

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

“ ___ ” червня 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напрямку підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: “Модифікований генетичний алгоритм визначення хроматичного числа графа”

Виконала: студентка IV курсу, групи КВ-53

Товстенко Ольга Вячеславівна

(підпис)

Керівник, доц. каф. СПіСКС, к.т.н. Марченко О. І.

(підпис)

Консультант з нормоконтролю, доц.каф.СПіСКС,к.т.н. Клятченко Я.М.

(підпис)

Рецензент доц. каф. ПЗКС, к.т.н. Заболотня Т.М.

(підпис)

Засвідчую, що у цьому дипломному
проекті немає запозичень з праць інших
авторів без відповідних посилань.

Студентка _____
(підпис)

Київ – 2019 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрямок підготовки 6.050102 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

«___» червня 2019 р.

ЗАВДАННЯ

**на дипломний проект студентки
Товстенко Ольги Вячеславівни**

1. Тема проекту “Модифікований генетичний алгоритм визначення хроматичного числа”,

керівник проекту: доц. каф. СПіСКС Марченко Олександр Іванович,

затверджені наказом по університету від «22» травня 2019. №1330-С

2. Термін подання студентом проекту _____

3. Вихідні дані проекту: див. технічне завдання.

4. Зміст пояснювальної записки:

- Аналіз існуючих рішень та обґрунтування теми дипломного проекту;
- Опис модифікованого генетичного алгоритму визначення хроматичного числа графа;
- Структура та реалізація модифікованого генетичного алгоритму визначення хроматичного числа графа;
- Порівняльний аналіз роботи генетичних операторів модифікованого генетичного алгоритму визначення хроматичного числа графа;

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

- Модифікований генетичний алгоритм визначення хроматичного числа. Діаграма класів.
- Модифікований генетичний алгоритм визначення хроматичного числа. Блок-схема алгоритму.
- Ініціалізація початкової популяції. Блок-схема алгоритму.
- Мутація з пошуком найгірше розміщеної вершини. Блок-схема алгоритму.
- Презентація на тему проекту

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.н.т.		

7. Дата видачі завдання _____.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту
1.	Вивчення літератури за тематикою проекту	10.11.2018
2.	Розроблення та узгодження технічного завдання	25.11.2018
3.	Аналіз існуючих рішень	07.02.2019
4.	Підготовка матеріалів текстової частини проекту	01.03.2019
5.	Підготовка графічної частини дипломного проекту	02.05.2019
6.	Оформлення документації дипломного проекту	20.05.2019
7.	Попередній огляд матеріалів диплому на кафедрі	29.05.2019

Студент _____
(підпис)

Товстенко О.В.

Керівник проекту _____
(підпис)

Марченко О.І.

* Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (73 сторінки, 15 рисунків., 21 псевдокод, 11 таблиць, 5 графіків, 4 додатки).

Метою даного дипломного проекту є створення модифікації генетичного алгоритму визначення хроматичного числа графа.

В роботі розглянуто та проаналізовано етапи роботи класичного генетичного алгоритму. Розібрано різні види селекції та схрещування, найбільш поширені методи кодування даних, існуючі умови завершення роботи алгоритму. З усіх можливих варіантів вибрано такі, що найбільш підходять до вирішення задачі знаходження хроматичного числа, вибір обґрунтовано.

В програмі передбачена можливість: задавати кількість вершин та ребер в графі; створювати випадкові графи; задавати кількість початкової популяції, кількість батьків, що мутують при кожній ітерації, кількість батьків для кросоверу; зміни штрафу за некоректний колір вершини та штрафу за кожен новий колір; вибирати функцію мутації та схрещування; вказати бажане хроматичне число та перевірити його на конфліктність. Розроблено такі тестові моделі, що найбільш підходять для демонстрації функцій алгоритму. Проаналізована швидкість та точність в залежності від параметрів. Підібрано оптимальний набір функцій мутації та кросоверу, їх послідовність та ознака початку дії кожної функції.

Для реалізації генетичного алгоритму було обрано мову програмування Python.

Даний алгоритм може бути використаний для складання розкладів, розподілу частот, реєстрів у мікропроцесорах, обчислення похідних, розпаралелювання обчислень за числовими методами.

Ключові слова: алгоритм, генетичний алгоритм, хроматичне число, граф, мутація, кросовер, фітнес-функція, Python.

ABSTRACT

The qualifying work includes an explanatory note (73 pages, 15 figures, 21 pseudocodes, 11 tables, 5 charts, 4 annexes).

The purpose of this diploma project is to create a modification of the genetic algorithm for determining the chromatic number of a graph.

The stages of the work of the classical genetic algorithm are considered and analyzed in this work. Different types of breeding and crossover are analyzed, the most common methods of encoding data, the existing conditions for the completion of the algorithm. Of all possible options, those that are most suited to the solution of the problem of finding a chromatic number are selected, the choice is justified.

The program provides the ability to: set the number of vertices and edges in the graph; create random graphs; set the number of initial population, the number of parents, mutate each iteration, the number of parents for the crossover; change the fine for the wrong color of the summit and fine for each new color; select mutation and crossing function; specify the desired chromatic number and check it for conflicts. The following test models are developed that are most suitable for demonstrating the functions of the algorithm. The speed and accuracy are analyzed, depending on the parameters. The optimal set of mutation and crossover functions, their sequence and a sign of the beginning of each function's operation are selected.

To implement the genetic algorithm was selected the Python programming language.

This algorithm can be used for scheduling, frequency distribution, registers in microprocessors, derivative calculations, parallelization of computations by numerical methods.

Key words: algorithm, genetic algorithm, chromatic number, graph, mutation, crossover, fitness function, Python.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.045420.002 ТЗ	Модифікований генетичний алгоритм визначення хроматичного числа графа Технічне завдання	4		
	A4	ІАЛЦ.045420.003 ТП	Модифікований генетичний алгоритм визначення хроматичного числа графа Відомість технічного проекту	2		
	A4	ІАЛЦ.045420.004 ПЗ	Модифікований генетичний алгоритм визначення хроматичного числа графа Пояснювальна записка	73		
	A4	ІАЛЦ.045420.005 Д1	Модифікований генетичний алгоритм визначення хроматичного числа графа. Діаграма класів.	1		

					ІАЛЦ.045420.001 ОА		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Товстенко О.В.			Літ.	Аркуш	Аркушів
Перевір.		Марченко О.І.				1	2
Н. контр.		Клятченко Я.М.			НТУУ "НТУУ «КПІ ім. Ігоря Сікорського», ФПМ КВ-53		
Затвер.		Тарасенко В.П.					
					Модифікований генетичний алгоритм визначення хроматичного числа графа. Опис альбому		

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.045420.006 Д2	Модифікований генетичний алгоритм визначення хроматичного числа графа. Блок-схема алгоритму.	1		
	A4	ІАЛЦ.045420.007 Д3	Ініціалізація початкової популяції. Блок-схема алгоритму.	1		
	A4	ІАЛЦ.045420.008 Д4	Мутація з пошуком найгірше розміщеної вершини. Блок-схема алгоритму.	1		
		Диск CD-ROM	Матеріали дипломного проекту.	1		
ІАЛЦ.045420.001 ОА						Арк. 2
Змін.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ	2
2. ПІДСТАВА ДЛЯ РОЗРОБКИ.....	2
3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ.....	2
4. ДЖЕРЕЛА РОЗРОБКИ.....	2
5. ТЕХНІЧНІ ВИМОГИ.....	3
5.1. Вимоги до програмного продукту, що розробляється	3
5.2. Вимоги до апаратного забезпечення	3
5.3. Вимоги до програмного забезпечення користувача.....	3
6. ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ. 045420.002 ТЗ			
Зм	Лист	№ докум.	Підп.	Дата	Модифікований генетичний алгоритм визначення хроматичного числа графа Технічне завдання	Лім.	Лист	Листів
Розроб.		Товстенко О.В.				1	4	
Перев.		Марченко О.І.						
Н. контр.		Клятченко Я.М.						
Затв.		Тарасенко В.П.						
						НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-53		

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Модифікований генетичний алгоритм визначення хроматичного числа графа».

Галузь застосування: дослідження хроматичного числа в залежності від графа, створення розкладів, задачі на розподілення (регістрів, частот, тощо), вирішення проблем пов'язаних з плануванням часу, Судоку.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є розроблення модифікації генетичного алгоритму на мові програмування Python.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

					ІАЛЦ.045420.002 ТЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		2

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з будь-якою операційною системою (Windows, Linux, MacOS);
- можливість задавати кількість вершин у графі;
- можливість задавати кількість ребер у графі;
- можливість створити випадковий граф;
- можливість вибрати одну або декілька функцій кроссоверу та мутації;
- можливість задання кількості початкової популяції та інших параметрів;
- виведення на екран статистики після роботи алгоритму (кількість ітерацій, час роботи, найкраща хромосома, тощо).

5.2. Вимоги до апаратного забезпечення

- Процесор: будь-який 2-ядерний з тактовою частотою 1 Гц та більше;
- Оперативна пам'ять: від 1 Гб для 32-бітних та від 2 Гб для 64-бітних систем, в залежності від величини графу.

5.3. Вимоги до програмного забезпечення користувача

- Операційна система Windows, Linux, MacOS.

					ІАЛІЦ.045420.002 ТЗ	<i>Лист</i>
<i>Зм</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Підп.</i>	<i>Дата</i>		3

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	10.11.2018
2.	Розроблення та узгодження технічного завдання	25.11.2018
3.	Аналіз існуючих рішень	07.02.2019
4.	Підготовка матеріалів текстової частини проекту	01.03.2019
5.	Підготовка графічної частини дипломного проекту	02.05.2019
6.	Оформлення документації дипломного проекту	20.05.2019
7.	Попередній огляд матеріалів диплому на кафедрі	29.05.2019

					ІАЛЦ.045420.002 ТЗ	<i>Лист</i>
<i>Зм</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Підп.</i>	<i>Дата</i>		4

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.045420.004 ПЗ	Модифікований генетичний алгоритм визначення хроматичного числа графа Пояснювальна записка	73		
	A4	ІАЛЦ.045420.005 Д1	Модифікований генетичний алгоритм визначення хроматичного числа графа Діаграма класів.	1		
	A4	ІАЛЦ.045420.006 Д2	Модифікований генетичний алгоритм визначення хроматичного числа графа Блок-схема алгоритму.	1		
	A4	ІАЛЦ.045420.007 Д3	Ініціалізація початкової популяції. Блок-схема алгоритму.	1		
	A4	ІАЛЦ.045420.008 Д4	Мутація з пошуком найгірше розміщеної вершини. Блок-схема алгоритму.	1		

					ІАЛЦ.045420.003 ТП		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Товстенко О.В.			Літ.	Аркуш	Аркушів
Перевір.		Марченко О.І.				1	2
Н. контр.		Клятченко Я.М.			НТУУ "НТУУ «КПІ ім. Ігоря Сікорського», ФПМ КВ-53		
Затвер.		Тарасенко В.П.					
					<i>Відомість технічного проекту</i>		

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
		Диск CD-ROM	Матеріали дипломного проекту.	1		
Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛІЦ.045420.003 ТП	
					Арк.	2

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ_____	4
ВСТУП_____	5
1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ_____	6
1.1. Види задач з теорії графів та алгоритми знаходження хроматичного числа графа_____	6
1.2. Аналіз існуючих модифікацій генетичного алгоритму для вирішення задачі знаходження хроматичного числа графа_____	8
1.3. Задачі, що можуть бути вирішені за допомогою модифікованого генетичного алгоритму визначення хроматичного числа графа_____	12
1.4. Обґрунтування теми дипломного проекту_____	13
1.5. Обґрунтування вибору середовища розробки_____	14
2. ОПИС МОДИФІКОВАНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ВИЗНАЧЕННЯ ХРОМАТИЧНОГО ЧИСЛА ГРАФА_____	15
2.1. Граф як вхідні дані модифікованого генетичного алгоритму визначення хроматичного числа_____	15
2.2. Хроматичне число як ціль роботи модифікованого генетичного алгоритму визначення хроматичного числа графа_____	17
2.3. Базові поняття модифікованого генетичного алгоритму визначення хроматичного числа графа_____	19
2.4. Кодування хромосом в генетичному алгоритмі визначення хроматичного числа графа_____	20
2.5. Схема роботи модифікованого генетичного алгоритму визначення хроматичного числа графа_____	21

					ІАЛЦ.045420.004 ПЗ							
Зм.	Лист	№ докум.	Підп.	Дата	Модифікований генетичний алгоритм визначення хроматичного числа графа Пояснювальна записка			Літ.	Лист	Листів		
Розроб.		Товстенко О.В.									1	73
Перев.		Марченко О.І.										
Н.контр.		Клятенко Я.М.										
Затв.		Тарасенко В.П.										
					НТУУ "НТУУ «КПІ ім. Ігоря Сікорського», ФПМ КВ-53							

2.5.1. Величина початкової популяції_____	23
2.5.2. Хроматичне число початкової популяції_____	24
2.5.3. Умова завершення роботи алгоритму_____	25
2.5.4. Фітнес функція_____	26
2.5.5. Селекція хромосом_____	28
2.5.6. Функція кросоверу_____	30
2.5.7. Функція мутації_____	33
2.5.8. Створення нової популяції_____	37
3. СТРУКТУРА ТА РЕАЛІЗАЦІЯ МОДИФІКОВАНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ВИЗНАЧЕННЯ ХРОМАТИЧНОГО ЧИСЛА ГРАФА_____	41
3.1. Призначення пакетів та файлів програми_____	41
3.2. Вихідні параметри МГА визначення ХЧ_____	42
3.3. Опис параметрів МГА визначення ХЧ_____	43
3.4. Реалізація графа_____	45
3.5. Вибір ФК, ФМ, створення пулу випадкових хромосом, елітарна селекція, визначення РФФ найкращої хромосоми в популяції_____	46
3.6. Функції кросоверу, мутації, оцінки_____	48
3.7. Основний цикл роботи МГА визначення ХЧ_____	54
3.8. Створення випадкового графа_____	59
4. ПОРІВНЯЛЬНИЙ АНАЛІЗ РОБОТИ ГЕНЕТИЧНИХ ОПЕРАТОРІВ МОДИФІКОВАНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ВИЗНАЧЕННЯ ХРОМАТИЧНОГО ЧИСЛА ГРАФА_____	61
ВИСНОВКИ_____	69

ДОДАТКИ**Додаток 1. Копії графічних матеріалів**

- ІАЛЦ.045420.005 Д1. Модифікований генетичний алгоритм визначення хроматичного числа. Діаграма класів.
- ІАЛЦ.045420.006 Д2. Модифікований генетичний алгоритм визначення хроматичного числа. Блок-схема алгоритму.
- ІАЛЦ.045420.007 Д3. Ініціалізація початкової популяції. Блок-схема алгоритму.
- ІАЛЦ.045420.008 Д4. Мутація з пошуком найгірше розміщеної вершини. Блок-схема алгоритму.

Додаток 2. Презентація

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		3

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ГА – генетичний алгоритм

МГА – модифікований генетичний алгоритм, що є предметом даного дипломного проекту

ХЧ – хроматичне число

ВГ – випадковий граф

ФФ – фітнес функція

ФМ – функція мутації

ФК – функція кросоверу

ГО – генетичні оператори

РФФ – рахунок фітнес функції

ПП – початкова популяція

DIMACS – The Center for Discrete Mathematics and Theoretical Computer Science

ЕСJ – A Java-based Evolutionary Computation Research System

ТГ – теорія графів

ОК – оператор кросинговеру

ОМ – оператор мутації

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		4

ВСТУП

З розвитком інформаційних технологій та збільшенням обчислювальних потужностей задачі з теорії графів стають все більш використовуваними, а алгоритми знаходження приблизних рішень NP-повних задач все більш популярними. Однією з актуальних проблем є визначення хроматичного числа випадкового графа будь-якими можливими засобами, як наприклад, генетичний алгоритм, що емулює процес еволюції та застосовується до завдань, рішення яких дуже важко перевірити.

Складання розкладів в наукових закладах, раціональне планування часу, розподілення регістрів процесора, кодування інформації, розміщення людей в обмеженому просторі, навіть рішення Судоку є потенційними галузями застосування модифікованого генетичного алгоритму визначення хроматичного числа графа, що розглядається в даному дипломному проекті.

Хоча тематика беззаперечно актуальна, робіт за темою зовсім невелика кількість і всі вони є публікаціями в іноземних наукових виданнях чи міжнародних конференціях, тобто не мають відкритого коду, що доступний для звичайного користувача, а переважна більшість використовують невеликі графи, хроматичне число до яких вже відоме.

Цілю даного дипломного проекту є створення такої модифікації генетичного алгоритму, що може працювати з випадковими графами, остаточне хроматичне число яких невідоме. Забезпечення наявності україномовної інформації, щодо можливих рішень задачі, перевірка їх працездатності.

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		5

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1. Види задач з теорії графів та алгоритми знаходження хроматичного числа графа

Граф – це математичний об’єкт представлений множиною вершин та ребер графа. Вершини також називають вузлами, а ребра – дугами.

Теорія графів – це розділ дискретної математики, що вивчає властивості графів.

Визначають такі задачі з ТГ:

- Розфарбовування вершин графа [1]
- Розфарбовування ребер графа [1]
- Розфарбовування планарного графа (теорема про чотири кольори) [2]
- Знаходження хроматичного числа [1]
- Задача комівояжера (англ. Travelling Salesman Problem або скорочено TSP) [3]
- Задача розпізнавання кліки заданого розміру [4]
- Задача знаходження максимальної кліки [4]
- Задача про хід коня [5]
- Задача про сім мостів Кенігсберга [6]
- Пошук ізоморфного графа [7]
- Пошук максимального спільного графа [7]
- Пошук мінімального кістякового дерева у графі [8]

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		6

- Задача знаходження мостів графа [9]
- Задача пакування рюкзака [10]

Теорія випадкових графів вивчає випадкові графи.

Випадкові графи – це графи утворені з n ізольованих вершин з додавання k ребер, які пов’язані з випадковими вершинами.

В статті «Моделі випадкових графів та їх застосування» А.М. Райгородського [11] ВГ класифікуються таким чином:

- Модель Ердеша-Рені
- Модель Барабаши-Альберт
- Модель Боллобаша-Ріодана

В цій роботі зроблено вибір в користь задачі знаходження хроматичного числа для випадкових графів.

Розфарбовування вершин графа – призначення кожній вершині кольору. При цьому на практиці, окрім випадків графічного зображення графа, замість кольорів використовують цілі числа.

Хроматичне число графа – кількість кольорів, в які можна розфарбувати вершини графа таким чином, щоб кінці будь-якого ребра мали різні кольори.

Існують такі види алгоритмів для задачі знаходження ХЧ графу методом розфарбовування вершин графа:

- Повний перебір (англ. brute force)
- Жадібний алгоритм (англ. greedy algorithm)
- Бджолиний алгоритм (англ. artificial bee colony optimization або скорочено ABC)

- Мурашиний алгоритм (алгоритм оптимізації мурашиної колонії, англ. ant colony optimization або скорочено ACO)
- Метод рою часток (англ. Particle Swarm Optimization або скорочено PSO)
- Пошук в глибину
- За допомогою динамічного програмування
- Алгоритм видалення та стягування
- Метод гілок і меж (англ. Branch-and-Bound)
- Гармонійний пошук (англ. Harmony search)
- Алгоритм імітації відпалу (англ. Simulated annealing)
- Пошук з заборонами (англ. Tabu search)

В дипломній роботі зроблено вибір на користь генетичного алгоритму для вирішення поставленої задачі.

1.2. Аналіз існуючих модифікацій генетичного алгоритму для вирішення задачі знаходження хроматичного числа графа

Пошук статей та публікацій за темою виконувався за допомогою пошукової системи google.com та сайту ResearchGate. Наукових робіт українською виявлено не було.

З визначеної теми було знайдено такі публікації:

- “Genetic Algorithm Applied to the Graph Coloring Problem” by Musa M. Hindi and Roman Yampolskiy (University of Louisville) [12]

В публікації розглядається та аналізується варіант ГА для задачі розфарбовування графа. Подається псевдокод основних функцій,

відкритого коду немає. В якості вхідних даних алгоритм використовує графи з колекції DIMACS, що є центром дискретної математики та теоретичної інформатики (The Center for Discrete Mathematics and Theoretical Computer Science) [13].

- “Solving the Graph Coloring Problem using Genetic Programming” by Justine W. Shen (Stanford University) [14]

У даній роботі розглядається використання такої технології як ECJ, еволюційної системи обчислень реалізованої на мові Java (англ. A Java-based Evolutionary Computation Research System). [15] Зважаються її недоліки та переваги, розглядається можливість реалізувати повноцінний ГА для заданої цілі. ECJ це програмний каркас (фреймворк), що дозволяє працювати з генетичними алгоритмами, генетичним програмуванням, еволюційними стратегіями, паралельною еволюцією, оптимізацією великого числа частинок і диференціальною еволюцією. За допомогою фреймворку можна скористатися окремо реалізованими функціями селекції, кросоверу, мутації, а також фітнес-функцією. Подається псевдокод можливого варіанту ГА, відкритого коду немає.

- “Genetic Algorithm for Graph Coloring: Exploration of Galinier and Hao’s Algorithm” by Celia A. Glass (Cass Business School, London) and Adam Prugel-Bennett (University of Southampton, Southampton) [16]

Дана стаття розглядає, аналізує, модифікує і тестує більш ранню публікацію “Hybrid Evolutionary Algorithms for Graph Coloring” by Philippe Galinier and Jin-Kao Hao [17]. У ній доводиться ефективність роботи алгоритму і без елементів пошуку з заборонами. В якості вхідної інформації розглядаються ВГ, що є такими ж як і у оригінальному дослідженні. Це графи, де задана кількість вершин, а ребра між

кожними двома вершинами утворюються з ймовірністю $\frac{1}{2}$. Також тестування проводилося на графах з колекцій David Johnson, Joe Culberson, Craig Morgenstern [18]. Ні псевдокоду, ні відкритого коду немає.

- “A Method for the Minimum Coloring Problem Using Genetic Algorithms” by Haidar Harmanani and Hani Abas (Lebanese American University) [19]

В статті представлений та описаний класичний генетичний алгоритм для задачі визначення хроматичного числа. При чому, якщо у двох перших роботах ХЧ було відоме і використовувалося як верхня межа для кількості кольорів, які можна використовувати, то у цій публікації ХЧ саме знаходиться, а задача не зводиться до РГ заданою кількістю кольорів. Хоча автори також використовують графи з колекції DIMACS, для яких ХЧ вже відоме. Також визначною рисою є те, що функції мутації та кросоверу застосовуються до різних нащадків. У статті наведений загальний псевдокод ГА, відкритого коду немає.

Також було знайдено інші варіації ГА для вирішення даної задачі. Наприклад, паралельний генетичний алгоритм, стаття “Parallel Genetic Algorithm for Graph Coloring Problem” by Zbigniew Kokosinski, Marcin Kolodziej and Krzysztof Kwarciany (Cracow University of Technology) [20] та модифікація паралельного генетичного алгоритму на CUDA, стаття “A Fast Parallel Genetic Algorithm for Graph Coloring Problem Based on CUDA” by Buhua Chen, Bo Chen, Hongwei Liu and Xuefeng Zhang [21].

Більш детальний огляд вмісту статей розглянуто у другому розділі дипломної роботи.

Так як усі розглянуті вище дослідження не мають відкритого коду, викладеного в інтернет, а в найкращому випадку містять псевдокод

основних ідей, аналіз рішень продовжився серед наявних у відкритому доступі реалізацій. Їх пошук здійснювався за допомогою пошукової системи google.com та сайту GitHub.

Було знайдено три роботи із зовсім невеликими звітами:

- “Heuristic Algorithms for NP-complete Problems” by Juan Jose Martin Cara [22]

В роботі розглянуто три види алгоритмів: генетичний алгоритм, алгоритм імітації відпалу та пошук із заборонами. Вхідними даними є графи Michael Trick [18]. Відображено графік залежності кількості кольорів від кількості ітерацій. Критерієм зупинки алгоритму є 10 ітерацій з однаковою найкращою хромосомою. Як теоретична основа бралось дослідження [12]. Програма написана мовою C++.

- “Graph Coloring” by Amir Deljouyi [23]

В цій роботі також розглянуто три алгоритми: Вельш-Пауелла, гармонійного пошуку та генетичного пошуку. Вхідними даними є граф Петерсена [24], граф з кенігсбергськими мостами [6], повні графи (це графи, де кожні дві вершини пов’язані між собою спільним ребром). Головною особливістю програми є графічне зображення графів та можливість видаляти, коригувати та додавати окремі вершини та ребра. Програма працює з невеликими графами. Її можливо використовувати тільки на операційній системі Windows. Програма написана мовою TypeScript.

- “Using Genetic Algorithms To Solve The Graph Coloring Problem” by Ahmed M. Zarie (The American University in Cairo) [25]

У цій роботі на відміну від попередніх розглянуто лише один ГА. Використовується одна ФК та одна мутації. Кросовер з одною точкою

поділу, у ФМ одна з вершин з неправильним кольором розфарбовується будь-яким іншим кольором. Вказані параметри алгоритму, наприклад початкова популяція – це 200 хромосом, тощо. В алгоритмі вказана верхня межа (100%) та нижня межі (50%) ймовірності мутації дітей. Практично це означає, що кожен з двох нових нащадків мутує з ймовірністю 50%. Максимальна кількість ітерацій 5000 і це є умовою зупинки алгоритму. Програма написана мовою C++.

1.3. Задачі, що можуть бути вирішені за допомогою модифікованого генетичного алгоритму визначення хроматичного числа графа

Задача знаходження ХЧ та РГ є важливою в сучасному світі і застосовується у всіх сферах життя, як то:

- складання розкладів. Нехай потрібно прочитати кілька лекцій, кожна з яких займає одну годину, але деякі лекції не можуть відбуватися одночасно (наприклад, їх читає один і той самий лектор). Тоді, лекції будуть вершинами, що суміжні тільки тоді, коли не можуть відбуватися одночасно, а правильне розфарбування усуне несумісності в розкладі [27];
- вирішення проблем пов'язаних з плануванням часу чи розподілення регістрів процесора;
- цифрові водяні знаки [28];
- задача розподілу устаткування [27]. Дано дві множини: види робіт та обладнання, що потрібне для виконання цих робіт. При чому деякі механізми не можна використовувати одночасно на кількох видах робіт;
- задача з розміщенням в'язнів. У в'язниці є тисяча людей, деякі з них настільки ненавидять один одного, що їх небезпечно саджати в одну

камеру. Питання, яка мінімальна кількість камер потрібна? [26]

- правильна відповідь у звичайній грі Судоку може бути швидко знайдена за допомогою РГ з 81 вершини 9 кольорами.

1.4. Обґрунтування теми дипломного проекту

Розглянувши всі актуальні задачі з ТГ та методи їх рішення темою дипломного проекту було вибрано створення модифікації генетичного алгоритму для визначення хроматичного числа графа мовою Python. Проблема визначення кількості кольорів для РГ була обрана через свою актуальність та невелику кількість об'ємних робіт за темою, особливо через відсутність україномовних публікацій. Як спосіб вирішення задачі був обраний ГА завдяки своїй гнучкості та можливості налаштування під кожен окрему проблематику.

Переваги алгоритму:

- Наявність кількох функцій кросоверу та кількох функцій мутації, можливість порівнювати результати роботи в залежності від їх вибору;
- Як вхідні данні використовуються ВГ, користувач може задати кількість вершин та ребер графу;
- Можливість визначення алгоритмом приблизного ХЧ, не знаючи його наперед (тобто для випадкових графів);
- Можливість вказання бажаного ХЧ та перевірки на конфліктність (чи підходить ця кількість кольорів для коректного розфарбовування та на скільки підходить);
- Легка зміна параметрів, таких як кількість ПП, кількість батьків, що мутують при кожній ітерації, кількість батьків для кросоверу, зміна

штрафу за некоректний колір вершини та штрафу за кожен новий колір, тощо;

- Можливість застосування алгоритму до великих графів;
- Можливість порівняти результати одного й того ж ВГ при різних вхідних параметрах.

Дана система може бути використана як самостійний інструмент або як основа для вирішення задач, де потрібно знаходити хроматичне число і/або правильно розфарбувати граф. Наприклад, складання розкладів, вирішення проблем пов'язаних з плануванням часу, розподілення регістрів процесора, Судоку, тощо.

1.5. Обґрунтування вибору середовища розробки

Для реалізації ГА, що визначає ХЧ графа з вищевказаними особливостями була обрана мова програмування Python. В першу чергу через свою популярність у сферах data science та автоматизації даних, де з найбільшою ймовірністю може знадобитися обрана задача. Python - мова з відносно простим та інтуїтивно зрозумілим синтаксисом, що є перевагою в ситуаціях, коли треба змінити параметри алгоритму або якусь із функцій кросоверу, мутації, тощо, а ГА - це гнучкий інструмент, що мусить підлаштовуватися під кожен конкретну проблематику.

Компактність програмного коду та швидкість його розробки є суттєвим плюсом для алгоритмічних задач. В свою чергу порівняно низька швидкодія є недоліком. Кросплатформність значно розширює сферу застосування алгоритму. З розглянутих вище трьох реалізацій генетичного алгоритму ні одна не написана на Python (два на C++ і одна на TypeScript), що також є причиною вибору цієї мови.

2. ОПИС МОДИФІКОВАНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ВИЗНАЧЕННЯ ХРОМАТИЧНОГО ЧИСЛА ГРАФА

2.1. Граф як вхідні дані модифікованого генетичного алгоритму визначення хроматичного числа

Вхідними даними ГА визначення ХЧ є графи. Вони можуть бути двох видів:

- випадковий граф;
- граф формату DIMACS [13].

При цьому мається на увазі неорієнтовані графи без петель.

Неорієнтовані графи – це графи з двонаправленими ребрами.

Петля – це ребро, в якого і початок і кінець в одній і тій самій вершині.

В переважній більшості використовують другий вид. Наприклад, у розглянутих в першому розділі [12], [14], а також у частині тестувань в публікації [16].

Випадкові графи використовують у [23] (генерується ВГ представлений графічно, а користувач може додавати, видаляти вершини та ребра), [16] (це графи, де задана кількість вершин, а ребра між кожними двома вершинами утворюються з ймовірністю $\frac{1}{2}$) та [19].

Граф формату DIMACS (The Center for Discrete Mathematics and Theoretical Computer Science) кодується у файл формату .col. Рядки, що починаються з літери «с» є коментарями. Рядок з початком типу «p edge 9 13» означає, що граф складається з 9 вузлів та 13 ребер ('p' – це «point», тобто вершина, а 'edge' перекладається як ребро), рисунок 2.1. Далі розташовані рядки, що починаються з символу 'e', тобто «edge», наприклад, 'e 1 2'. Дві цифри після – це номери вершин, яких з'єднає дане ребро.

Переваги:

- дозволяє порівняти результати роботи ГА з іншими алгоритмами чи модифікаціями ГА, що використовували такі самі графи;
- дозволяє вибрати лише ті графи, що створені спеціально для вирішення конкретної проблеми (наприклад, знаходження кістяку графа або максимальної кліки);

Недоліки:

- Використовування графів з DIMACS колекції є також і недоліком, адже знаходження ХЧ будь-якого іншого графа алгоритмом не передбачено;
- В більшості алгоритмів вже вказано остаточне ХЧ (воно визначене для кожного графа з DIMACS колекції). А це означає, що алгоритм у ПП вже розфарбовує вершини заданою кількістю кольорів, тобто вирішує задачу не знаходження ХЧ, а саме розфарбовування вершин графа правильними кольорами;

Треба зазначити, що ХЧ вказане DIMACS вважається верхньою межею, тобто теоретично результат можна покращити, однак це малоімовірно зважаючи на те, що мінімальна кількість кольорів визначалася як найкращий результат з усіх алгоритмів, що використовували визначений граф як вхідний параметр.

Випадковий граф – це граф утворений з ізольованих вершин та ребер, що пов'язують дві випадкові вершини. Його можна описати двома способами:

- визначити усі вузли графа, а потім послідовно додавати ребра, випадково обираючи вершини на його кінцях;
- визначити всі вузли графа, а потім кожні дві вершини зв'язати

ребром з заданою ймовірністю (наприклад, $\frac{1}{2}$ [16]).

```
s number nodes, edges: 9, 13
p edge 9 13
e 1 2
e 1 4
e 1 5
e 2 4
e 2 5
e 3 5
e 3 6
e 4 5
e 5 6
e 5 8
e 6 9
e 7 8
e 8 9
```

Рисунок 2.1 – Приклад подання графа у форматі DIMACS

Переваги:

- працює з усіма видами графів;
- ХЧ не відоме і може бути будь-яким.

Недоліки:

- не дозволяє порівняти результати роботи ГА з іншими алгоритмами чи модифікаціями ГА, адже кожен раз граф різний.

МГА визначення ХЧ, що розглядається в даній дипломній роботі використовує ВГ як вхідну інформацію. Структурою програми зберігається можливість змінити ВГ на графи формату DIMACS.

2.2. Хроматичне число як ціль роботи модифікованого генетичного алгоритму визначення хроматичного числа графа

Хроматичне число графа – це величина, що дорівнює кількості кольорів, в які можна пофарбувати всі вершини графа так, щоб кінці будь-якого ребра мали різні кольори [26].

Розфарбовування вершин графа – це призначення кожній вершині кольору або цілого числа, що ідентифікує якийсь колір.

В роботі [14] доводиться, що якщо дати алгоритму свободу розфарбовувати вершини у стільки кольорів, скільки йому заманеться (тобто не обмежити його якоюсь визначеною верхньою межею), то рішення буде неоптимальним.

В більшості алгоритмів, що використовують графи з DIMACS колекції ХЧ вказане в DIMACS вважається верхньою межею (див. розділ 2.1). В роботі [19] ХЧ обмежується кількістю вершин у графі, кожна фарбується в унікальний колір. А потім за допомогою ФМ це число поступово зменшується.

В МГА визначення ХЧ використовуються ВГ і було вирішено базуватися на теоремі розглянутій нижче, аби звузити пошук ХЧ.

Теорема. Для всякого графа G виконується нерівність $X_p(G) \leq r + 1$. Де $X_p(G)$ – хроматичне число графа, а r – максимальний степінь вершини графа G .

Степінем вершини називається кількість ребер, що суміжні до неї. Максимальним степенем графа є найбільший зі степенів вершин графа.

Доведення теореми можна знайти в книжці «Основи дискретної математики» Капітонова, Кривий, Летичевський, Луцький, Печурін на сторінці 261 [27].

Для подальшого зменшення ХЧ використовується бінарний пошук. Як початкова права (верхня) межа використовується знайдена степінь графа,

як ліва (нижня) межа нуль. МГА визначення ХЧ працює з поточною кількістю кольорів (заданою бінарним пошуком), якщо при розфарбовуванні виникають конфлікти ліва межа зсувається вправо, якщо конфліктів не виявлено права межа зсувається вліво, і МГА знову намагається пофарбувати граф вже новою кількістю кольорів. Так відбувається до тих пір, поки права межа не стане менше лівої. Останній знайдений результат з нульовим конфліктом вважається остаточною відповіддю.

2.3. Базові поняття модифікованого генетичного алгоритму визначення хроматичного числа графа

Генетичний алгоритм (англ. genetic algorithm) – це евристичний алгоритм пошуку, що використовується для вирішення задач оптимізації та моделювання шляхом послідовного підбору, комбінування та варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію. Є різновидом еволюційних обчислень (англ. evolutionary computation). Особливістю ГА є акцент на операторі схрещування, який здійснює операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещування в живій природі. Батьком ГА вважається Джон Холланд (англ. John Holland) завдяки своїй класичній в цій області книжці «Adaptation in Natural and Artificial Systems». [29]

Деякі терміни:

Популяція – це кінцева множина особин.

Особини, що входять в популяцію, в ГА є хромосомами з закодованою в них множиною параметрів задачі, тобто рішень, які інакше називаються точками в просторі пошуку (search points). В деяких роботах особи називаються організмами.

Хромосоми – це впорядковані послідовності генів.

Ген – це атомарний елемент генотипу, зокрема, хромосоми.

Генотип – це набір хромосом даної особи. Отже, особами в популяції можуть бути генотипи або одиничні хромосоми. В МГА визначення ХЧ графа особами популяції є саме одиничні хромосоми, кожна з яких окреме вирішення задачі РГ. А геном у такій хромосомі є вершина графа пофарбована в заданий колір.

Аллель – це значення конкретного гена. В МГА визначення ХЧ графа алелями є кольори вершин графа.

Локус – це позиція, що вказує на місце розміщення конкретного гена в хромосомі. В МГА визначення ХЧ графа локусами є індекси списку, що також є номерами вершин графа.

Локи – це множина позицій генів.

2.4. Кодування хромосом в генетичному алгоритмі визначення хроматичного числа графа

В класичному ГА найчастіше використовуються такі методи кодування даних як двійкове, код Грея (найкращий варіант для мутацій) та логарифмічне кодування (для зменшення довжини хромосоми), але особливістю ГА його гнучкість та величезна кількість модифікацій для вирішення окремих задач.

Для задачі знаходження використовується таке кодування даних як на рисунку 2.2.

Хромосома – це список довжиною в N елементів, де N – це кількість вершин графа. Одна хромосома кодує одне ймовірне РГ, де індексами списку є номери вершин графа, а їх значеннями кольори цих вершин. При

чому кольори – це множина цілих чисел, де кожному кольору відповідає своя цифра. Виняток становить лише ситуація, коли граф зображується графічно, як на рисунку 2.2. Тоді створюється словник відповідності колір – число і при виведення на екран вершини графа дійсно позначені кольорами.

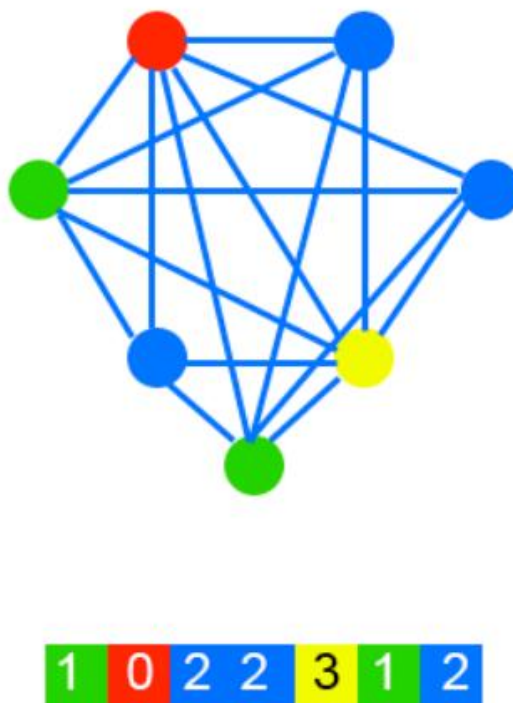


Рисунок 2.2 – Кодування даних в МГА визначення ХЧ графа. Взято з [12].

2.5. Схема роботи модифікованого генетичного алгоритму визначення хроматичного числа графа

Кожен ГА працює за схемою зображеною на рисунку 2.3. Спочатку відбувається ініціалізація ПП, її величина є параметром алгоритму. Це означає, що створюється задана кількість хромосом, кожна з яких є можливим розв'язком задачі.

Далі кожна з хромосом популяції оцінюється за заданими критеріями. І

алгоритм перевіряє чи виконалася умова його завершення. Якщо умова виконалася, з популяції вибирається хромосома з найкращим рахунком ФФ, вона і є вихідною інформацією, розв'язком задачі.

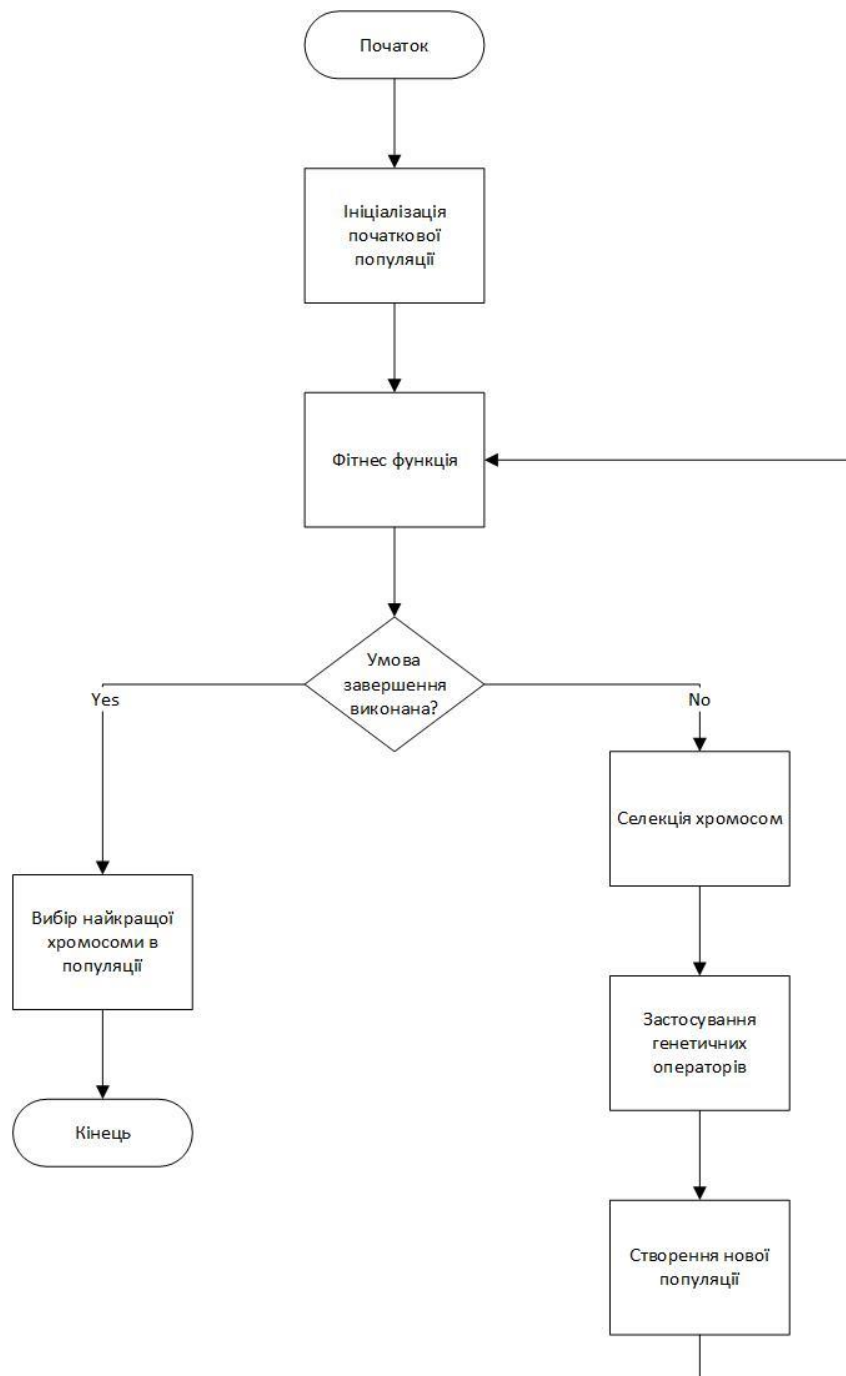


Рисунок 2.3 – Класична блок-схема генетичного алгоритму

Якщо ж умова завершення ще не досягнута, відбувається селекція хромосом з популяції. Тобто з популяції відбираються такі хромосоми, що задовольняють задані умови селекції. Над ними застосовуються ГО – кросовер та мутація. Після цього утворюється нова популяція зі змінених хромосом, що знову оцінюється ФФ. І так відбувається до тих пір, поки не буде виконано умову зупинки алгоритму.

2.5.1. Величина початкової популяції

Розмір ПП може бути різним. Він може змінюватися в залежності від розміру графа, як наприклад, в статті [16], автори змінюються число популяції від ста до п'ятисот особин в залежності від розміру графа.

В МГА визначення ХЧ використовується саме такий метод, однак найменша кількість популяції не сто особин, а 12, для малого графа в 7 вершин та 9 ребер. ПП можна зменшити і до 5 особин, але подальші популяції все ж мають мати 12 хромосом для коректного результату.

ПП також може бути незмінною. Наприклад, у роботах [12] та [19] вона задається як 50 особин. Це є найменше число при якому можуть працювати наведені в статтях алгоритми.

При цьому [12] тестується на різних за величиною графах, найменший з яких має 11 вершин та 20 ребер, а найбільший - 561 вершину та 1629 ребер, це видно на рисунку 2.4.

В роботі ж [19] різниця у величині графів ще більша. Найменший граф використовується той самий з 11 вершинами та 20 ребрами, а от найбільшим є граф з 1000 вершин та 245 000 ребер, який однак має невелику кліку, через що розфарбовується лише 61 кольором.

В роботі [25] також використовується статичне число ПП, що дорівнює

200 особинам. Найменший використовуваний граф тут складається з 74 вершин та 602 ребер, а найбільший з 561 вершини та 3258 ребер.

File	IVI	IEI	Expected $\chi(G)$	k_{min}	HPGAGCP Result	Time (s)
myciel3.col	11	20	4	4	4	0.003
myciel4.col	23	71	5	5	5	0.006
queen5_5.col	23	160	5	5	5	0.031
queen6_6.col	25	290	7	7	8	6.100
myciel5.col	36	236	6	6	6	0.014
queen7_7.col	49	476	7	7	8	6.270
queen8_8.col	64	728	9	9	10	47.482
huck.col	74	301	11	11	11	0.015
jean.col	80	254	10	10	10	0.015
david.col	87	406	11	11	11	0.019
games120.col	120	638	9	9	9	0.027
miles250.col	128	387	8	8	8	0.076
miles1000.col	128	3216	42	42	42	48.559
anna.col	138	493	11	11	11	0.058
fpsol2.i.1.col	496	11654	65	65	65	22.656
homer.col	561	1629	13	13	13	0.760

Рисунок 2.4. – Таблиця графів, що використовувалися як вхідні данні в роботі [12]

2.5.2. Хроматичне число початкової популяції

Підхід до вибору ХЧ початкової популяції у всіх робіт різний. Загалом він складається з чотирьох можливих варіантів:

- всі вершини графа розфарбовуються в один колір, тобто ХЧ ПП дорівнює одному. Як наприклад, в роботі [25]. При цьому підході з кожною новою ітерацією ХЧ збільшується за рахунок ФМ;
- кожна вершина графа розфарбовується в свій унікальний номер, тобто ХЧ ПП графа дорівнює кількості вершин в цьому графі. Такий підхід використано в публікації [19]. При такому підході з кожною ітерацією ХЧ зменшується і відбувається це також за допомогою ФМ, яких в цьому алгоритму дві;
- величина ХЧ необмежена, тобто алгоритм просто фарбує кожен

вершину у випадковий колір. Такий підхід не є ефективним і це розглянуто в статті [14]. Надалі в ній використовується інший підхід, коли ХЧ є наперед визначеним;

- використовуються графи з DIMACS колекції [13], в яких ХЧ вже відоме з роботи інших алгоритмів. Ця відома кількість кольорів використовується як верхня межа та відправний пункт, тож по суті робота алгоритму зводиться до правильного РГ. Такий підхід використовується в таких роботах як [12], [14], [16] та [22];
- використання теореми про степінь вершини, де зазначається, що ХЧ не може бути більшим ніж степінь графа плюс один [27], більш детально теорема розглянута в розділі 2.2. Саме така верхня межа для ХЧ використовується в МГА визначення ХЧ графа, що розглядається у цій роботі.

2.5.3. Умова завершення роботи алгоритму

Існує два види умов завершення роботи ГА визначення ХЧ, при чому в деяких роботах використовують одразу дві умови.

Перша умова – це обмеження за кількістю ітерацій. Тобто після створення визначеної за рахунком популяції алгоритм припиняє роботу навіть якщо шукане рішення до сих пір не знайдено. Таку тактику використовують у статтях [19], де максимальна кількість ітерацій 200 000, та [25], кількість популяцій 5 000.

Друга умова – це ситуація, коли деяка кількість хромосом з популяції мають правильне розфарбовування. Наприклад, робота [22], де умова завершення коли 8 з 10 хромосом популяції мають правильне розфарбовування.

Третя умова – змішана, коли або знайдене рішення без наявності «поганих» ребер в розфарбовування або досягнуте обмеження за кількістю ітерацій. Приклад, стаття [12]. Обмеження за номером популяцій складає 20 000.

Четверта умова – коли РФФ не змінюється задану кількість ітерацій, кількість ітерацій є вхідним параметром алгоритму, який можна легко змінити. Такий підхід використовується в МГА визначення ХЧ, що є предметом розгляду цієї дипломної роботи. Кількість ітерацій дорівнює 10 за замовчуванням.

2.5.4. Фітнес функція

Фітнес функція – це функція, що оцінює кожну хромосому з популяції та присвоює їй певний фітнес рахунок. Це, як правило, ціле число, що показує наскільки дана хромосома відповідає висунутим до неї критеріям. В МГА визначення ХЧ таких критеріїв є два:

- кількість ребер, на кінцях яких вершини, що пофарбовані в однакові кольори або кількість вершин, що з'єднані з ребрами, на протилежних кінцях яких розташовані вершини з тим самим кольором;
- кількість кольорів використаних при РГ.

Градацій оцінок ФФ також є дві:

- чим менший РФФ, тим він кращий. Використовується в таких роботах як [14], [19], [12].
- чим більший РФФ, тим він кращий. Саме така ситуація є класичною для ГА. Вона використовується в такій роботі як [25] та МГА визначення ХЧ, що розглядається в цьому дипломному проекті. В цій

ситуації РФФ має бути менше нуля та поступово наближуватися до нього.

Не зважаючи на те, що критеріїв усього два варіантів реалізації ФФ безліч, в кожній роботі використовується свій.

В роботі [12] рахунком фітнес функції є кількість «поганих» ребер, тобто ребер, на кінцях яких розташовані вершини пофарбовані в один колір. При чому кількість використаних кольорів не грає ролі, так як за основу беруться графи з вже заданим на початку оптимальним ХЧ.

В статті [14] розглядається використання такої технології як ЕСІ, еволюційної системи обчислень реалізованої на мові Java (англ. A Java-based Evolutionary Computation Research System) [15]. І в ній непередбачено використання ФФ з двома підцілями, через що алгоритм досягає локального оптимуму, він або фарбує всі вершини в один колір и тоді РФФ за критерієм використаних кольорів зовсім малий, але граф розфарбований неправильно, або фарбує кожну вершину в різний колір і тоді виникає ситуація, коли граф пофарбований правильно, але кількість кольорів далека від оптимальної. Вихід з цієї ситуації – це задати верхній ліміт ХЧ, що автори й зробили використавши DIMACS графи, в яких ХЧ вже відоме [13]. Тоді РФФ вираховується як кількість неправильно пофарбованих вершин, при чому враховується яку степінь має вершина (реалізація чого не дуже зрозуміла, в роботі не надано коефіцієнти до штрафу за величину ступеня вершини).

В публікації [19] ФФ вираховується як сума зі штрафів за кожен використаний колір плюс кількість суміжних вершин пофарбованих у цей колір.

В роботі [25] ФФ вираховується за формулою: $((100 * \text{кількість неправильно пофарбованих ребер}) + (50 * \text{кількість використаних кольорів}))$

– 1)) / кількість вершин в графі.

В МГА визначення ХЧ, що розглядається в даній роботі використовується така формула підрахунку фітнес рахунку: $k * \text{кількість неправильно пофарбованих вершин} + p * \text{кількість використаних кольорів}$, де k та p параметри, що задаються в залежності від величини графа. Для малих графів k буде щонайменше на 5 одиниць більше p , а для великих графів все буде навпаки. Така залежність була знайдена евристичним методом.

2.5.5. Селекція хромосом

Селекція хромосом полягає у відборі тих хромосом, що будуть задіяні для створення нової популяції, тобто наступного покоління. Існують різні види селекції хромосом, ось деякі з них:

- селекція методом рулетки;

Метод отримав назву за аналогією з відомою азартною грою. Кожній хромосомі надається сектор рулетки, величина якого є пропорційною до РФФ, тобто чим кращий цей рахунок, тим більший сегмент займає хромосома на колесі рулетки. Все колесо відповідає сумі рахунків ФФ кожної хромосоми в популяції. Вибір секції утворюється випадково, причому логічно те, що чим кращий РФФ в хромосоми, тим більша ймовірність її вибору. Програмно це реалізується як вибір числа з інтервалу $[0, \text{сума рахунків фітнес функцій}]$, де числа та їх кількість відповідають сегментам рулетки.

- турнірна селекція;

При використанні турнірної селекції всі особи в популяції діляться на підгрупи, в кожній з яких потім вибирається найкраща хромосома.

Найчастіше розмір таких підгруп 2-3 особи. Існує два способи вибору найкращої хромосоми: перший, вибирається та, в якій найкращий РФФ, і другий, хромосома вибирається випадково.

- рангова селекція;

Цей метод схожий своєю ідеєю на метод рулетки. Всі особи сортуються за їх РФФ, кожній присвоюється певний ранг. Чим більший цей ранг тим більша кількість копій хромосом попаде в наступну популяцію.

- елітарна селекція;

Модифікації методу елітарної селекції є найуживанішими в ГА, це можна побачити на прикладах написаних нижче. Метод полягає в захисті найкращих осіб на наступних ітераціях, вони гарантовано проходять в нову популяцію, при чому в незміненому ГО вигляді.

- селекція з частковою заміною популяції.

Метод ще називають селекцією з зафіксованим станом. Характеризується тим, що частина популяції проходить в наступну без змін. Тобто над нею не проводяться операції схрещування та мутації. Цей метод також є дуже поширеним для модифікацій ГА визначення ХЧ.

В роботі [19] на 50% популяції використовується елітарна селекція, а інші 50% відбираються з усієї популяції випадково, при чому кожна хромосома має рівну з іншими ймовірність бути вибраною.

В МГА визначення ХЧ, що є предметом даної дипломної роботи використовується елітарна селекція. Кількість осіб, що пройдуть у наступну популяцію є параметром, який легко змінити. Отже задана кількість проходить на наступну ітерацію без змін, а це елітарна селекція з частковою заміною популяції. Також як параметр задається кількість хромосом, на яких буде застосований оператор схрещування та кількість

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		29

хромосом, на буде застосований оператор мутації. Ці два параметри можуть бути різними. Ось ці хромосоми також відбираються елітарною селекцією. В четвертому розділі розглянуто оптимальний підбір параметрів в залежності від величини графа та інших параметрів.

В статті [12] використовуються два види селекції: турнірна та елітарна з частковою заміною популяції рисунок 2.5. В турнірній селекції з підгрупи обирається особа з найкращим РФФ, рисунок 2.6.

*Algorithm3: parentSelection2:
define: parent1, parent2*

*parent1 = the top performing chromosome;
parent2 = the top performing chromosome;*

return parent1, parent2;

Рисунок 2.5. – Псевдокод елітарної селекції з частковою заміною популяції в роботі [12]

*Algorithm2: parentSelection1:
define: parent1, parent2, tempParents;*

tempParents = two randomly selected chromosomes from the population;

parent1 = the fitter of tempParents;

tempParents = two randomly selected chromosomes from the population;

parent2 = the fitter of tempParents;

return parent1, parent2;

Рисунок 2.6. – Псевдокод турнірної селекції в роботі [12]

2.5.6. Функція кросоверу

Кросовер – це один з ГО. Він застосовується одразу до двох хромосом,

внаслідок чого вони обмінюються генами.

Кросовер буває:

- з однією точкою схрещування;

На рисунку 2.7 показано як виглядає такий кросовер. Точка схрещування може бути статичною в кожній парі, як наприклад, в статті [19], так і вибиратися випадковим чином окремо для кожної пари, як в роботах [25] та [12], на рисунку 2.8 показано псевдокод такого кросоверу.

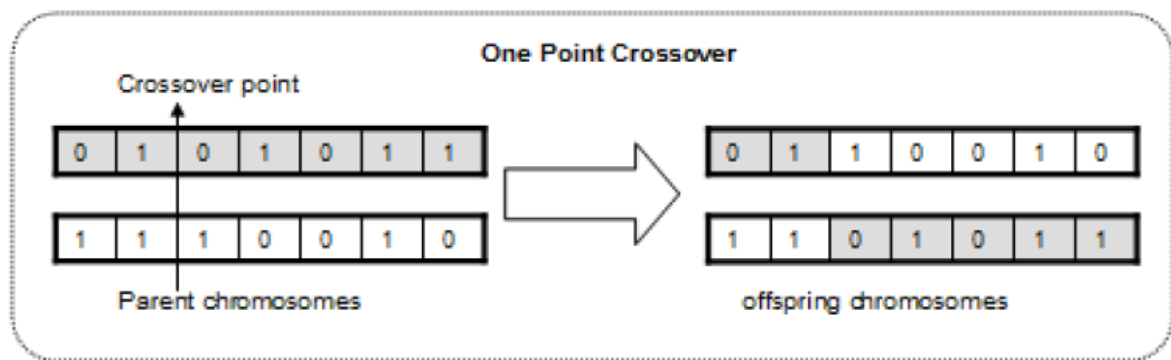


Рисунок 2.7. – Приклад роботи кросовера з однією точкою схрещування з роботи “A Novel Crossover Operator for Genetic Algorithms: Ring Crossover” by Yilmaz Kaya, Murat Uyar and Batman Üniversitesi

Algorithm4: crossover

define: crosspoint, parent1, parent2, child

crosspoint = random point along a chromosome;

child = colors up to and including crosspoint from parent 1 + colors after crosspoint to the end of the chromosome from parent2;

return child;

Рисунок 2.8. – Псевдокод кросоверу з однією випадковою точкою схрещування в роботі [12]

- з двома точками схрещування, рисунок 2.9;

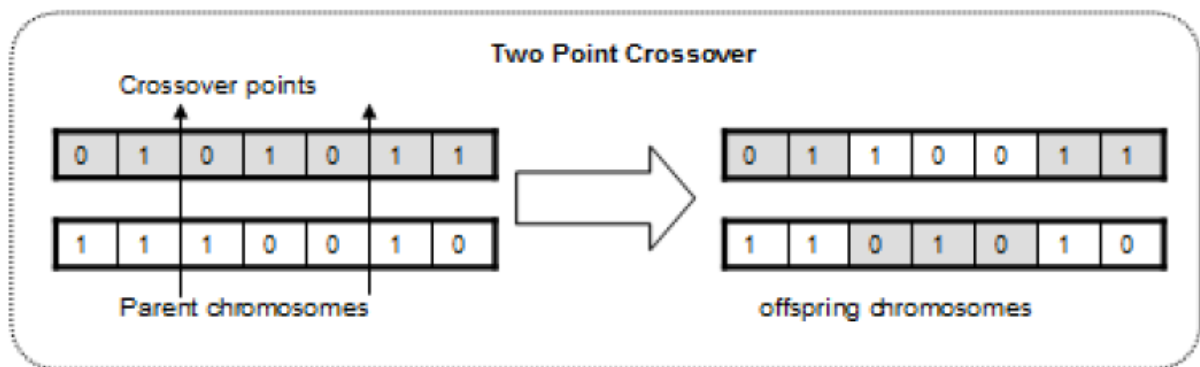


Рисунок 2.9. – Приклад роботи кросовера з двома точками схрещування з роботи “ A Novel Crossover Operator for Genetic Algorithms: Ring Crossover” by Yilmaz Kaya, Murat Uyar and Batman Üniversitesi

- з трьома точками схрещування, так званий кільцевий кросовер (англ. ring crossover) [30].

На рисунку 2.10 показано як виглядає такий кросовер. Його використовують в роботі [16], як заміну пошуку з заборонами, що використовувався раніше разом зі звичайним кросовером з однією точкою схрещування. В дослідженні стверджується, що алгоритм працює так само ефективно з такою цікавою заміною.

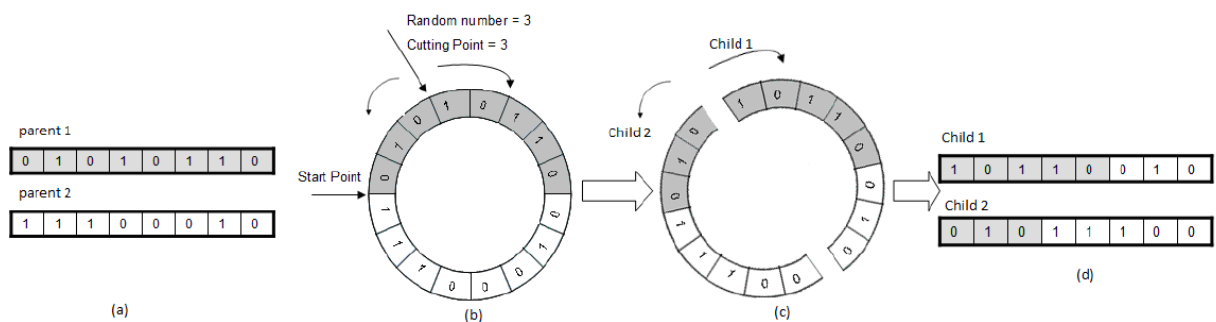


Рисунок 2.10. – Приклад роботи кільцевого кросовера з роботи “ A Novel Crossover Operator for Genetic Algorithms: Ring Crossover” by Yilmaz Kaya

В МГА визначення ХЧ, що розглядається в даній дипломній роботі використовується кросовер з однією точкою схрещування.

2.5.7. Функція мутації

Мутація – це генетична зміна, що призводить до якісно нового прояву основних властивостей генетичного матеріалу [31]. Вважається, що при мутації відбуваються раптові стрибкоподібні зміни в структурі генотипу. Ці зміни можуть бути як позитивні та нейтральні, так і негативні.

Видів мутацій дуже багато, загалом їх вид залежить від специфікації алгоритму. Якщо ФК забезпечує постійним обмін генетичним матеріалом між хромосомами, то функція мутації забезпечує його розмаїття. Такі раптові зміни запобігають досягненню локального оптимуму.

Також в природі саме мутації відповідають за пристосовуваність до навколишнього середовища, коли воно частково або повністю змінюється та запобігає вимиранню виду. ГА перейняв цю її функцію, замість навколишнього середовища в алгоритму є якась проблема, що потребує вирішення і в залежності від її специфікації код всередині ФМ кардинально різний для кожної проблематики, адже його задача - знайти рішення до якоїсь конкретної задачі.

В класичному алгоритмі розділяють генні та хромосомні мутації. В генних змінюються один або кілька генів, а в хромосомних змінюється число, розмір та внутрішня організація. Так як в МГА визначення ХЧ хромосома складається всього лише з одного гену, то й мутації будуть генні. В роботі [14] функція мутації змінює одразу ж кілька генів, адже використовує ідеї з жадібного алгоритму. В статті [19] одна з мутацій також змінює одразу кілька генів, адже розфарбувавши на початку всі вершини в один колір, їй треба зменшувати ХЧ. І це відбувається саме за рахунок

однієї з мутації. В усіх інших роботах, в тому числі й в МГА визначення ХЧ мутації змінюють по одному гену за раз.

В публікації [19], як було вже описано вище, ХЧ ПП дорівнює кількості вершин в графі. В роботі представлено дві ФМ. Перша шукає в графі дві суміжні вершини з один і тим самим кольором і змінює колір першої з них на такий, що задовольняє поточне РГ. Друга ФМ використовується, коли конфліктів з кольором вже немає і повністю виключає випадковий колір з РГ. Тоді знову включається в роботу перша функція і підбирає правильні кольори там, де виникли конфлікти. Потім друга знову створює ці конфлікти виключаючи з розфарбовування новий колір. І так до тих пір поки не стане неможливо усунути конфлікти з поточною кількістю кольорів. Треба зазначити, що хоча цей алгоритм і працює добре, його швидкість досить повільна. Наприклад, граф з 864 вершиною та 18 707 ребрами алгоритм розфарбував за 8 годин.

В роботі [25] використовується популярна для ГА визначення ХЧ функція мутації. В кожній парі з суміжних вершин пофарбованих в той самий колір одна з вершин перефарбовується в будь-який колір крім того, що мала до цього. Псевдокод такої мутації зображений на рисунку 2.11.

```
Mutate(Chromosome ch)  
For each edge connecting gene[u]->gene[v]  
If (gene[u] == gene[v])  
    C = RANDCOLOR  
    such that C != gene[v]  
    gene[u] = C
```

Рисунок 2.11. – Псевдокод функції мутації в роботі [25]

В цьому варіанті також вводяться дві ймовірності мутації, верхня та нижня. Верхня відповідає на питання: для кожних двох хромосом яка ймовірність мутації хоча б однієї з них? Нижня ж розподіляє отриману

верхню ймовірність на дві випадкові частини, де кожна з частин визначає ймовірність мутації однієї хромосоми в парі.

В статті [14] розглядається робота фреймворку ЕСJ для створення повноцінного ГА для задачі РГ. В ЕСJ існують дві функції, що підходять для реалізації мутації. Перша з них – це ColorRand. Вона привласнює вершині випадковий колір з заданої множини кольорів графа. Друга функція – це ColorSafe. Вона привласнює поточній вершині колір не прямого сусіда, а сусіда свого сусіда. Якщо ж всі сусіди вершини є її прямими сусідами, наприклад, у випадку кліки, тоді колір залишається незмінним.

Algorithm5: mutation1:

define: chromosome, allColors, adjacentColors, validColors, newColor;

```
for each(vertex in chromosome) {  
  if (vertex has the same color as an adjacent vertex) {  
    adjacentColors = all adjacent colors;  
    validColors = allColors – adjacentColors;  
  
    newColor = random color from validColors;  
    chromosome.setColor(vertex, newColor)  
  }  
}  
return chromosome;
```

Рисунок 2.12. – Псевдокод мутації з привласнюванням вершині валідного кольору в роботі [12]

Автори пропонують дві ФМ. Перша використовує ColorRand. Вона знаходить непофарбовану вершину та фарбує її за допомогою функції ColorRand. Якщо вершина пофарбована, але є суміжні вершини того ж кольору, то всі ці вершини фарбуються в білий колір, тобто розфарбовуються. Цей спосіб є наполовину жадібним і проходить багато часу до досягнення потрібного результату. Друга функція використовує

ColorSafe та працює так само як і попередня, однак коли колір до вершини без кольору підібрати не вдається, просто залишає її нерозфарбованою.

```
Algorithm6: mutation2:  
define: chromosome, allColors  
  
for each(vertex in chromosome) {  
    if (vertex has the same color as an adjacent vertex) {  
        newColor = random color from allColors;  
        chromosome.setColor(vertex, newColor)  
    }  
}  
return chromosome;
```

Рисунок 2.13. – Псевдокод мутації з привласнюванням вершині випадкового кольору в роботі [12]

В публікації [12] також використовують дві ФМ. Причому одна з них використовується з селекцією зображеною на рисунку 2.5, а інша з селекцією з рисунку 2.6. Одна з функцій кожній неправильно пофарбованій вершині привласнює валідний колір, рисунок 2.12, а інша випадковий, рисунок 2.13. Тут використовується дійсно цікаве рішення, коли РФФ хромосоми далекий від оптимального використовується турнірна селекція і вершини фарбуються в правильні кольори. А от коли РФФ вже дуже близько до шуканого, алгоритм селекцією копіює найкращих батьків та змінює їх генотип випадковою мутацією. Такий підхід дозволяє швидше прийти до потрібного результату. Однак використовувати його можна тільки в ситуаціях, коли відоме оптимальне ХЧ, як в роботі [12]. В МГА визначення ХЧ такий підхід, на жаль, не працює, так як вхідними даними є випадкові графи.

В МГА визначення ХЧ, що розглядається в цій дипломній роботі використовуються 5 видів мутації. Рішення, яку з функцій використовувати приймає користувач. Цілю такого підходу є порівнювання впливу різних

видів мутації на правильність та швидкість роботи алгоритму.

Перша функція просто міняє два гена місцями, тобто фактично змінює колір вершини на колір іншої випадково взятої вершини, а та в свою чергу робить так само.

Друга функція спочатку шукає найгірше розташовану вершину, яка визначається як вершина з найбільшою кількістю суміжних вершин пофарбованих в колір поточної вершини. Потім сортує всі кольори, що використовуються в розфарбовуванні поточного графа, так, щоб найбільшу кількість балів мали ті, в які пофарбовані суміжні до поточної вершини. Чим більше суміжних вершин пофарбовано в даний колір, тим більше балів він набирає. Якщо ж вершина не має суміжних вершин такого кольору, то його рахунок буде нулем. В кінці ФМ фарбує поточну вершину кольором, що набрав найменше балів.

Третя функція працює так само як і друга, однак фарбує поточну вершину в будь-який колір, не сортуючи їх за частотою використання.

Четверта функція шукає будь-які дві суміжні вершини пофарбовані в один колір, потім сортує кольори за вживаністю, як це робить друга функція мутації, та привласнює поточній вершині колір з найменшою кількістю балів.

П'ята функція працює так само як і четверта, однак фарбує поточну вершину в будь-який колір, не сортуючи їх за частотою використання.

Псевдокод усіх цих функцій поданий та розглянутий в третьому розділі, а наслідки їхньої роботи розглянуті в четвертому розділі.

2.5.8. Створення нової популяції

Хромосоми, отримані в результаті застосування ГО до хромосом

тимчасової батьківської популяції, включаються до нової популяції. І вона стає поточною популяцією на даному етапі роботи ГА. Кожна нова популяція також може називатися ітерацією. А кількість ітерацій – це кількість створених популяцій за роботу алгоритму з один графом.

Цікаво, що ГО можуть застосовуватися як на одній і тій самій множині хромосом, а можуть і на різних. Перший випадок ілюструє робота [12], псевдокод цієї частини можна побачити на рисунку 2.14. А другий випадок ілюструє робота [19], псевдокод можна побачити на рисунку 2.15.

В роботі [19] до половини відібраної селекцією популяції (50% елітарна та 50% випадкова з тих, що залишилися, дивитися в пункті 2.5.5.) застосовується ОК, а до іншої половини ОМ.

```
// a single run of a genetic algorithm  
function gaRun() {  
    parents = getParents();  
    child = crossover(parents);  
    child = mutate(child);  
    population.add(child);  
}
```

Рисунок 2.14. – Псевдокод застосування генетичних операторів в роботі [12]

В роботі [12] розмір популяції завжди однаковий. Половина найгірших за РФФ особин забирається назавжди, до половини найкращих застосовуються ГО. Інша ж половина – це нові випадково створені хромосоми. І такий процес відбувається з кожною ітерацією.

В статті [16] нова популяція завжди складається з однієї найкращої за РФФ хромосоми, що залишається незмінною та просто копіюється до нової популяції. До інших же хромосом застосовується ОК.

В роботі [22] нова популяція завжди складається з 40% найкращих

хромосом попередньої популяції, що просто копіюються до поточної, ще з 40% найкращих хромосом, до яких застосували ОК, та з 20% найкращих хромосом, до яких застосували мутацію.

В МГА визначення ХЧ є кілька параметрів, що визначають формування нової популяції. Як вже сказано в 2.5.5. в даному алгоритмі використовується елітарна селекція. Оператори кросинговеру та мутації застосовуються на різних хромосомах, як це є в роботі [19].

```

Genetic_GraphColoring()
{
  M = Population size.
  Ng = Number of generations
  N0 = Number of offsprings =  $\frac{N_g}{2}$ 
  Get the population size and the nb. of generations (Ng)
  Generate an initial population, current_pop
  for i = 1 to M
    evaluate(current_pop)
  Keep_the_best()
  NumberColors = V
  for i = 0 to Ng do
    {
      if (BestFitness < NumberColors)
        Squeeze the colors set
      Divide Population into two halves, P1 and P2
      for (i = 1 to N0)
        Select two chromosomes randomly from P1
        apply crossover with probability Pc
        Add offspring to the population
      for (i = 1 to N0)
        Select a chromosome from P2
        apply mutation with probability Pm
        Add offspring to the population
      Evaluate the population fitness for Pi
      new_pop = select(current_pop, offspring)
      current_pop ← new_pop
    }
}

```

Рисунок 2.15. – Псевдокод роботи ГА в роботі [19]

Користувач вказує, яка кількість хромосом копіюється до нової популяції, до якої кількості застосовується ФК, а до якої – мутації. Також

вказує, скільки нових випадково створених хромосом додасться до нової популяції. Все це є параметрами, значення яких не залежать один від одного. Тож по суті, нова популяція в МГА визначення ХЧ складається зі скопійованих хромосом, хромосом після кросинговеру та після мутації і нових випадково створених. Все це дозволяє постійно оновлювати популяцію та запобігати виникненню локального оптимуму. В залежності від величини графа можна змінювати пропорції в параметрах тим самим покращуючи остаточний результат.

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		40

3. СТРУКТУРА ТА РЕАЛІЗАЦІЯ МОДИФІКОВАНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ВИЗНАЧЕННЯ ХРОМАТИЧНОГО ЧИСЛА ГРАФА

3.1. Призначення пакетів та файлів програми

Програма МГА визначення ХЧ складається з трьох пакетів та окремого файлу main.

Перший пакет graph складається з файлів edge.py, graph.py та point.py, кожен з яких складається з класів Edge, Graph, Point відповідно. Клас Edge описує ребра графа, Point вершини графа, а Graph сам граф, тобто послідовність ребер та вершин графа та їх взаємозв'язок.

Другий пакет util складається з файлів stats.py та params.py, що містять в собі класи Stats та Params відповідно. Перший містить в собі всю статистику про РГ та виводить її на екран. В другому розташовані усі вхідні параметри алгоритму та функції, за допомогою яких ці параметри можна встановити в програмі. Також пакет містить файл util з двома функціями get_random_points() та get_random_edges(), що створюють ВГ.

Третій пакет genetic містить в собі файли gene.py та population.py, що складаються з класів Gene та Population відповідно. Перший описує окрему хромосому та функції мутації, кросоверу, оцінки, що застосовуються до неї. Другий же описує все цілу популяцію таких хромосом та містить в собі функції селекції, вибору найкращої хромосоми, виду кросинговеру та мутації, створення популяції з випадковими даними.

Файл main основну функцію, що працює з екземплярами всіх описаних вище класів, створює початкову популяцію та всі наступні, відповідає за критерій зупинки алгоритму, зберігає у файл ВГ або завантажує його для роботи.

Для написання алгоритму використовувався pure Python, окрім фрагментів з використанням бібліотеки operator (для сортування словників), функцій randint() і choice() з модулю random, perf_counter() з модулю time та модулю json для роботи з файлами.

3.2. Вихідні параметри МГА визначення ХЧ

Екземпляр класу Stats, як вже було написано вище, містить в собі статистику про кінцевий результат застосування МГА визначення ХЧ на визначеному ВГ. Полями класу є:

- кількість ітерацій;

Тобто скільки популяцій знадобилося для досягнення кінцевого результату. В Python це реалізується просто як окрема змінна-лічильник “iterations” в циклі створення наступної популяції та її оцінки і так доки не буде виконано критерій зупинки алгоритму.

- Час роботи МГА;

Відстежується за допомогою perf_counter() лічильника з модулю time. Він був обраний через свою точність. Час вимірюється в секундах з 18 знаками після коми. Кожен наступний виклик perf_counter() повертає час, що пройшов з його найпершого виклику.

- Хроматичне число;

Як вже було сказано, хромосома представляє собою список з кольорами вершин. За допомогою Python кількість кольорів можна легко визначити перетворивши список в множину (в множині немає однакових елементів): len(set(gen)), де gen – це хромосома.

- Найкраща хромосома з останньої популяції;

- Існуючі конфлікти;

Вираховується за формулою $-1 * (\text{фітнес рахунок хромосоми} + (\text{ХЧ} * \text{штраф за використаний колір})) / \text{штраф за однакові кольори суміжних вершин}$. Так як РФФ завжди від'ємний, то при правильному розфарбуванні ми отримуємо 0 в цій частині: фітнес рахунок хромосоми + (ХЧ * штраф за використаний колір). За цією ж причиною використовується -1 при неправильному розфарбуванні. Пошук конфліктів використовується для перевірки правильності роботи алгоритму (при зміні параметрів) та в ситуаціях, коли користувач наперед визначає ХЧ і хоче перевірити чи можливо розфарбувати граф в таку кількість кольорів без виникнення помилок.

- Сам граф, як об'єкт класу Graph.

Після роботи алгоритму всі ці поля виводяться в консоль. Для реалізації цієї цілі був перевизначений вбудований метод `__str__()` класу Stats. Підрахунок полів класу можна побачити в додатку А.

3.3. Опис параметрів МГА визначення ХЧ

Вхідні параметри алгоритму описані як поля класу Params. В програмі їх можна змінювати за допомогою методів `set_назва_параметру()`.

- `penalty_per_same_color`

Штраф, що використовується у ФФ та призначається за кожні дві суміжні вершини одного кольору.

- `penalty_per_color_used`

Штраф, що використовується у ФФ та призначається за кожен використаний колір в РГ.

- multiplier

Початково дорівнює одиниці. Коефіцієнт для визначення кількості хромосом, що приймають участь у створення нової популяції.

- initial_population_size

Розмір ПП. Визначається як ціле число помножене на multiplier.

- crossover_parents

Кількість хромосом з батьківської популяції (вже відібраних елітарною селекцією), на яких застосовується ОК. Визначається як ціле число number помножене на multiplier. Треба зазначити, що дітей після кросоверу буде $number * number - number$.

- mutation_parents

Кількість хромосом з батьківської популяції (вже відібраних елітарною селекцією), на яких застосовується ОМ. Визначається як ціле число number помножене на multiplier.

- random_count

Кількість хромосом, що будуть створені за тими ж самими правилами, що і хромосоми з ПП, та додані до нової популяції. Потрібні для запобігання досягнення локального оптимуму, забезпечення різноманіття генетичного матеріалу. Визначається як ціле число number помножене на multiplier.

- propagation_count

Кількість хромосом з батьківської популяції відібраних елітарною селекцією, що копіюються без змін до нової популяції. Визначається як ціле число помножене на multiplier.

- stop_genetic_after_count

Є ознакою завершення роботи алгоритму. Ціле число, що визначає скільки генів підряд повинно мати однаковий РФФ аби робота алгоритму припинилася. За замовчуванням дорівнює 10.

- `no_colours`

Є вже відомим ХЧ. Використовується в ситуаціях, коли треба перевірити, чи можливо розфарбувати граф такою кількістю кольорів. За замовчуванням не використовується.

- `mutation_function`

Визначає, яка ФМ використовується. Значення задається числом в діапазоні 0-4 в подвійних лапках (наприклад, "2").

- `crossover_function`

Визначає, яка ФК використовується. Значення може бути "Standard" або "Jumbled", їх розглянуто детально в розділі 3.6.

3.4. Реалізація графа

В МГА визначення ХЧ граф реалізується з допомогою трьох класів, див. розділ 3.1.

Вершини ініціалізуються за допомогою двох координат x та y , що є цілими числами. Значення цих координат в кожній вершині унікальні, реалізуються функцією `randint()` (обирає випадкове число з заданого діапазону), уникаючи повторювань. Це зроблено аби потім граф можна було просто зобразити графічно в двовимірній площині.

Ребра графа ініціалізуються за допомогою двох вже створених об'єктів-вершин, що є початком та кінцем ребра.

А в класі графа містяться три поля: ХЧ графа на поточний момент (воно

постійно змінюється), список з об'єктів, що є вузлами графа та список з об'єктів, що є ребрами графа.

У всіх трьох класів перевизначений метод `__str__()` для коректного виводу в консоль.

3.5. Вибір ФК, ФМ, створення пулу випадкових хромосом, елітарна селекція, визначення РФФ найкращої хромосоми в популяції

Клас `Population` описує загальні функції, що виконуються над визначеною множиною хромосом. Програмно ця множина описана списком генів, кожен з яких є об'єктом класу `Gene`. Містить два статичних методи: функцію `crossover()` та `mutate()`. Обидві функції як параметр приймають список хромосом, що є батьками нової популяції.

В методі `crossover()` обирається спосіб кросинговеру, яких є два `Standard` та `Jumbled`. Псевдокод обох наданий та описаний в розділі 2.6. Якщо як метод вказано `"None"`, то операції кросинговеру в МГА не відбувається. В псевдокоді 3.1. показаний механізм виклику `Jumbled`-кросоверу, що є методом класу `Gene`. Виклик інших типів кросоверів відбувається аналогічно. `Dad` та `mom` по суті є лічильниками списку `parents`. Кросовер відбувається для кожної такої пари (якщо вони не дорівнюють один одному), при чому порядок має значення (пари `dad-mom` та `mom-dad` не ріноцінні).

В методі `mutate()` обирається спосіб мутації, яких є п'ять. Всі вони описані в розділі 2.5.7. Їх псевдокод наданий та описаний в розділі 2.6. А блок-схема мутації з пошуком найгірше розміщеної вершини показана на додатку 4. Якщо як метод вказано `"None"`, то операції мутації в МГА не відбувається. Застосовується оператор до кожного елементу списку, що складається з хромосом.

```

for dad in parents:
    for mom in parents:
        if not mom == dad:
            children = mom.crossover_1(dad)

```

Псевдокод 3.1. - Механізм виклику Standard-кросоверу, що є методом класу Gene.

Метод `get_max_evaluation()` повертає РФФ найкращого гена в популяції. Для цього потрібно розрахувати РФФ для кожної хромосоми окремо, що робиться за допомогою метода `evaluate()` з класу Gene, псевдокод 3.2.

```

max_evaluation = minus_infinity
for gene in genes:
    if gene.evaluate() > max_evaluation:
        max_evaluation = gene.evaluate()
return max_evaluation

```

Псевдокод 3.2. - Метод `get_max_evaluation()`

Мінус нескінченність в Python можна реалізувати як `-1 * float("Inf")`, де `float("Inf")` визначає незрівнянно велике число.

Елітарна селекція реалізується в класі `Population` методом під назвою `best_n()`, який повертає список з `n` хромосом з самим високим РФФ. Спочатку створюється пустий словник виду: хромосома – її фітнес рахунок, що поступово в циклі заповнюється значеннями за допомогою метода `evaluate()` з класу Gene. Далі дані в словнику сортуються за значенням та метод повертає `n` кращих.

Також в класі `Population` містяться методи, ціллю яких є створення пулу з хромосом, де вершини мають випадкове розфарбування. Ілюстрація цього процесу показана в блок-схемі додатку 7. Це реалізується за допомогою

двох методів `random` та `get_random_parent()`.

В першому в циклі, кількість обертів якого визначає параметр `random_count`, див. розділ 3.3, викликається функція `get_random_parent()`. Метод повертає список з вже створених хромосом, псевдокод 3.3.

```
children = список
for i to random_count:
    children.append(get_random_parent())
return children
```

Псевдокод 3.3. - Метод `random()`

В методі `get_random_parent()` знаходиться степінь графа, більш детально про це можна прочитати в розділі 2.2. Ця степінь графа є верхньою межею ХЧ. Враховуючи це створюється масив довжиною в кількість вершин графа та заповнюється кольорами в діапазоні від 0 до ступеня графа за допомогою функції `randint()`. Псевдокод 3.4.

```
sample_numbers = список з першим елементом, що дорівнює 0
for i to graph.no_points - 1:
    sample_numbers[i] = randint(0, colors_used - 1)
return Gene(sample_numbers)
```

Псевдокод 3.4. – Метод `get_random_parent()`

3.6. Функції кросоверу, мутації, оцінки

Клас `Gene` описує окрему хромосому та дії над нею, така назва обрана тому, що в МГА визначення ХЧ генотип складається лише з однієї хромосоми, див. розділи 2.3. і 2.4. Цей ген представлений масивом цілих чисел, саме він ініціалізується в конструкторі класу.

Над хромосомою можна виконати оператори кросинговеру (тут як параметр передається друга хромосома, адже батьків повинно бути двоє) та мутації, а також оцінити її ФФ. В МГА визначення ХЧ представлена одна функція оцінки, дві кросоверу та п'ять мутацій. В розділі 4 порівнюються їх результативність, розглядаються причини успіху та питання до якого графа, яка функція підходить більше.

Фітнес функція представлена методом `evaluate()`. Вона призначає штраф за кожні дві суміжні вершини однакового кольору та штраф за кожен використаний колір. Величина штрафів є вхідним параметром, див. розділ 3.3. В залежності від величини графа, вони мають пропорційно змінюватись для коректної роботи МГА, див. розділ 4. Значення ФФ завжди від'ємне, та чим ближче воно до нуля (тобто чим більшим є число) тим воно краще. Псевдокод 3.5.

```
evaluation = 0
for edge in graph.edges:
    if gene[edge.start] = gene[edge.end]:
        evaluation += penalty_per_same_color
evaluation+=len(set(self.array))*penalty_per_color_used
return -1 * evaluation
```

Псевдокод 3.5. – Метод `evaluate()`

ОК реалізується двома методами `crossover()` та `crossover_1()`.

Перший, `crossover()`, є класичним варіантом з однією точкою перетину. До точки перетину генетичний матеріал дитини складається з генів одного з батьків, а після з генів другого. Якщо генів в хромосомі непарна кількість, наприклад, 7, то в дитині буде 3 гена від одного і 4 від другого, округлення відбувається в меншу сторону. Псевдокод 3.6.

```
def crossover(self, obj2):
```

```

child = []
for i to no_points:
    if i < no_points / 2:
        child.append(self.gene[i])
    else:
        child.append(obj2.gene[i])
return Gene(child)

```

Псевдокод 3.6. – Метод `crossover()`

Другий метод кросинговеру, `crossover_1()`, має безліч точок перетину. Гени з парними індексами та індексом нуль беруться від одного батька, а з непарним індексом від іншого. Цей метод забезпечує більше різноманіття генетичного матеріалу та меншу стабільність, що однак компенсується елітарною селекцією. Псевдокод 3.7.

```

def crossover_1(self, obj2):
    child = []
    for i to no_points:
        if i mod 2 = 0:
            child.append(self.gene[i])
        else:
            child.append(obj2.gene[i])
    return Gene(child)

```

Псевдокод 3.7. – Метод `crossover_1()`

Оператор мутації реалізується п'ятьма методами: `mutate()`, `mutate_1()`, `mutate_2()`, `mutate_3()`, `mutate_4()`.

Перший метод `mutate()` змінює місцями два випадкових гена (тобто перефарбовує дві вершини) за допомогою статичного методу `swap()`. Ген

для обміну обирається за допомогою методу randint() з модулю random. В Python зміну зробити дуже просто, однак потрібно пам'ятати, що список – це змінювана структура даних, тож зміну треба робити на копії гена, що реалізується за допомогою зрізу. Псевдокод 3.8.

```
def mutate(self, graph="None"):
    i = randint(1, len(self.gene) - 1)
    j = randint(1, len(self.gene) - 1)
    return Gene(swap(self.gene, i, j))
```

Псевдокод 3.8. – Метод mutate()

```
def mutate_1(self, graph):
    worst_pos = get_worst_placed_node(self.gene, graph)
    final_color = get_final_color(worst_pos, self.gene,
graph)
    new_gene = self.gene[:]
    new_gene[worst_pos] = final_color
    return Gene(new_gene)
```

Псевдокод 3.9. – Метод mutate_1()

В другому методі mutate_1(), псевдокод 3.9., «найгіршій» вершині привласнюється найбільш «валідний» колір з вже наявних в розфарбовуванні кольорів. «Найгірша» вершина, це вершина, що має найбільше суміжних вузлів того ж кольору, що й вона сама. Її пошук виконується за допомогою статичного методу get_worst_placed_node(), псевдокод 3.10. Найбільш «валідний» колір знаходиться за допомогою методу get_final_color(), псевдокод 3.11. В ньому створюється словник, де ключами є всі наявні в розфарбовуванні кольори, а значеннями цілі числа – кількість суміжних до поточної вершини вузлів пофарбованих у такий колір. Зі словника обирається такий колір значення якого найменше.

Зм	Лист	№ докум.	Підп.	Дата


```

def get_worst_placed_node(gene, graph):
    scores_dict = словник
    for edge in graph.edges:
        if gene[edge.start] = gene[edge.end]:
            if edge.start is in scores_dict:
                scores_dict[edge.start] += 1
            else:
                scores_dict[edge.start] = 1
        if edge.end is in scores_dict:
            scores_dict[edge.end] += 1
        else:
            scores_dict[edge.end] = 1
    return ключ в словнику, значення якого найбільше

```

Псевдокод 3.10. – Метод `get_worst_placed_node()`

```

def get_final_color(worst_pos, gene, graph):
    connected_nodes = список
    for edge in graph.edges:
        if edge.start = worst_pos:
            connected_nodes.append(edge.end)
        if edge.end = worst_pos:
            connected_nodes.append(edge.start)
    colors_dict = словник
    for i in set(gene):
        colors_dict[i] = 0
    for node in connected_nodes:
        colors_dict[gene[node]] += 1

```

return ключ в словнику, значення якого найменше

Псевдокод 3.11. – Метод `get_final_color()`

В третьому методі `mutate_2()`, псевдокод 3.12., також шукається «найгірша вершина» за допомогою методу `get_worst_placed_node()`, але тут їй вже привласнюється випадковий колір з тих, що вже використовуються в розфарбовуванні. Це робиться за допомогою методу `choice()` з модулю `random`, що обирає випадковий елемент зі списку.

```
def mutate_2(self, graph):
    worst_pos = get_worst_placed_node(self.gene, graph)
    new_gene = self.gene[:]
    new_gene[worst_pos] = choice(list(set(self.gene)))
    return Gene(new_gene)
```

Псевдокод 3.12. – Метод `mutate_2()`

В четвертому методі `mutate_3()`, псевдокод 3.13., знаходиться будь-яка неправильно пофарбована вершина за допомогою статичного методу `get_any_wrongly_colored_node()`, псевдокод 3.14. та фарбується в правильний колір за допомогою вже розглянутого вище методу `get_final_color()`.

```
def mutate_3(self, graph):
    pos = get_any_wrongly_colored_node(self.gene, graph)
    new_gene = self.gene[:]
    new_gene[pos] = get_final_color(pos, self.gene,
graph)
    return Gene(new_gene)
```

Псевдокод 3.13. – Метод `mutate_3()`

```
def get_any_wrongly_colored_node(gene, graph):
```

```

for edge in graph.edges:
    if gene[edge.start] = gene[edge.end]:
        return edge.start
return 1

```

Псевдокод 3.14. – Метод `get_any_wrongly_colored_node()`

В п'ятому методі `mutate_4()`, псевдокод 3.15., також знаходиться будь-яка неправильно пофарбована вершина за допомогою статичного методу `get_any_wrongly_colored_node()` та фарбується в випадковий колір з наявних за допомогою методу `choice()`.

```

def mutate_4(self, graph):
    pos = get_any_wrongly_colored_node(self.gene, graph)
    new_gene = self.gene[:]
    new_gene[pos] = choice(list(set(self.gene)))
    return Gene(new_gene)

```

Псевдокод 3.15. – Метод `mutate_4()`

3.7. Основний цикл роботи МГА визначення ХЧ

В файлі `main` містяться функції `work()`, `do_genetic()`, `population_propagation_default()`, `find_out_chorom_num()`, а також функції для роботи з файлами `in_file()` та `out_file()`. За допомогою них функціонують всі описані вище класи.

В функції `in_file()`:

- вводиться кількість ребер та вершин ВГ;
- виконується перевірка на кількість ребер, тобто чи є коректним те число, що ввів користувач. Відбувається це за допомогою формули

```
no_edges > no_points * (no_points - 1) / 2 [27];
```

- викликаються функції створення ВГ з модуля util: get_random_points() та get_random_edges();
- виконується запис списку вершин в файл під назвою points, а списку ребер в файл edges з використанням бібліотеки json.

В функції out_file() виконується вивантаження списків вершин та ребер з відповідних файлів та ініціалізація їх як об'єктів класів. Функція work() також викликається в цьому місці.

В функції find_out_chorom_num(), псевдокод 3.16., знаходиться верхня межа ХЧ, яка є степеню графа, детальніше в розділі 2.2.

```
dictionary = словник
```

```
for i to кількість вершин графа:
```

```
    dictionary[i] = 0
```

```
for edge in graph.edges:
```

```
    dictionary[edge.start] += 1
```

```
    dictionary[edge.end] += 1
```

```
    chrom_num = ключ словника, що має найбільше значення
```

Псевдокод 3.16. – Метод find_out_chorom_num()

В функції work(), псевдокод 3.17:

- використовується лічильник perf_counter(), що замірює час роботи алгоритму, див. розділ 3.3;
- виконується ініціалізація ВГ як об'єкту;
- викликається метод знаходження степені графа;
- виконується основний цикл бінарного пошуку ХЧ;

- викликається функція ініціалізації ПП;
- викликається функція do_genetic();
- викликається конструктор класу Stats, що друкує статистику про роботу алгоритму.

```
def work(points, edges):
    t1 = perf_counter()
    graph = Graph(points, edges)
    chrom_num = find_out_chrom_num(graph)
    min = 1
    save_result = None
    iterations_sum = 0
    while chrom_num > min:
        result = int((chrom_num + min) / 2)
        population = initialize_population(graph, result)
        iterations, best_gene, conflicts =
do_genetic(population, graph, result)
        colors_used = len(set(best_gene.gene))
        iterations_sum += iterations
        time = perf_counter() - t1
        Stats(iterations, time, colors_used, best_gene,
conflicts, graph)
        if conflicts = 0:
            chrom_num = result
            save_result = result
        else:
            min = result + 1
    time = perf_counter() - t1
```

виведення в консоль остаточного ХЧ, часу роботи алгоритму та кількості ітерацій

Псевдокод 3.17. – Метод work()

В функції do_genetic(), псевдокод 3.18 крутиться цикл, умовою завершення якого є виконання термінальної умови МГА. Цією ознакою є ціле число, що визначає скільки генів підряд повинно мати однаковий РФФ аби робота алгоритму припинилася (за замовчуванням дорівнює 10), а це значення параметру stop_genetic_after_count(), див. розділ 3.3. Також do_genetic() викликає функцію population_propagation_default().

```
def do_genetic(population, graph):
    iterations = 0
    last_n = список из stop_genetic_after_count элементов,
    що є нескінченностями
    best_gene = None
    while True:
        iterations += 1
        max_evaluation=population.get_max_evaluation(graph)
        видаляення зі списку першого елемента
        вставлення в кінець списку max_evaluation
        flag = False
        for i to len(last_n):
            if i != 0 and last_n[i] != last_n[i - 1]:
                flag = True
        if not flag:
            break
        best_gene = population.best_n(1, graph)
        new_population=population_propagation_default(population, graph)
```

Зм	Лист	№ докум.	Підп.	Дата

```

    population = Population(new_population)
    Graph.no_colours = len(set(best_gene.gene))
    eval = best_gene.evaluate(graph)
    conflicts = -1 * (eval + (Graph.no_colours *
    Params.penalty_per_color_used)) /
    Params.penalty_same_color

    return iterations, best_gene, conflicts, colors_used

```

Псевдокод 3.18. – Метод do_genetic()

Функція population_propagation_default(), псевдокод 3.19., є загальною функцією, що викликає більш спеціалізовані методи задля створення нової популяції. Використовує такі параметри crossover_parents, mutation_parents, propagation_count, random_count, які були детально розглянуті в розділі 3.3.

```

def population_propagation_default(population, graph):
    parents_crossover = population.best_n(crossover_parents,
    graph)
    parents_mutation = population.best_n(mutation_parents,
    graph)
    new_population = список

    new_population.extend(population.best_n(propagation_count,
    graph))

    new_population.extend(population.crossover(parents_crossover))

    new_population.extend(population.mutate(parents_mutation,
    graph))

    new_population.extend(population.random(random_count,
    graph.no_points, colors_used))

```

```
return new_population
```

Псевдокод 3.19. – Метод `def population_propagation_default()`

3.8. Створення випадкового графа

В файлі `util.py` з пакету `util` містяться дві функції: `get_random_points()`, псевдокод 3.20., створює список вершин ВГ, а `get_random_edges()`, псевдокод 3.21., список ребер ВГ. При чому виконуються перевірки на повторювані вершини, щоб не було двох з однаковими координатами, на відсутність петель, на ідентичні ребра та на неорієнтованість графа, тобто ребро (2, 3) дорівнює ребру (3, 2).

```
def get_random_points(no_points):
    points_list = список
    i = 0
    while i < no_points:
        x = random.randint(0, 100)
        y = random.randint(0, 100)
        new_point_list = [x, y]
        f = False
        for point in points_list:
            if point[0] = new_point_list[0] and point[1] =
new_point_list[1]:
                f = True
        if not f:
            points_list.append(new_point_list)
            i = i + 1
    return points_list
```

Зм	Лист	№ докум.	Підп.	Дата

Псевдокод 3.20. – Метод `get_random_points()`

```
def get_random_edges(no_points, no_edges):
    edges_list = список
    i = 0
    while i < no_edges:
        start = random.randint(0, no_points - 1)
        end = random.randint(0, no_points - 1)
        f = False
        if start = end:
            continue
        else:
            new_edge_list = [start, end]
            for edge in edges_list:
                if (edge[0] = new_edge[0] and edge[1] =
new_edge[1])\
                    or (edge[1] = new_edge[0] and edge[0] =
new_edge[1]):
                    f = True
            if not f:
                edges_list.append(new_edge_list)
            i += 1
    return edges_list
```

Псевдокод 3.21. – Метод `get_random_edges()`

Зм	Лист	№ докум.	Підп.	Дата

ІАЛЦ.045420.004 ПЗ

Лист

60

4. ПОРІВНЯЛЬНИЙ АНАЛІЗ РОБОТИ ГЕНЕТИЧНИХ ОПЕРАТОРІВ МОДИФІКОВАНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ВИЗНАЧЕННЯ ХРОМАТИЧНОГО ЧИСЛА ГРАФА

В МГА визначення ХЧ були реалізовані дві ФК та п'ять ФМ, вони детально описані в попередніх розділах. Як вхідна інформація використовуються ВГ. З огляду на це аналіз роботи алгоритму зводиться до тестування одних і тих самих графів, при застосуванні різних комбінацій ГО.

Для роботи з ВГ було створено дві функції `in_file()` `out_file()` для створення та збереження згенерованих графів у файл. Ці графи є додатком 2 до диплому.

Було вирішено тестувати МГА на чотирьох графах різного розміру:

- малий – 7 вершин та 9 ребер;
- середній з малою кількістю ребер – 178 вершин та 567 ребер;
- середній з більшою кількістю ребер – 178 вершин та 1484 ребер;
- великий – 581 вершина та 11348 ребер.

Критеріями порівняння є:

- кількість ітерацій МГА;
- час роботи МГА;
- величина ХЧ.

Також в ході тестування були визначені оптимальні за часом та результатом роботи параметри:

- `penalty_per_same_color = 10` та `penalty_per_color_used = 1`. На малому графі та середньому з меншою кількістю ребер

penalty_per_same_color може бути п'ять або навіть менше. Головне, щоб пропорційно значення цього штрафу було більше за penalty_per_color_used, інакше ймовірність виникнення конфліктів в результаті різко збільшується.

- crossover_parents = 3, mutation_parents = 10, random_count = 3, propagation_count = 50 оптимальні пропорції для створення нових популяцій. Зменшення частки мутації до 3, тобто mutation_parents = 3 також працює непогано, однак час роботи збільшується. Загалом тенденція така, що при частці мутації від 0 до 3 працює значно гірше, від 3 до 10 час роботи пропорційно зменшується, а от якщо зробити mutation_parents > 10, то час роботи почне погіршуватися, а результат залишиться таким самим. Щодо кросоверу, то тут оптимальне значення 3, зважаючи на те, що дітей при цьому буде 6., див. розділ 3.3. Зменшувати значення недоцільно, а при збільшенні результат такий самий зі значно більшим часом роботи. Це особливо помітно на великому графі.
- stop_genetic_after_count = 10. Так як всі графи тестуються при однакових параметрах, то таке значення оптимально. Але для малого графа та частково середніх при збільшенні параметру, ймовірність кращого результату збільшується, а от для великих графів це суттєво збільшує час роботи. При цьому треба зауважити, що для Jumbled кросинговеру значення потрібно збільшувати в меншій мірі ніж для Standard.

При аналізі результатів МГА треба зважати, що багато залежить від випадкового розфарбовування вершин на початку роботи алгоритму. Якщо розфарбовування вдале, то час роботи покращується та ХЧ може бути меншим. В таблицях нижче розглядаються загальні тенденції.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4	None
Standard	27	22	25	24	24	33
Jumbled	28	23	25	24	25	24

Таблиця 4.1.1. – Кількість ітерацій МГА для графа на 7 вершин та 9 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4	None
Standard	0.0485	0.0378	0.0464	0.0437	0.04217	0.0524
Jumbled	0.0505	0.0399	0.0422	0.0423	0.041	0.0338

Таблиця 4.1.2. – Час роботи МГА для графа на 7 вершин та 9.

В усіх випадках у графі з таблиць 4.1.1. та 4.1.2. ХЧ буде три.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4	None
Standard	270	181	215	125	273	не працює
Jumbled	338	129	305	100	308	не працює

Таблиця 4.2.1. – Кількість ітерацій МГА для графа на 178 вершин та 567 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	25.9369	16.5852	21.1169	11.9099	25.5745
Jumbled	31.45	11.7982	30.4063	9.3298	28.3542

Таблиця 4.2.2. – Час роботи МГА для графа на 178 вершин та 567 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	10	8	7	7	8
Jumbled	11	7	6	7	6

Таблиця 4.2.3. – Хроматичне число МГА для графа на 178 вершин та 567 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	635	248	446	214	588
Jumbled	не працює	306	504	172	599

Таблиця 4.3.1. – Кількість ітерацій МГА для графа на 178 вершин та 1484 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	151.8735	60.0248	110.528	47.2168	141.592
Jumbled	не працює	74.1851	124.282	40.4742	143.213

Таблиця 4.3.2. – Час роботи МГА для графа на 178 вершин та 1484 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	19	10	10	12	11
Jumbled	не працює	11	11	13	11

Таблиця 4.3.3. – Хроматичне число МГА для графа на 178 вершин та 1484 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	1381	1124	2496	693	2883
Jumbled	1371	1175	2258	1199	3108

Таблиця 4.4.1. – Кількість ітерацій МГА для графа на 581 вершин та 11348 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	2489	3511	6065	1467	4759
Jumbled	2493	2240	4050	2931	5528

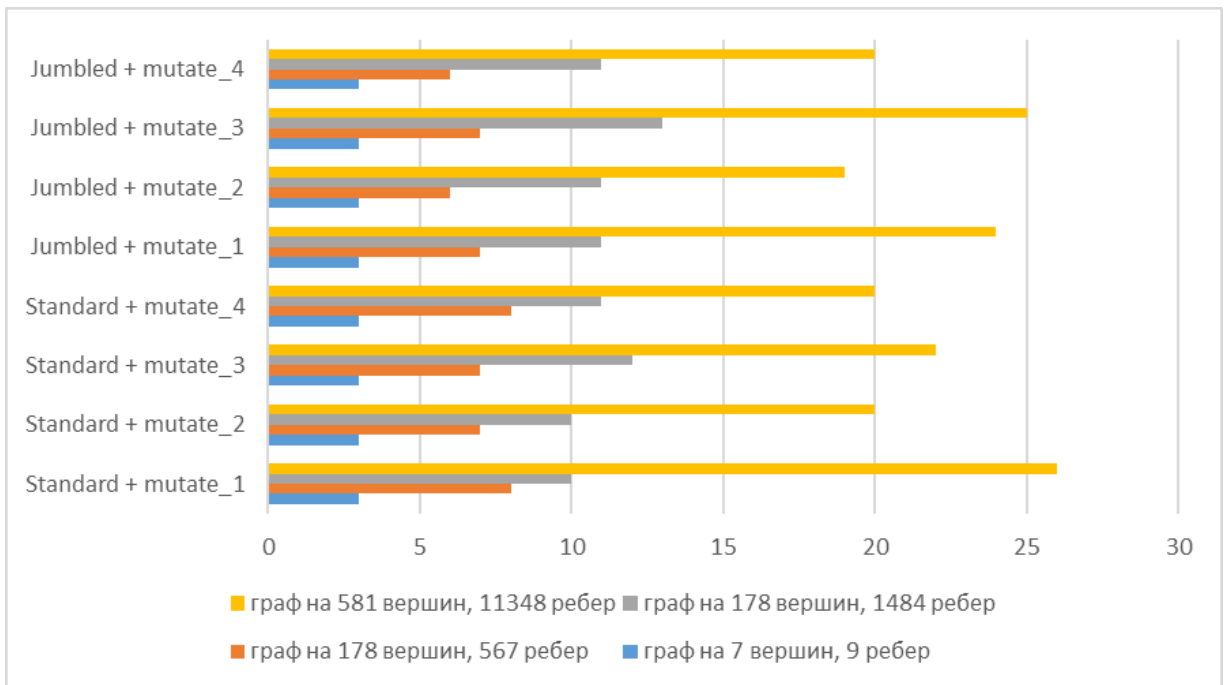
Таблиця 4.4.2. – Час роботи МГА для графа на 581 вершин та 11348 ребер.

	mutate_0	mutate_1	mutate_2	mutate_3	mutate_4
Standard	None	26	20	22	20
Jumbled	None	24	19	25	20

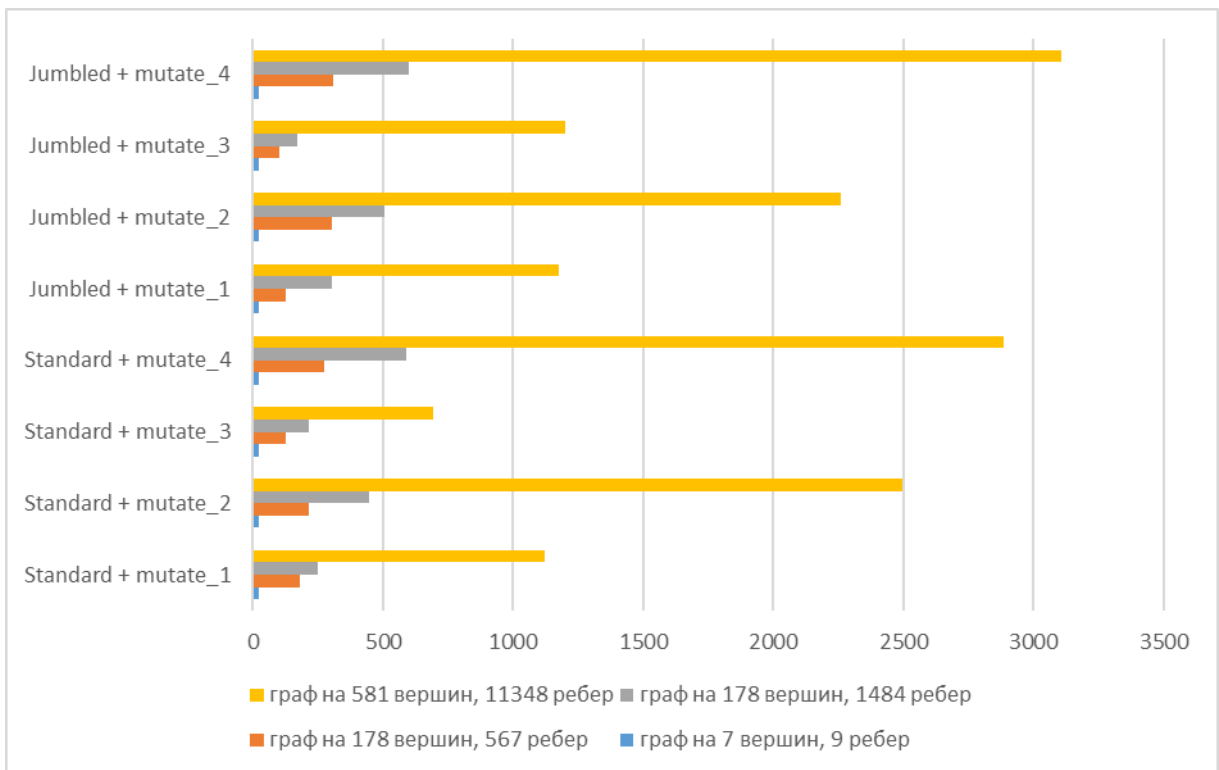
Таблиця 4.4.3. – Хроматичне число МГА для графа на 581 вершин та 11348 ребер хроматичне число.

З таблиць вище впливає:

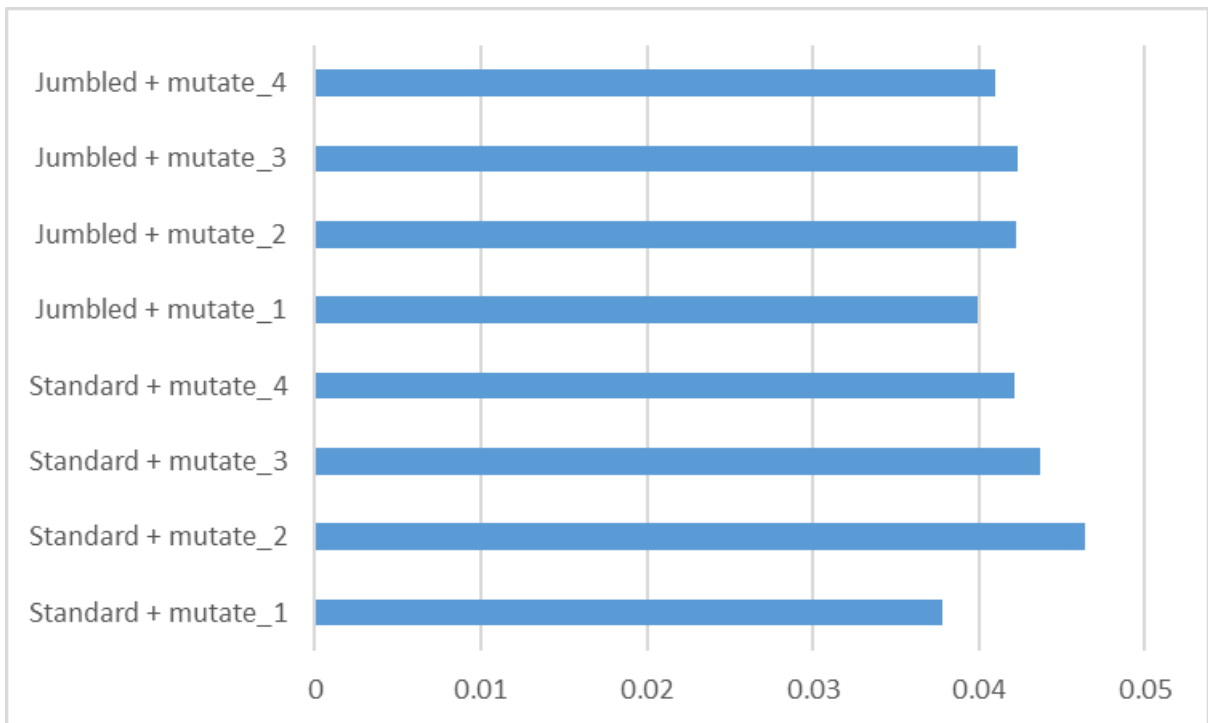
- без ГО мутації ХЧ знаходиться тільки для самих малих графів;
- ФМ mutate_0 працює гірше всіх інших. При чому вона гірше поєднується з Jumbled кросовером. Зважаючи на специфіки цих ГО, можна припустити, що це відбувається через випадковість результатів функцій. Чим більший граф, тим вища ймовірність на неоптимальний результат;
- Найкращою виявилася комбінація Jumbled-кросинговеру та mutate_2, приблизно такою самою за результативністю є комбінація Jumbled-кросинговеру та mutate_4. Що цікаво, тому що в обидвох цих мутаціях вершини фарбуються в випадковий колір.
- За часом роботи та mutate_2 випереджає mutate_4, але видно це тільки на великих графах. На малих розриву немає;
- За кількістю ітерацій mutate_2 також випереджає mutate_4;
- Загалом Jumbled-кросовер працює ефективніше, розподіл результатів тут більш значний, велика залежність від ФМ;
- Найшвидшою є mutate_3 і працює вона краще за mutate_1, що виявилася неоптимальною, хоча раціональних обчислень в ній найбільше.



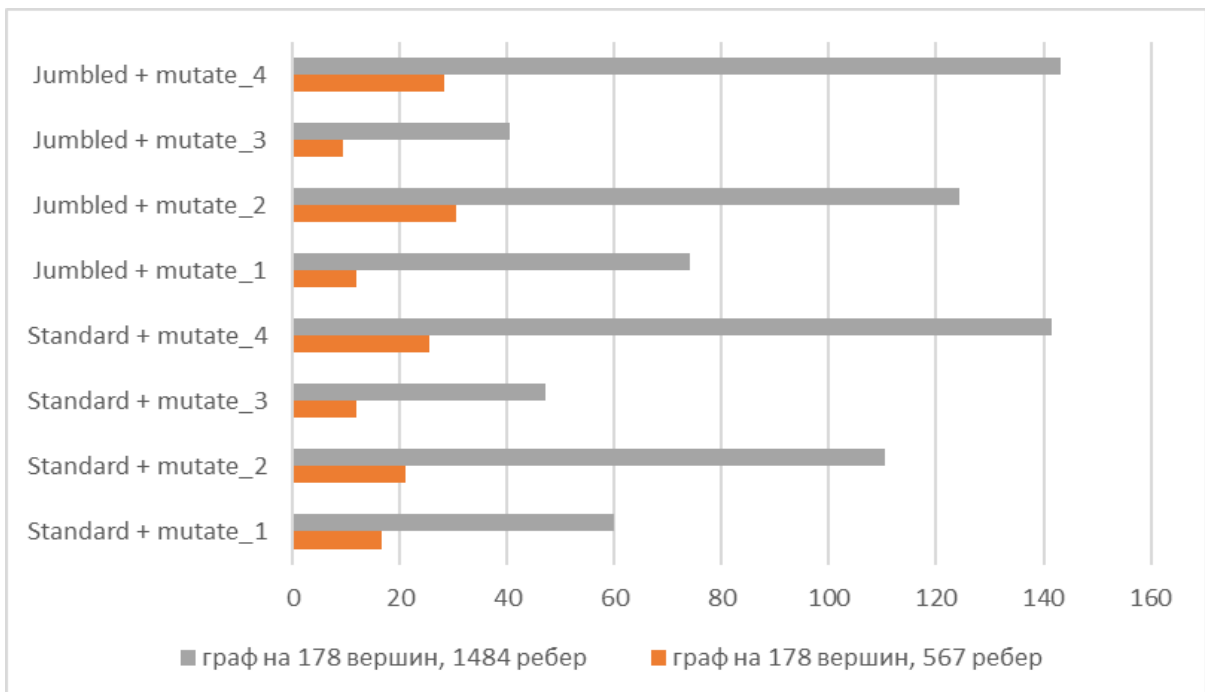
Графік 4.1. – Залежність хроматичного числа МГА від ГО



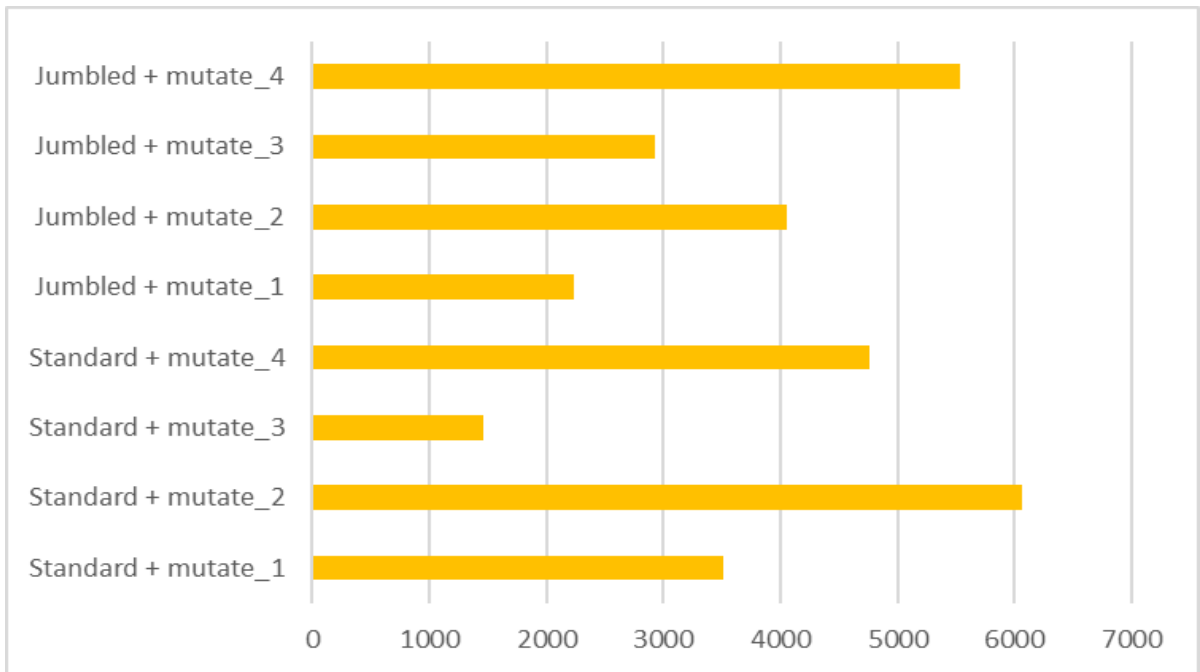
Графік 4.2. – Залежність кількості ітерацій МГА від ГО



Графік 4.3. – Залежність часу роботи МГА від ГО в графі на 7 вершин та 9 ребер



Графік 4.4. – Залежність часу роботи МГА від ГО в середніх за розміром графах



Графік 4.5. – Залежність часу роботи МГА від ГО в графы на 581 вершину та 11348 ребер

Зм	Лист	№ докум.	Підп.	Дата

ВИСНОВКИ

Результатом виконання дипломного проекту є готовий до використання та вбудовування в інші програми модифікований генетичний алгоритм визначення хроматичного числа графа з можливостями налаштування параметрів та вибору найбільш придатних для конкретного випадку генетичних операторів. Як вхідна інформація використовуються випадкові графи, але зберігається можливість тестування будь-яких інших графів.

Алгоритм був реалізований мовою Python, без використання сторонніх бібліотек. Об'єктно-орієнтована структура та наявність коментарів дозволяють легко розширити функціонал за необхідністю.

В ході виконання проекту було розглянуто та проаналізовано безліч наукових публікацій та описаних там способів розв'язку схожих задач, протестовано різні комбінації генетичних операторів, серед яких дві функції кросоверу та п'ять мутацій за такими критеріями як загальна кількість ітерацій, час роботи, величина хроматичного числа.

Описано переваги елітарної селекції та впровадженої в програмі фітнес функції. Показано механізм створення початкової популяції та генерування наступних поколінь, роботу термінального критерія алгоритму. Підібрано оптимальні пропорції параметрів алгоритму, таких як штрафи за однаковий колір суміжних вершин графа та кількість використовуваних кольорів у розфарбовуванні, кількість батьків, на яких застосовуються генетичні оператори та які беруть участь у створенні наступних популяцій алгоритму.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Molloy, M. Graph Colouring and the Probabilistic Method [Text] / M. Molloy, B. Reed - Springer-Verlag Berlin Heidelberg, 2002 – 326 p.
2. Kenneth, A. Every Planar Map is Four Colorable. I. Discharging [Text] / A. Kenneth, W. Haken // Illinois Journal of Mathematics, 21 (3) – 1977 - 429–490 pp.
3. Schrijver, A. On the History of Combinatorial Optimization (Till 1960) [Text] / A. Schrijver // Handbooks in Operations Research and Management Science. – 2001. - #12
4. Karp, R. M. Reducibility Among Combinatorial Problems [Text] / R. M. Karp // Complexity of Computer Computations. New York: Plenum. – 1972. - 85–103 pp.
5. Euler, L. Solution d’une question curieuse que ne paroît soumise à aucune analyse [Text] / L. Euler // Mémoires de l’académie des sciences de Berlin, 15. - (1759) 1766. - 310—337 pp.
6. Euler, L. Solutio problematis ad geometriam situs pertinentis [Text] / L. Euler - Comment. Acad. Sci. U. Petrop 8, 1736 – 128 p.
7. Hassler, W. Congruent Graphs and the Connectivity of Graphs [Text] / W. Hassler // American Journal of Mathematics. 54 (1) – 1932. – 150 - 168 pp.
8. Seth, P. An optimal minimum spanning tree algorithm [Text] / P. Seth, R. Vijaya // Journal of the Association for Computing Machinery, 49 (1). – 2002. - 16–34 pp.
9. Bollobas, B. Modern Graph Theory [Text] / New York: Springer-Verlag - 1998. —184 p.
10. Silvano, M. Knapsack problems / M. Silvano, P. Toth. — Great Britain: Wiley. - 1990. — 306 p.

11. Райгородский А. М. Модели случайных графов и их применения [Текст] // ТРУДЫ МФТИ. — 2010. — Том 2, № 4 – 130-141 сс.
12. Hindi, Musa M. Genetic Algorithm Applied to the Graph Coloring Problem [Electronic resource] // Musa M. Hindi, Roman Yampolskiy pp. 1–7. - Mode of access: https://www.researchgate.net/publication/256169514_Genetic_Algorithm_Applied_to_the_Graph_Coloring_Problem - Last access: 2019.
13. URL: <http://dimacs.rutgers.edu/>
14. Shen, J. W. Solving the Graph Coloring Problem using Genetic Programming [Electronic resource] // Justine W. Shen pp. 1-10. - Mode of access: <http://www.genetic-programming.org/sp2003/Shen.pdf> - Last access: 2019.
15. URL: <https://cs.gmu.edu/~eclab/projects/ecj/>
16. Glass, C. A. Genetic Algorithm for Graph Coloring: Exploration of Galinier and Hao's Algorithm [Electronic resource] // Celia A. Glass, Adam Prugel-Bennett pp. 229-236. - Mode of access: <https://pdfs.semanticscholar.org/aa33/cc4018ae09b1401ffea056528aeab8b64a69.pdf> - Last access: 2019.
17. Galinier P. Hybrid Evolutionary Algorithms for Graph Coloring [Electronic resource] // Philippe Galinier, Jin-Kao Hao pp. 379–397. - Mode of access: <https://link.springer.com/article/10.1023/A%3A1009823419804> - Last access: 2019.
18. URL: <https://mat.tepper.cmu.edu/COLOR/instances.html#XXLEI>
19. Harmanani H. A Method for the Minimum Coloring Problem Using Genetic Algorithms [Electronic resource] // Haidar Harmanani, Hani Abas pp. 487-492. - Mode of access: https://www.researchgate.net/publication/258452604_A_Method_for_the_Minimum_Coloring_Problem_Using_Genetic_Algorithms - Last access: 2019.
20. Kokosinski Z. Parallel Genetic Algorithm for Graph Coloring Problem

- [Electronic resource] // Zbigniew Kokosinski, Marcin Kolodziej, Krzysztof Kwarciany pp. 215-222. - Mode of access: https://link.springer.com/chapter/10.1007/978-3-540-24685-5_27 - Last access: 2019.
21. Chen B. A Fast Parallel Genetic Algorithm for Graph Coloring Problem Based on CUDA [Electronic resource] // Buhua Chen, Bo Chen, Hongwei Liu, Xuefeng Zhang - Mode of access: <https://ieeexplore.ieee.org/document/7307802> - Last access: 2019.
22. Cara, J. J. M. Heuristic Algorithms for NP-complete Problems [Electronic resource] // Juan Jose Martin Cara Mode of access: <https://github.com/JuanjoMrt/GraphColoring/blob/master/resources/Report.pdf> - Last access: 2019.
23. Deljouyi, A. Graph Coloring [Electronic resource] // Amir Deljouyi - Mode of access: <https://github.com/amirdeljouyi/graph-coloring> - Last access: 2019.
24. Holton, D. A. The Petersen graph [Text] / D. A. Holton, J. Sheehan - Cambridge University Press, 1993 – 127 p.
25. Zarie, A. M. Using Genetic Algorithms To Solve The Graph Coloring Problem [Electronic resource] // Ahmed M. Zarie - Mode of access: <https://github.com/zare3/Graph-Coloring-Using-Genetic-Algorithms> - Last access: 2019.
26. Райгородский, А. М. Модели случайных графов [Текст] / МЦНМО. — 2011. — 135 с.
27. Капітонова, Ю. В. Основи дискретної математики [Текст] / Ю. В. Капітонова, С. Л. Кривий, О. А. Летичевський, Г. М. Луцький, М. К. Печурін. - Київ Наукова думка. — 2002. — 581 с.
28. Qu, G. Analysis of watermarking techniques for graph coloring problem [Text] / Gang Qu, Miodrag Potkonjak, ICCAD. – 1998. - 190-193 pp.
29. Кононюк, А. Е. Дискретно-непрерывная математика. Алгоритмы.

Генетические алгоритмы [Текст] / А. Е. Кононюк. - Киев Освіта України. — 2017. — 443 с.

30. Kaya, Y. A Novel Crossover Operator for Genetic Algorithms: Ring Crossover [Electronic resource] // Yilmaz Kaya, Murat Uyar, Ramazan Tekin - Mode of access: https://www.researchgate.net/publication/220485962_A_Novel_Crossover_Operator_for_Genetic_Algorithms_Ring_Crossover - Last access: 2019.
31. Курейчик, В.В. Генетические алгоритмы [Текст] / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик. - Москва ФИЗМАТЛИТ. — 2006. — 320 с.

					ІАЛЦ.045420.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		73