алгоритмів, що дозволяє отримати якісні зображення за певний період часу.

Ступінь опрацювання компонентів даного проекту, дозволяє оцінити роботу на «задовільно» і рекомендувати присвоїти студенту Д.Е. Колотиліну кваліфікацію «Професіонал з розробки та тестування прогрманого забепечення».

Керівник дипломного проекту,
д.т.н., проф. кафедри ПЗКС /Алексєєв М.О./

Додаток В

Рецензія
на дипломний проект студента групи 122м-16-1 Колотилина Д.Е.
на тему: «Дослідження оцінки ефективності нейронних мереж при обробці зображення»

Як свідчать різні джерела інформації, в даний час в нашій країні та за кордоном за результатами оцінки внеску різних технічних засобів в побудову хмарних рішень для наукових досліджень, високонавантажених алгоритмів, які доступні кожному для своїх експериментів найкращими являються Google Compute Engine, Microsoft Azure.

В дипломному проекті в результаті проведеної роботи було розроблено фрагмент аналізу по апроксимації растрової графіки. Для досягнення основної мети виконано наступні завдання: аналіз спектрів RGB зображень; аналіз особливостей генетичних алгоритмів, шляхом природного відбору; аналіз методів розпізнавання графічної інформації; В результаті сформовано спроектовний та розроблений алгоритм для конвертації зображень з растрового формату на векторний. Під час проведення дипломної роботи була розробленна формула для фукції пристосування, основний механізм порівняння результатів роботи якої є алгоритм, що використовує методику поколінь які взаємозв'язані з попередніми поколіннями генів.

Розроблена система дозволить підвищити швидкість обробки інформації, оптимізувати місце зберігання, скоротить терміни завантаження зображень, позбавить від проблем масштабування jpeg і заощадить час роботи користувача користувачів.

Студент Колотилін Д. Е. розібрався в специфіці застосування різноманітних інформаційних технологій розпізнавання графічної інформації, розробкою, порівняння з використанням генетичних алгоритмів у різних сферах діяльності.

З огляду на вищевикладене, можна зробити висновок, що даний проект цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт рівня магістра.

В цілому проведенний аналіз використанних компонентів даного проекту, дозволяє оцінити роботу на «відмінно» і рекомендувати присвоїти студенту кваліфікацію «Професіонал з розробки та тестування прогрманого забепечення».

Рецензент

Додаток Г

Перелік документів на диску

ім'я файлу	опис
Пояснювальна записка	
Пояснювальна запіска.docx	Пояснювальна записка
програма	
project.zip	Архів. Містить вихідні коди програми.
презентація	
Презентація.pptx	Презентація дипломного проекту