НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: Еволюційні алгоритми глобальної пошукової оптимізації

Виконав (-ла): студент (-ка) __4__ курсу, групи __ІП-73__
(шифр групи)

_____Герега Богдан Дмитрович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н. Волокита А. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль)

_____професор, доктор технічних наук, Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Гереги Богдана Дмитровича

1. Тема проєкту *Еволюційні алгоритми глобальної пошукової оптимізації*
керівник проєкту Волокита Артем Миколайович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2021 року №1139-с
2. Термін здачі студентом закінченого проєкту _____*20 травня 2021 р.*
3. Вихідні дані до проєкту *технічна документація. теоретичні та статистичні дані.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд еволюційних алгоритмів для вирішення задачі глобальної оптимізації

Розділ 2. Огляд алгоритмів та технологій для розробки системи

Розділ 3. Деталі розробки системи

Розділ 4. Дослідження та аналіз розробленої системи

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1	доцент, к.т.н. Волокита А. М.		
Розділ 2	доцент, к.т.н. Волокита А. М.		
Розділ 3	доцент, к.т.н. Волокита А. М.		
Розділ 4	доцент, к.т.н. Волокита А. М.		

7. Дата видачі завдання _____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2020-15.12.2020*	
2.	*Вивчення та аналіз завдання*	*15.12.2020-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*16.06.2021*	

Студент-дипломник _____ Богдан ГЕРЕГА
(підпис)

Керівник проєкту _____ Артем ВОЛОКИТА
(підпис)

АНОТАЦІЯ

У даній роботі було детально розглянуто маловідомі еволюційні алгоритми, схеми їх робіт та основні принципи роботи. Були проаналізовані їхні слабкі та сильні сторони. На основі аналізу було вибрано два кращих алгоритми, які лягли в основу вирішення задачі глобальної пошукової оптимізації. В результаті роботи було розроблено два еволюційних алгоритми, які вирішують задачу глобальної оптимізації, проведено дослідження ефективності роботи алгоритмів в залежності від настроюваних параметрів, а також було здійснено модифікацію одного з алгоритмів. Розроблена програма дає можливість знаходити глобальні мінімуми багатомірних функцій з різною складністю ландшафтів. Програмний продукт був розроблений на мові C#.

Ключові слова: глобальна оптимізація, задача оптимізації, еволюційні алгоритми, C#.

ANNOTATION

In this project for a Bachelor's Degree, little-known evolutionary algorithms, schemes of their work and basic principles of work were considered in detail. Their strengths and weaknesses were analyzed. Based on the analysis, the two best algorithms were selected, which formed the basis for solving the problem of global search engine optimization. As a result of work two evolutionary algorithms which solve a problem of global optimization were developed, research of efficiency of work of algorithms depending on the adjusted parameters was carried out, and a modification of one of the algorithms was performed. The developed program makes it possible to find global minima of multidimensional functions with different complexity of landscapes. The software product was realized in C #.

Key words: global optimization, optimization problem, evolutionary algorithms, C#.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Еволюційні алгоритми	3		
			глобальної пошукової			
			оптимізації			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Еволюційні алгоритми	106		
			глобальної пошукової			
			оптимізації			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Еволюційні алгоритми	1		
			глобальної пошукової			
			оптимізації			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Еволюційні алгоритми	1		
			глобальної пошукової			
			оптимізації			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467200.006 Д3***	Еволюційні алгоритми	1		
			глобальної пошукової			
			оптимізації			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Еволюційні алгоритми	41		
			глобальної пошукової			
			оптимізації			
			Текст програмного коду			

					# *ІАЛЦ.467200.001 ОА*		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Герега Б.Д.			*Еволюційні алгоритми глобальної пошукової оптимізації* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Волокита А.М.					
						НТУУ "КПІ" ФІОТ ІП-73	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЕКТУ

на тему: «*Еволюційні алгоритми глобальної пошукової оптимізації*»

Київ – 2021

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи пошуку глобальних екстремумів на основі еволюційних алгоритмів, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування цієї системи є проектування акустичного обладнання, аналіз даних, економічне та фінансове прогнозування, оцінка та управління екологічним ризиком, дизайн промислового продукту, оптимізація в числовій математиці, оптимальна робота "закритих" (конфіденційних) інженерних або інших систем, для яких поставлена задача глобальної оптимізації.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи пошуку глобальних екстремумів багатомірних функцій із використанням еволюційних алгоритмів, що дозволить ефективно вирішувати дану задачу.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Надати можливість користувачам передавати вхідні дані для пошуку екстремумів та отримувати у відповідь результат.
- Надати можливість користувачам власноруч конфігурувати специфічні параметри обчислення для деяких алгоритмів.
- Надати можливість користувачам власноруч вибирати бажаний алгоритм для обчислення.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Visual Studio 2017 IDE версії або вище.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-15.12.2021
Вивчення та аналіз завдання	15.12.2021-15.03.2021
Розробка архітектури та загальної структури системи	15.03.2021-25.03.2021
Розробка структур окремих частин системи	25.03.2021-5.04.2021
Програмна реалізація системи	5.04.2021-15.04.2021
Виправлення помилок	15.04.2021-20.05.2021
Оформлення пояснювальної записки	25.04.2021
Передзахист	23.05.2021
Захист	16.06.2021

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Еволюційні алгоритми глобальної пошукової оптимізації*»

Київ – 2021

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

BI	(Bat-Inspired) Алгоритм, інспірований кажанами
CS	(Cuckoo Search) Алгоритм зозулі
FSS	(Fish School Search) Алгоритм пошуку косяком риб
GA	(Genetic Algorithm) Генетичний алгоритм
GSO	(Glowworm Swarm Optimization) Алгоритм оптимізації роєм світлячків
IWO	(Invasive Weed Optimization) Алгоритм бур’янистої оптимізації
PYPL	(PopularitY of Programming Language) Популярність мови програмування
TIOBE	(The Importance of Being Earnest) Як важливо бути поважним

ВСТУП

Багато завдань, які з’являються в таких фундаментальних науках, як фізика, хімія та молекулярна біологія, а також у багатьох прикладних науках, зводяться до завдання глобальної оптимізації. Властивостями таких завдань часто є:

- нелінійність;
- недиференційованість;
- багатоекстремальність (мультимодальність);
- овражність;
- відсутність аналітичного виразу;
- висока обчислювальна складність оптимізаційних функцій;
- висока розмірність простору пошуку;
- складна топологія областей допустимих значень і т. д.

Для ефективного вирішення заданих завдань у 80-х роках минулого століття почали інтенсивно розширювати еволюційні пошукові алгоритми оптимізації. У цій роботі я розгляну класи таких алгоритмів, які в різних джерелах називають по-різному: поведінковими, інтелектуальними, метаевристичними, інспірованими природою, роєвими, багатоагентними, популяційними та іншими.

Переважна більшість розглянутих алгоритмів представлена в англомовній літературі, у якій прийнятно використовувати термін “алгоритм” замість загальновживаного в українській літературі терміна “метод”.Щоб уникнути можливої неоднозначності при ідентифікації розглянутих в роботі об'єктів, я також використовую останній термін.

Можна запропонувати кілька класифікацій еволюційних алгоритмів. Виділю наступні класи таких алгоритмів:

- еволюційні алгоритми, включаючи генетичні алгоритми;
- популяційні алгоритми, натхненні живою природою;
- алгоритми, натхненні неживою природою;
- алгоритми, інспіровані людським суспільством та інші.

Еволюційні алгоритми, а також такі популяційні алгоритми, натхненні живою природою, як алгоритми рою частинок, колонії мурах, рої медоносних бджіл і алгоритми на основі штучної імунної системи досить добре освітлені в українській літературі.

Дана робота являє собою огляд великої кількості інших еволюційних алгоритмів, які рідко зустрічаються в нашій літературі.

РОЗДІЛ 1. ОГЛЯД ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ ОПТИМІЗАЦІЇ

1.1 Глобальна оптимізація

1.1.1 Основні поняття

У широкому сенсі, глобальна оптимізація - це пошук стану деякої системи (технічної, виробничої, бізнес-системи), який забезпечить її оптимальне функціонування (максимізує прибуток, мінімізує витрати), а також комплекс заходів, спрямованих на досягнення цього стану.

В аналізі даних глобальна оптимізація - це розділ прикладної математики і чисельного аналізу, який займається проблемами пошуку глобальних екстремумів функцій. У більшості випадків вирішується завдання мінімізації, оскільки максимізація еквівалентна пошуку зворотного функціоналу для завдання мінімізації [1].

Оптимізація є процесом знаходження точки, яка мінімізує функцію. Зокрема:

- Локальний мінімум функції є точкою, де значення функції менше, ніж або дорівнює значенню в сусідніх точках, але можливо більше, ніж в віддаленій точці.
- Глобальний мінімум є точкою, де значення функції менше, ніж або дорівнює значенню у всіх інших допустимих точках [2].

Як видно з рис. 1.1. шукаючи глобальний мінімум можна потрапити у “пастку”, тобто “натрапити” на локальний мінімум.

Глобальна оптимізація набагато складніша, ніж локальна, оскільки аналітичні методи неможливо застосувати, а чисельні в більшості випадків призводять до дуже складних рішень.

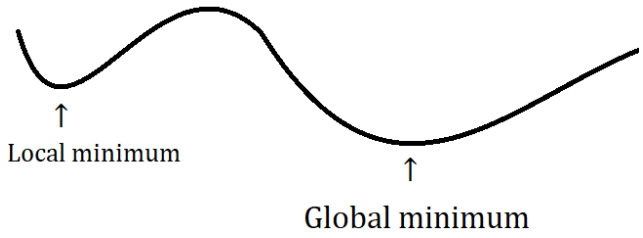


Рисунок. 1.1 – Локальний та глобальний мінімуми

Методи глобальної оптимізації діляться на:

- детерміновані - лінійне і нелінійне програмування, алгебраїчні методи, метод гілок і меж та інші;
- стохастичні - методи Монте-Карло;
- евристичні - алгоритм мурашиної колонії, еволюційні алгоритми, метод рою частинок, метод косяка риб та інші.

Перейдемо до застосування глобальної оптимізації у різних галузях діяльності людей.

Кілька прикладів застосування глобальної оптимізації включають проектування акустичного обладнання, планування терапії раку, моделювання хімічних процесів, аналіз даних, класифікацію та візуалізацію, економічне та фінансове прогнозування, оцінку та управління екологічним ризиком, дизайн промислового продукту, дизайн лазерного обладнання, пристосування моделі до даних (калібрування), оптимізація в числовій математиці, оптимальна робота "закритих" (конфіденційних) інженерних або інших систем, упаковка та інші завдання розміщення об'єктів, моделі потенційних енергій в обчислювальній фізиці та хімії, управління процесами, проектування та маніпуляції роботів, системи нелінійних рівнянь та нерівностей, а також управління системами очищення стічних вод.

Задачі глобальної оптимізації застосовуються в різних областях сучасної науки і техніки. Вони надзвичайно важливі для розвитку економіки в цілому і таких галузей, як нанотехнології, мікроелектроніка, біологія, легка і важка промисловість. У зв'язку з цим розробка ефективних методів вирішення таких завдань і їх реалізація у вигляді прикладних програмних комплексів представляється актуальною науковою і практичною проблемою [3].

1.1.2 Задачі оптимізації та методи їх вирішення

Завдання пошуку глобального мінімуму (максимуму) функції $f(x)$ на допустимій множині $X \subseteq R^n$ полягає у знаходженні такої точки $x_* \in X$, що $f(x_*) \leq f(x)(f(x_*) \geq f(x))$ для всіх $x \in X$. Припустимо, що вирішується завдання мінімізації функції f. Обмеження, пов'язані з обчислювальною похибкою або недоліком ресурсів, часто не дозволяють знайти точне рішення даного завдання. В цьому випадку слід перейти до пошуку наближеного рішення, тобто точки з безлічі ε-оптимальних рішень $X_*^{\varepsilon} = \{ x \in X: f(x) \leq f(x_*) + \varepsilon \}$. Пошук точного рішення можна розглядати як окремий випадок пошуку наближеного рішення з $\varepsilon = 0$ [4].

Спосіб задання множини X дозволяє провести грубу класифікацію задач. Якщо $X = R^n$, то задача відноситься до класу задач безумовної оптимізації. Якщо допустима множина задана за допомогою обмежень, то говорять про умовну оптимізацію. При цьому, якщо X звичайно, розглянута задача відноситься до області дискретної оптимізації.

Можна виділити два основних сімейства методів вирішення завдань кінцевомірної оптимізації - точні і наближені. Точні методи дозволяють гарантувати оптимальність знайденого рішення. До цього класу можна віднести різні варіанти методу гілок і меж, відсікань та інші. Для точних методів характерна висока трудомісткість, яка часто не дозволяє застосовувати їх при

вирішенні реальних завдань. Евристичні методи засновані на припущеннях про властивості оптимального рішення. На відміну від точних евристичні методи не гарантують оптимальність знайденого рішення. Однак в умовах обмеженості обчислювальних ресурсів евристики найчастіше є єдиним способом знаходження рішення. Поширені і гібридні методи, при яких евристичні методи застосовуються для знаходження рішення, а точні - для доказу оптимальності. Ефективність гібридних методів обумовлена тим, що евристичні алгоритми нерідко мають більш високу швидкість збіжності до оптимуму в порівнянні з точними методами [5].

Однією з найбільш поширених схем організації точних методів є так званий метод гілок і меж (МГМ), заснований на розбитті допустимої множини. Вперше метод гілок і меж був запропонований Лендом і Дойгом в 1960 для вирішення загальної задачі цілочислового лінійного програмування. Інтерес до цього методу і фактично його "друге народження" пов'язане з роботою Літтла, Мурті, Суїні і Керела [6], присвяченій задачі комівояжера.

Наведу загальну схему методу. Протягом всього часу роботи підтримується список $\{X_i\}$ підмножин допустимої множини. Спочатку він складається з одного елемента - допустимої множини X. Далі вибирається один з елементів списку - підмножина X_i. Якщо X_i задовольняє правилам відсіву, обраний об'єкт був видалений зі списку. В іншому випадку він піддається дробленню на більш дрібні підмножини за допомогою правила декомпозиції. Отримані підмножини заміщають в списку вибраний елемент. Алгоритм завершує свою роботу, коли в послідовності $\{X_i\}$ не залишається жодного елемента.

Правилом відсіву будемо називати будь-яку функцію $\xi: S \to \{0,1\}$, визначену на деякій множині S підмножин R^n і зіставляє кожній підмножині з S число 0 або 1. Якщо для деякого $V \in S$ виконується $\xi(V) = 1$, то будемо

говорити, що підмножина V задовольняє правилу відсіву ξ. В іншому випадку будемо стверджувати, що V не задовольняє правилу відсіву ξ.

Нехай задані правила відсіву $Y = \{\xi_1, .., \xi_m\}$. Кінцеву сукупність підмножин $C = \{X_1, .., X_t\}$ таку, що і для будь-якого $j = \{1, \dots, t\}$ знайдеться хоча б одне $i = \{1, \dots, m\}$ таке, що $\xi_i(X_j) = 1$, будемо називати покриттям безлічі X для набору правил відсіву Y.

Наведу формулювання одного з найбільш часто вживаних правил відсіву. Для цього знадобляться поняття оцінки та рекорду. Оцінкою називається функція $g: S \to R$, визначена на множині S підмножин R^n, така, що для будь-якого $V \in S$ виконується $g(V) \leq \min\limits_{x \in V \cap X} f(x)$. В якості оцінки, як правило, вибираються легко обчислювані функції. Рекордом називають найменше значення цільової функції, знайдене в процесі вирішення. Якщо $x_1, .., x_k$ - послідовність точок, в яких обчислювалося значення цільової функції, то $f_k = f(x_r) = \min\limits_{i=1,\dots,k} f(x_i)$ - рекорд; x_r - рекордне рішення. Правило відсіву ξ_{rec} по рекорду формулюється так:

$$\xi_{rec} = \begin{cases} 1, & g(V) \geq f_k - \varepsilon \\ 0, & g(V) < f_k - \varepsilon \end{cases}$$

Сукупність правил відсіву повинна вибиратися таким чином, щоб з існування покриття випливало, що знайдений рекорд є ε-оптимальним рішенням.

У процесі знаходження рекорду послідовність точок $x_1, .., x_k$, в яких обчислюється значення цільової функції, формується на основі розбиваючих множин. Наприклад, якщо множини $\{X_i\}$ є паралелепіпедами, то в якості елементів послідовності $x_1, .., x_k$ беруться їх центри. Також можливий підхід, при якому послідовність точок $x_1, .., x_k$ формується незалежним від побудови покриття способом за допомогою різних евристичних процедур. В якості евристик можуть застосовуватися різні локальні алгоритми (метод градієнтного

спуску, Ньютона, перебір в деякій околиці і т.п.), а також більш складні алгоритми, наприклад, стохастичні або генетичні підходи.

Важлива складова методу - правило декомпозиції $\Delta: S \to 2^S$, яке є функцією, визначеною на множині S підмножин R^n і зіставляє деякій підмножині $V \in S$ кінцеву сукупність його підмножин $V_1, .., V_m$, при цьому, $V = \bigcup_{i=1}^{m} V_i$, $V_i \cap V_j = \emptyset$ для всіх $1 \leq i < j \leq m$ [5, 7].

1.2 Огляд еволюційних алгоритмів

Далі будуть розглянуті такі алгоритми, що використовуються для глобальної оптимізації, як генетичний алгоритм, алгоритм світлячків, алгоритм бур'янистої оптимізації, зозулин пошук, алгоритм, інспірований кажанами, алгоритм мавпячого пошуку та алгоритм косяка риб.

1.2.1 Генетичний алгоритм

Генетичний алгоритм - це метод багатовимірної оптимізації, тобто метод пошуку мінімуму багатовимірної функції. Потенційно цей метод можна використовувати для глобальної оптимізації, але з цим виникають складності.

Сама суть методу полягає в тому, що ми модулюючи еволюційний процес: у нас є якась популяція (набір векторів), яка розмножується, на яку впливають мутації і проводиться природний відбір на підставі мінімізації цільової функції. Розглянемо докладніше ці процеси.

Отже, перш за все наша популяція повинна розмножуватися. Основний принцип розмноження - нащадок схожий на своїх батьків. Тобто ми повинні задати якийсь механізм успадкування. І краще буде, якщо він буде включати елемент випадковості. Але швидкість розвитку таких систем дуже низька -

генетична різноманітність падає, популяція вироджується. Тобто значення функції перестає мінімізуватися.

Для вирішення цієї проблеми був введений механізм мутації, який полягає в випадковій зміні якихось особин. Цей механізм дозволяє внести щось нове в генетичну різноманітність.

Наступний важливий механізм - селекція. Як було сказано, селекція - відбір особин (можна тільки з тих, що щойно народилися, а можна з усіх - практика показує, що це не грає вирішальну роль), які краще мінімізують функцію. Зазвичай відбирають стільки особин, скільки було до розмноження, щоб з епохи в епоху у нас була стала кількість особин в популяції. Також прийнято відбирати «щасливчиків» - якесь число особин, які, можливо, погано мінімізують функцію, але зате внесуть різноманітності в наступні покоління.

Цих трьох механізмів найчастіше недостатньо, щоб мінімізувати функцію. Так популяція вироджується - рано чи пізно локальний мінімум забиває своїм значенням всю популяцію. Коли таке відбувається, проводять процес, який називають струсом (в природі аналогії - глобальні катаклізми), коли знищується майже вся популяція, і додаються нові (випадкові) особини [8].

Розглянемо основні відмінності генетичних алгоритмів від традиційних.

1. Генетичні алгоритми працюють з кодами, де представлений набір параметрів, які залежать від аргументів цільової функції. Причім інтерпретація цих кодів здійснюється тільки перед початком роботи алгоритму та по його завершенню. У процесі роботи маніпуляції із кодами відбуваються зовсім незалежно від їх інтерпретації, код розглядається всього лиш навсього як бітовий рядок.

2. Генетичний алгоритм використовує декілька точок пошукового простору водночас, однак не переходить від точки до точки, так як це робиться у деяких традиційних методах. Це дає можливість подолати один з їхніх недоліків - можливість потрапляння в локальний екстремум цільової функції, якщо функція не є унімодальною, тобто має кілька екстремумів.

3. Генетичні алгоритми під час процесу роботи не користуються додатковою інформацією, що підвищує швидкість роботи. Єдина використовувана інформація - область допустимих значень параметрів та цільової функції в довільній точці.

4. Генетичні алгоритми використовують як імовірнісні правила для появи нових точок аналізу, так і детерміновані правила, щоб була можливість переходу від одних точок до інших. Паралельне використання елементів випадковості та детермінованості дає істотно більший ефект, ніж роздільне.

На початку дії генетичного алгоритму формується перше покоління особин. Далі відбувається відбір декількох елітних особин (якщо включена відповідна стратегія) за допомогою обчислення і порівняння значень цільової функції для кожної особини. Елітними є ті особини, у яких отримані значення (наприклад, довжина шляху комівояжера) є найменшими.

Існують три види схрещування: одноточковий кросовер і два варіанти двохточкового кросовера. Відбір перехресних пар здійснюється за допомогою генератора випадкових чисел. На наступному етапі особини нового покоління з певною ймовірністю піддаються мутації. При процедурі редукції розширена популяція скорочується до вихідного розміру за рахунок видалення з неї особин з найгіршими значеннями цільової функції. Закінчення процесу еволюції відбувається, якщо краще значення цільової функції не змінюється протягом заданого числа поколінь.

1.2.2 Алгоритм світлячків

У світі налічується близько двох тисяч видів світлячків, більшість з яких мають здатність світитися, виробляючи короткі і ритмічні спалахи. Спалахи світла виробляються за допомогою процесу біолюмінесценції. Вважається, що основними функціями таких спалахів є залучення осіб протилежної статі і потенційних жертв. Крім того, сигнальні спалаху можуть служити захисним

механізмом попередження потенційних хижаків про те, що світлячок гіркий на смак. Деякі тропічні світлячки можуть синхронізувати свої мерехтіння, демонструючи тим самим приклад біологічної самоорганізації. Інтенсивність світла, як функція відстані від його джерела, підкоряється закону зворотних квадратів. Крім того, інтенсивність світла падає зі збільшенням відстані r внаслідок поглинання світла повітрям. Комбінація цих двох факторів визначає відстань, на якій світлячки бачать один одного, і вона рівна вночі декільком сотням метрів. Відомі два варіанти популяційних алгоритмів оптимізації, інспірованих поведінкою світлячків, - алгоритм світлячків (Firefly algorithm) і алгоритм оптимізації роєм світлячків (Glowworm Swarm Optimization, GSO). Основна відмінність між firefly і glowworm світлячками полягає в тому, що другі є безкрилими.

Алгоритм світлячків (F-алгоритм) запропонований в Кембриджському університеті (Великобританія) в 2007 році Янгом. Алгоритм використовує наступну модель поведінки світлячків [10]:

- всі світлячки можуть залучати один одного, незалежно від своєї статі;
- привабливість світлячка для інших особин пропорційна його яскравості;
- менш привабливі світлячки переміщаються в напрямку більш привабливого світлячка;
- яскравість випромінювання даного світлячка, видима іншим світлячком, зменшується зі збільшенням відстані між світлячками;
- якщо світлячок не бачить біля себе світлячка яскравішого, ніж він сам, тоді він переміщається випадковим чином.

Алгоритм світлячків належить до тих евристичних алгоритмів, які мають різні застосування. Його нескладні та легкі кроки своєю ефективністю приваблюють дослідників з різних дисциплін.

1.2.3 Алгоритм бур’янистої оптимізації

Алгоритм бур’янистої оптимізації (Invasive Weed Optimization, IWO) натхненний таким загальнопоширеним явищем, як колонізація сільськогосподарських угідь бур’янами. Алгоритм запропонований в 2006 році іранськими вченими Мехрабіаном (AR Mehrablan) і Лукасом (C. Lucas) і заснований на моделюванні таких властивостей, як посів, зростання і конкуренція в колонії бур’яну.

Говорячи простою мовою, бур’ян - це будь-яка рослина, що росте там, де вона небажана. У загальному випадку, будь-яка рослина може бути розцінена як бур’ян. Однак зазвичай термін використовують стосовно до тих рослин, чиї експансіоністські властивості представляють серйозну загрозу для культурних рослин. В силу високої актуальності боротьби з бур’янами в світі видається значна кількість журналів, повністю присвячених систематиці бур’янів, їх екології і фізіології, методам контролю і так далі. Найцікавішою особливістю проблеми бур’янів є загальне переконання, що вони завжди перемагають, і чим більше люди намагаються їм противитись, тим сильніше вони розмножуються [11].

Просторові можливості для поширення бур’янів створюють системи землеробства людини. Бур’яни пробираються в ці простори за допомогою спочатку розсіювання насіння, потім колонізації і, нарешті, окупації полів. Біорізноманіття бур’янів, а також їх висока адаптованість до місцевих умов забезпечують високу ефективність зазначених процесів. Залежно від виду бур’ян може відтворюватися з використанням або без використання статевих клітин. Статеве розмноження здійснюється за допомогою насіння, які формуються в материнській рослині після того, як яйцеклітина запліднюється пилком. Потім це насіння поширюються за допомогою вітру, води, тварин і так далі (просторове розсіювання). Якщо насіння потрапляє в комфортні для життя

умови, воно проростає, виростає у дорослу рослини, потім квітне і виробляє насіння.

Основним механізмом, що визначає динаміку спільноти будь-яких рослин, є природний відбір, з якого виділяють два крайніх типи: r-відбір і К-відбір. Реальні стратегії відбору лежать між цими граничними типами. Можна сказати, що девізом r-відбору є живи швидко, розмножуйся швидко, вмирай молодим (live fast, reproduce quick, die young). Даний тип відбору необхідний для успіху в нестабільному, непередбачуваному навколишньому середовищі. При r-відборі кращі такі якості, як висока плодючість, маленький розмір насіння і пристосованість до розсіювання їх на велику відстань. К-відбір використовує принцип живи повільно, розмножуйся повільно, вмирай в старості (live slow, reproduce slow, die old). Цей тип відбору необхідний для успіху в стабільному, передбачуваному навколишньому середовищі, коли ймовірно важке суперництво за обмежені ресурси між конкурентоспроможними індивідуумами. Ситуація має місце, якщо розмір популяції в ареалі проживання близький до максимуму, який він здатний вмістити.

Розглянемо задачу глобальної безумовної максимізації фітнес-функції. В алгоритмі IWO модель поведінки бур'янів при колонізації враховує такі базові властивості процесу.

1. Розподіл кінцевого числа насіння по всій області пошуку (ініціалізація популяції).
2. Виробництво насіння в залежності від пристосованості рослин (відтворення).
3. Розподіл виробленого насіння у випадковому порядку по області пошуку (просторовий розподіл).
4. Повторення кроків 2, 3 до тих пір, поки не буде досягнутий заданий максимум числа рослин.
5. Відбір рослин з більш високою пристосованістю, їх відтворення і просторовий розподіл (конкурентний виняток).

6. Повторення кроку 5 до виконання умови закінчення процесу.

Таким чином, рослини і їх нащадки оцінюються разом, і тим з них, які мають кращу пристосованістю, дозволяють розмножуватися. Даний механізм дозволяє рослинам з меншою пристосованістю відтворюватися, і, якщо їх нащадки мають гарну пристосованістю, вони можуть вижити [12].

1.2.4 Зозулин пошук

Зозулин пошук (Cuckoo Search) - алгоритм натхненний поведінкою зозуль, які підкладають свої яйця в гнізда інших птахів.

В алгоритмі CS кожне яйце в гнізді є рішенням, а яйце зозулі - нове рішення. Мета полягає у використанні нових та потенційно кращих (зозулиних) рішень, щоб замінити менш гарне рішення в гніздах. В найпростішому варіанті даного алгоритму в кожному гнізді знаходиться по одному яйцю.

Припустимо, що мова йде про завдання глобальної безумовної оптимізації, в якій фітнес-функція підлягає максимізації. Алгоритм заснований на трьох наступних правилах [13]:

- кожна зозуля відкладає одне яйце за один раз у випадково обране гніздо;
- кращі гнізда з яйцями високої якості (високим значенням придатності) переходять в наступне покоління;
- яйце зозулі, відкладене у гніздо, може бути виявлено господарем з певною ймовірністю $\xi_a \in (0; 1)$ і видалено з гнізда.

Схема алгоритму CS:

1. Ініціалізуємо популяцію $S = s_i, i \in [1: |S|]$ з $|S|$ хазяйських гнізд і зозулю, тобто визначаємо вектори X_0^i і вектор початкового положення зозулі X_c.
2. Виконуючи польоти Леві, знаходимо нове положення зозулі

$$X_c' = X_c + V \otimes L_{|X|}(\lambda),$$

де $V = (v_j, j \in [1:|X|])$ – вектор розміру кроків по відповідним компонентам вектора X;

$L_{|X|}(\lambda) - (|X| \times 1)$ – випадковий вектор незалежних дійсних випадкових чисел, розподілених за законом Леві [14].

1. Випадковим чином вибираємо гніздо s_j, $j \in [1:|S|]$ і, якщо $\varphi(X_c) > \varphi(X_j)$, замінюємо яйце в цьому гнізді на яйце зозулі, тобто вважаємо $X_j = X_c$.
2. З ймовірністю ξ_a видаляємо з популяції певну кількість гірших випадково обраних гнізд (включаючи, можливо, гніздо s_j) і за правилами кроку 1 будуємо таке ж число нових гнізд.

Початкові положення гнізд і зозулі приймаємо випадковими, рівномірно розподіленими в деякому гіперпаралелепіпеді. Зазвичай вважають всі компоненти вектора V однаковими і рівними v, де величина v пов'язана з масштабами області пошуку. Основними вільними параметрами алгоритму є константи, що визначають траєкторію польотів Леві, і ймовірність видалення гнізд ξ_a [15].

1.2.5 Алгоритм, інспірований кажанами

Алгоритм, інспірований кажанами (Bat-Inspired, BI), запропонований Янгом в 2010 році. Алгоритм може здатися трохи складнішим, ніж більшість інших алгоритмів ройового інтелекту, а також еволюційних алгоритмів, проте він може бути доволі ефективно застосований до проблем оптимізації і давати хороші результати, витрачаючи меншу кількість часу [16].

Більшість видів кажанів має досконалі засоби ехолокації, які вони використовують для виявлення здобичі та перешкод, а також для забезпечення можливості орієнтування в темряві. Для локалізації рухомих об'єктів миші

використовують ефект Доплера [17]. Алгоритм BI передбачає таку модель поведінки кажанів.

1. За допомогою ехолокації всі миші можуть вимірювати відстань до здобичі і перешкод, а також розрізняти їх.
2. Миші рухаються випадковим чином. Поточні положення і швидкість миші S_i рівні X_i і V відповідно. Для пошуку здобичі миші можуть змінювати частоту сигналів, і також частоту повторення випромінюваних імпульсів (rate of pulse).
3. Частота сигналів може змінюватися в діапазоні $[\omega^{min}, \omega^{max}]$, $\omega^{max} > \omega^{min} \geq 0$, а гучність сигналів - в межах від 0 до 1.

Припустимо, що мова йде про завдання глобальної безумовної мінімізації функції. Основні кроки схеми алгоритму BI мають такий вигляд.

1. Ініціалізація популяції. Задаємо початкові положення агентів S_i.
2. Визначаємо глобально кращого агента і відповідне йому рішення X^{**}.
3. Міграція агентів. Виконуємо переміщення усіх агентів на один крок відповідно до використовуваної міграційної процедури.
4. Локальний пошук. З ймовірністю ξ_{ir} (вільний параметр алгоритму) реалізуємо процедуру локального пошуку в околиці кращого рішення X_i^* знайденого агентом S_i за усі попередні ітерації. Приймаємо знайдене рішення у якості нового поточного становища агента S_i.
5. Глобальний пошук. В околиці поточного рішення X, випадковим чином генеруємо рішення X_i'. Якщо $\varphi\ (X_i') < \varphi(X)$, то з ймовірністю ξ_{ia} приймаємо рішення X_i' в якості нового поточного становища агента S_i. Знаходимо нове глобально краще рішення X .
6. Еволюція параметрів ξ_{ir}, ξ_{ia}.

7. Завершення ітерацій. Перевіряємо виконання умов завершення ітерацій. Якщо дані умови виконані, приймаємо у якості рішення поточну точку X^{**} та завершуємо обчислення, в іншому випадку повертаємося до кроку 2.

Ініціалізацію популяції реалізуємо шляхом випадкового рівномірного розподілу агентів S_i в даній області простору пошуку. На цьому етапі задаємо початкові значення частот ω_i, гучностей a_i і частот повторення імпульсів r_i, рівномірно випадково розподіляючи їх у відповідних інтервалах $[\omega^{min}, \omega^{max}]$, $[a^{min}, a^{max}]$, $[0;\ 1]$.

Міграція агентів. Міграцію агента s_i здійснюємо за формулами:

$$X_i' = X_i + V_i',$$

$$V_i' = V_i + \overline{w_i'}(X_i - X^{**}),$$

$$\omega_i = \omega^{min} + (\omega^{max} - \omega^{min}) \bigcup\nolimits_1 (0; 1)$$

Іншими словами, відповідно до міграційної процедури агент переміщається в напрямку, який визначається сумою вектора переміщення на попередній ітерації (доданок V_i) і випадковим чином збуреного вектора напрямку на кращого агента $(X_i - X^{**})$.

Локальний пошук виконуємо за наступною схемою.

Випадковим чином варіюємо поточний стан агента S_i відповідно до формули.

$$X_i' = X_i + \bar{a} \cup_{|x|}(-1,1),\ i \in [1{:}\ |S|],$$

де $\bar{a}$ – поточне середнє значення гучності всіх агентів популяції:

$$\bar{a} = \frac{1}{|S|}\sum_{i=1}^{|S|} a_i$$

Обчислюємо значення функції в новій точці. Якщо воно більше, то завершуємо процедуру локального пошуку, в іншому випадку повертаємося до кроку 1.

Авторами алгоритму виконано широке експериментальне порівняння ефективності алгоритму BI з генетичним алгоритмом і алгоритмом рою частинок. Показано, що з точки зору ймовірності локалізації глобального екстремуму алгоритм BI володіє більшою ефективністю, ніж зазначені алгоритми.

1.2.6 Алгоритм мавпячого пошуку

Алгоритм натхненний поведінкою мавпи, яка здатна лазити по деревах у пошуках їжі. Мавпі ставиться у відповідність агент, який обстежує поверхні фітнес функції з метою пошуку екстремуму в задачі глобальної максимізації. Вважається, що мавпи керуються тим, що чим вище гора, тим більше буде їжі на її вершині.

Кожна мавпа рухається вгору зі свого поточного положення, поки не досягне вершини гори. Потім мавпа здійснює серію локальних стрибків у довільному напрямку з надією знайти більш високу гору, та рух вгору далі повторюється.

Після виконання фіксованого числа підйомів та локальних стрибків мавпа вважає, що у достатній мірі досліджувала ландшафт у околиці свого початкового положення. Щоб обстежити нову область простору пошуку, мавпа здійснює довгий глобальний стрибок.

Вказані вище дії повторюються задану кількість разів. Рішенням задачі оголошується найвища з вершин, знайдених даною популяцією мавп.

Схему М-алгоритму можна представити в наступному вигляді [18]:

Ініціалізацію популяції мавп $s_i, i \in [1:|S|]$ виконують за загальними правилами, а саме шляхом рівномірного випадкового розподілу агентів у пошуковому просторі.

Процес руху вгору (climb process) являє собою процес локального пошуку та може бути реалізований багатьма способами. В оригінальному алгоритмі

автори пропонують використовувати алгоритм на основі процедури стохастичної апроксимації [19].

Локальні стрибки (watch-jump process). Схема процесу локальних стрибків для агента $s_i, i \in [1:|S|]$ має наступний вигляд:

1. Виходячи з поточного стану агента X_i, генеруємо його нове можливе положення X_i' за формулою:

$$x'_{i,j} = U_1\left(\left(x_{i,j} - b\right); \left(x_{i,j} + b\right)\right), i \in [1:|S|], j \in [1:|X|],$$

тобто вважаємо, що по кожній з координат нове положення агента являє собою випадкову величину, рівномірно розподілену в інтервалі $[\left(x_{i,j} - b\right); \left(x_{i,j} + b\right)]$. Тут $b > 0$ - максимально можлива довжина стрибка (вільний параметр алгоритму).

2. Якщо точка X_i' є допустимою (належить області допустимих значень D) і $\varphi\ (X_i') \geq \varphi(X)$, то вважаємо $X_i = X_i'$ і завершуємо для даного агента локальні стрибки, в іншому випадку задане число раз повертаємося до кроку 1.

Глобальні стрибки (somersault process) агенти здійснюють зі свого поточного положення у напрямку поточного центру ваги їх координат за наступною схемою.

1. Генеруємо випадкове дійсне число $v = U_1(v^-; v^+)$, що має величину кроку глобального стрибка. Тут v^-, v^+ - нижня і верхня межі цієї величини, які мають в загальному випадку різні знаки, так що величина v може бути як позитивною, так і негативною.
2. Знаходимо нове можливе положення X_i' агента s_i' за формулою:

$$x'_{i,j} = x_{i,j} + v\left(x_j^c - x_{i,j}\right), i \in [1:|S|], j \in [1:|X|],$$

де $x_j^c = \frac{1}{|S|}\sum_{i=1}^{|S|} x_{i,j}$ - поточний стан центра тяжіння агентів популяції по j-му координатному напрямку.

3. Якщо положення X_i' є допустимим, то вважаємо $X_i = X_i'$ і завершуємо для даного агента глобальні стрибки, інакше задане число раз повертаємося до кроку 1.

Відзначимо, що якщо величина кроку глобального стрибка v приймає позитивне значення, то агент здійснює стрибок в сторону центра ваги X^c, а якщо від'ємне значення - в протилежну сторону.

В якості критерію завершення ітерацій використовуємо, як зазначалося вище, досягнення заданого числа повторень цих дій алгоритму.

Оскільки М-алгоритм не дає гарантії, що краще рішення буде знайдено на останній ітерації, кожен раз потрібно зберігати в пам'яті ЕОМ поточне краще рішення [20].

1.2.7 Алгоритм косяка риб

Алгоритм пошуку косяком риб (Fish School Search, FSS) запропонували в 2008 році Філо і Нето. В алгоритмі FSS риби плавають у акваріумі (області пошуку) у пошуках їжі. Вага кожної рибини формалізує її індивідуальний успіх у пошуку рішення та відіграє роль її пам'яті. Якраз наявність ваги в агентів популяції є головною особливістю парадигми FSS. Оператори алгоритму FSS об'єднані в дві групи [21]:

- Оператор годування, який формалізує успішність дослідження рибами тих чи інших областей акваріума;
- Оператори плавання, які реалізують алгоритми міграції агентів.

Оператор годування (feeding operator). Позначимо W_i, поточна маса агента S_i. В алгоритмі FSS прийнято, що вага агента пропорційна нормалізованій різниці значень фітнес-функції на наступній і поточній ітераціях.

Оператори плавання (swimming operators). У алгоритмі FSS виділяють три види плавання - індивідуальне, інстинктивно-колективне та колективно-

вольове. Ці види плавань виконуються на інтервалах $(t, \tau], (\tau, \theta], (\theta, t\,']$ основного ітераційного інтервалу $(t, t']$ відповідно.

Індивідуальне плавання (individual swimming). Напрямок переміщення агента в цьому випадку рівноймовірно випадковий. Якщо це переміщення виводить агента за межі області допустимих значень D, то переміщення не виконуємо. Аналогічно, переміщення агента S_i не проводиться, якщо в новій точці $X_{i\tau}$ значення фітнес-функції не вище її значення в попередній точці X_{it}. Процес індивідуального плавання може включати в себе не одну ітерацію, а деяке їх фіксоване число. Таким чином, індивідуальне плавання агента можна інтерпретувати як локальний пошук в околиці поточного становища агента.

Інстинктивно-колективне плавання (collective-instinct swimming) реалізуємо після завершення усіма агентами індивідуальних плавань по формулі:

$$W_i' = W_i + \frac{\varphi(X_i') - \varphi(X_i)}{max(\varphi(X_i'), \varphi(X_i))}, \qquad i \in [1{:}\,|S|]$$

Другий доданок у формулі є ні що інше, як загальний для всіх агентів крок міграції, який представляє собою зважену суму індивідуальних переміщень агентів. Формула вище означає, що у процесі інстинктивно-колективного плавання на кожного агента впливають усі інші агенти популяції, та цей вплив пропорційний індивідуальним успіхам агентів.

Колективно-вольове плавання (collective volition swimming) виконуємо слідом за інстинктивно-колективним плаванням. Колективно-вольове плавання полягає в зміщенні всіх агентів в напрямку поточного центру жорсткості популяції, якщо сумарна вага косяка в результаті індивідуального і інстинктивно-колективного плавання збільшилась, і в протилежному напрямку - якщо ця вага зменшилася. Іншими словами, в разі успішних в середньому зазначених плавань популяції стягується до свого центру тяжіння, тобто підвищує інтенсивність пошуку. В іншому випадку популяція розширюється від того ж цента, підвищуючи свої диверсифікаційні властивості.

Відомий ряд модифікованих алгоритмів FFS. Як приклад можна розглянути так званий щільний алгоритм FFS (Density FFS, DFFS), який використовує модифікацію операторів класичного алгоритму, а також нові оператори - так звані оператори пам'яті і розбиття. Оператор пам'яті (memory operator) будується на основі - векторів пам'яті M_i, агентів популяції і формалізує поточний ряд попередніх впливів на кожного даного агента інших агентів популяції. Оператор розбиття (partitioning operator) використовує пам'ять агентів і призначений для формування на основі популяції ряду підпопуляцій [22].

1.3 Порівняльний аналіз еволюційних алгоритмів

Далі будуть описані у табл. 1.1. такі переваги та недоліки алгоритмів, що використовуються для глобальної оптимізації, як генетичний алгоритм, алгоритм світлячків, алгоритм бур’янистої оптимізації, зозулин пошук, алгоритм мавпячого пошуку та алгоритм косяка риб.

Таблиця 1.1. – Переваги та недоліки еволюційних алгоритмів

Алгоритм	Переваги	Недоліки
Генетичний алгоритм	1. Легко модифікується 2. Підтримка багатоцільової оптимізації 3. Легко паралелізується 4. Використовують правила імовірнісного переходу 5. Надійний до локальних мінімумів / максимумів	1. Обчислювально дорогий 2. Будь-який невідповідний параметр ускладнить сходження алгоритму або просто дасть безглузді результати

Продовження таблиці 1.1

Алгоритм	Переваги	Недоліки
Алгоритм світлячків	1. Здатність мати справу з мультимодальністю 2. Автоматичний підрозділ (популяції автоматично підрозділяються на підгрупи) 3. Параметри можуть бути налаштовані для управління випадковістю під час ітерацій	1. Висока обчислювальна часова складність 2. Повільна швидкість збіжності
Алгоритм бур'янистої оптимізації	1. Всі можливі кандидати беруть участь у процесі відтворення 2. Алгоритм є простим і включає меншу кількість обчислювального навантаження	1. При збільшенню пошукового простору збільшується час обчислення та кількість змінних, що підлягають налаштуванню 2. Поступове зменшення стандартної дисперсії може призвести до незрілої конвергенції
Зозулин пошук	1. Алгоритм простий в застосуванні 2. Алгоритм залежить від меншої кількості параметрів і є надійним 3. Алгоритм забезпечує кращу продуктивність для локального пошуку	1. Дуже легко потрапляє в місцеві оптимальні рішення 2. Повільна швидкість конвергенції

Продовження таблиці 1.1

Алгоритм	Переваги	Недоліки
Алгоритм, інспірований кажанами	1. Алгоритм простий, гнучкий та простий у реалізації 2. Може забезпечити дуже швидку конвергенцію на самому початковому етапі 3. Алгоритм використовує контроль параметрів, який може змінювати значення параметрів у процесі ітерацій 4. Може ефективно вирішувати масштабні проблеми	1. Неправильне налаштування параметрів може призвести до застою після певної початкової стадії 2. Відсутність хороших дослідницьких здібностей
Алгоритм мавпячого пошуку	1. Алгоритм має кілька параметрів для налаштування, що робить його особливо простим у реалізації. 2. Алгоритм може ефективно уникнути потрапляння в оптимальні локальні рішення 3. Розмір популяції майже не чутливий до виміру проблем	1. Оскільки кількість скелелазінь велика, у процесі підйому витрачається багато часу 2. Можливий невпорядкований напрямок підйому, що веде до невдалого зближення

Кінець таблиці 1.1

Алгоритм	Переваги	Недоліки
Алгоритм косяка риб	1. Прості обчислення у всіх особин 2. Вирішує завдання глобальної оптимізації для функцій зі складним ландшафтом та великим простором пошуку 3. Автономність (тобто здатність до самоконтролю функціонування) 4. Масштабованість (з точки зору складності завдань оптимізації / пошуку)	1. Будь-який невідповідний параметр може призвести до отримання невірного рішення

Як видно із таблиці 1.1 кожен еволюційних алгоритм має свої переваги та недоліки. Проаналізувавши всі позитивні та негативні сторони алгоритмів було прийнято рішення використати два найкращі алгоритми для вирішення задачі глобальної оптимізації.

У наступному розділі буде розглянуто більше детально генетичний алгоритм та алгоритм косяка риб, адже генетичний алгоритм має найбільше переваг у вирішенні задачі пошуку екстремуму, а алгоритм косяка риб має найменше недоліків. Отож, у об’єкті розробки даної роботи візьмуть участь саме ці два алгоритми.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі були розглянуті задачі оптимізації та методи їх вирішення. Існує два основних методи вирішення задачі кінцевомірної оптимізації: точний метод та наближений метод. Точні методи можуть гарантувати оптимальність знайдених рішень. Ця категорія включає різні варіанти методів, таких як метод гілок і меж, відсікань та інші. Точні методи є трудомісткими, і їх, як правило, не використовують для вирішення практичних задач. Евристичні методи базуються на припущеннях про природу найкращого рішення. На відміну від точних методів, еврестичні не можуть гарантувати оптимальність знайдених рішень. Також для вирішення задач оптимізації використовують еволюційні алгоритми, які є дуже ефективними. Адже основні труднощі чисельного рішення задачі оптимізації пов'язані з її розмірністю і видом цільової функції, яка в загальному випадку може бути нелінійною, недиференційованою і багатоекстремальною. У кожному конкретному випадку необхідно підбирати такий метод, який дає найбільш точне рішення, а також стежити потім, щоб процедура пошуку не була надто тривалою. Але трапляється, що традиційні методи і алгоритми оптимізації через складність завдання не дають бажаного результату, і тому виникає необхідність використання інших методів. Одним з таких нових підходів, що дозволяють долати зазначені труднощі, є еволюційно-генетичний підхід. На його основі можна розробляти алгоритми пошуку оптимальних рішень. Особливість подібних алгоритмів полягає в тому, що в них використовуються механізми, подібні механізмам популяційної генетики. Ці алгоритми є надійними та стійкими по відношенню до виду цільової функції, а пошук оптимального рішення в них здійснюється шляхом прямого маніпулювання сукупністю з

декількох допустимих рішень, що утворюють популяцію, кожне з яких представлено в деякому коді.

Також у першому розділі було розглянуто наступні еволюційні алгоритми: генетичний алгоритм, алгоритм світлячків, алгоритм бур'янистої оптимізації, зозулин пошук, алгоритм заснований на поведінці кажанів, алгоритм мавпячого пошуку та алгоритм косяка риб. Були розглянуті схеми їх роботи, а також принцип роботи кожного із цих алгоритмів. Були встановлені їх недоліки та переваги. Зокрема, було встановлено, що алгоритм косяка риб має складний алгоритм роботи, проте вирішує завдання глобальної оптимізації для функцій зі складним ландшафтом та великим простором пошуку та видає при цьому хороші результати.

У об’єкті розробки даної роботи – вирішення задачі глобальної оптимізації було прийнято рішення використовувати два алгоритми. Перший із них – це алгоритм косяка риб, він був вибраний через те, що показує себе найкраще у роботі, тобто вирішує задачу оптимізацію з найоптимальнішим часом. Другий же вибраний алгоритм – це генетичний, який є також досить ефективним, але поступається у часі обчислення, проте видає точний результат. Основна мета роботи – порівняння ефективності роботи маловідомого алгоритму (косяка риб) порівнюючи його з роботою стандартного генетичного алгоритму та оптимізація алгоритму FSS.

РОЗДІЛ 2. ОГЛЯД АЛГОРИТМІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Детальний огляд еволюційних алгоритмів, які лягли в основу об’єкту розробки

Далі буде детально описано роботу алгоритму косяка риб та генетичного алгоритму.

2.1.1 Огляд генетичного алгоритму

Генетичний алгоритм заснований на моделюванні популяції особин протягом великої кількості поколінь. В процесі роботи генетичного алгоритму з'являються нові особини з кращими параметрами, а найменш вдалі особини гинуть. Для визначеності далі перераховані терміни, які використовуються в генетичному алгоритмі.

- особина - одне значення x серед безлічі можливих значень разом зі значенням цільової функції для даної точки x.
- хромосоми - значення x.
- хромосома - значення x і в разі, якщо x є вектором.
- функція пристосованості (фітнес-функція, цільова функція) - оптимізуюча функція $f(x)$.
- популяція - сукупність особин.
- покоління - номер ітерації генетичного алгоритму.

Кожна особина являє собою одне значення x серед безлічі всіх можливих рішень. Значення оптимізуючої функції (в подальшому для стислості будемо

вважати, що ми шукаємо мінімум функції) розраховується для кожного значення x. Будемо вважати, що чим менше значення має цільова функція, тим краще дане рішення [23].

Структурна схема генетичного алгоритму показана на наступному малюнку:

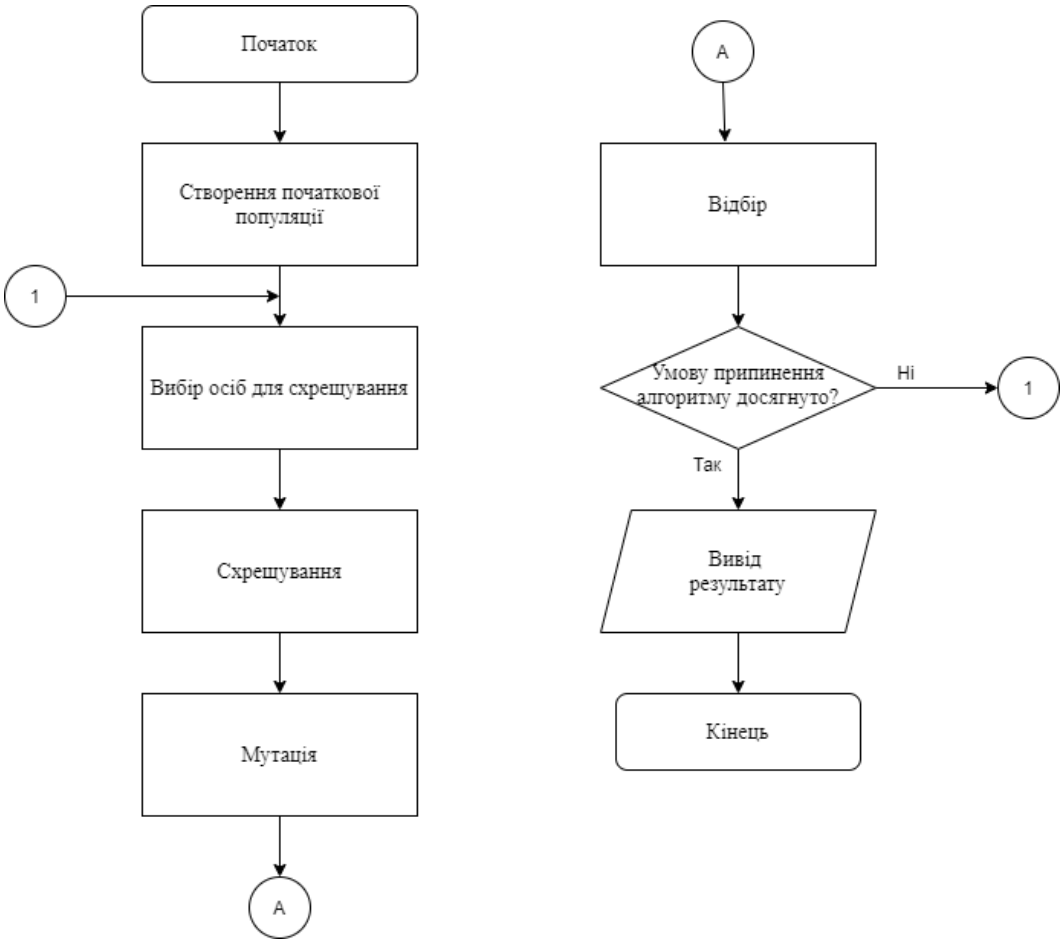


Рисунок 2.1. – Блок-схема генетичного алгоритму

Розглянемо кожен етап алгоритму більш детально [24].

1) Створення початкової популяції

Перший етап роботи алгоритму - це створення початкової популяції, тобто створення безлічі особин з різними значеннями хромосом x. Як правило, початкову популяцію створюють з особин з випадковим значенням хромосом, при цьому намагаються, щоб значення хромосом в популяції покривали всю область пошуку рішення відносно рівномірно, якщо немає якихось припущень щодо того, де може перебувати глобальний екстремум.

Чим більше особин буде створено на даному етапі, тим більша ймовірність того, що алгоритм знайде правильне рішення, а також при збільшенні розміру початкової популяції як правило потрібна менша кількість ітерацій генетичного алгоритму (кількість поколінь). Однак при зростанні розміру популяції потрібна все більша кількість розрахунків цільової функції і виконання інших генетичних операцій на наступних етапах алгоритму. Тобто при збільшенні розміру популяції збільшується час розрахунку одного покоління.

В принципі, розмір популяції не зобов'язаний залишатися незмінним протягом усієї роботи генетичного алгоритму, часто зі збільшенням номера покоління розмір популяції можна зменшувати, тому що згодом все більша кількість особин будуть розташовуватися все ближче до шуканого рішення. Однак часто розмір популяції підтримують приблизно постійним.

2) Вибір особин для схрещування

Після створення популяції потрібно визначити, які особи будуть брати участь в операції схрещування. Для даного етапу існують різні алгоритми. Найпростіший з них - схрещувати кожну особину з усіма, але тоді на наступному етапі доведеться виконувати дуже багато операцій схрещування і розрахунку значень цільової функції. Проте у даній роботі, я визначаю значення імовірності схрещування особин, тобто у схрещуванні візьмуть участь тільки ті особини, які входять в імовірнісне значення, а інші особини позбавляються можливості залишати потомство.

Таким чином, на даному етапі потрібно створити пари особин, для яких на наступному етапі буде проводитися операція схрещування.

Результатом даної операції буде список партнерів для схрещування.

3) Схрещування

Схрещування - це генетична операція, в результаті якої створюються нові особини з новими значеннями хромосом. Нові хромосоми створюються на основі хромосом батьків. Найчастіше в процесі схрещування одного набору

партнерів створюється одна дочірня особина, але теоретично дочірніх особин може створюватися більше. Алгоритм схрещування також можна реалізувати різними способами.

Сенс схрещування полягає в тому, що беремо одну частину хромосоми і другу частину іншого виду і створюємо з них нову хромосому, яка і буде міститися в новому вигляді. Часто хромосоми схрещують побітово. Суть його полягає в тому, що спочатку випадково вибираємо точку розриву (схрещування) хромосоми, потім створюємо нову хромосому, яка складається з лівої частини першої хромосоми і правої частини другої.

Нехай, наприклад, у нас є дві хромосоми (8-бітові для простоти): 10110111 і 01110010. Випадково вибираємо точку розриву (відзначена символом '|'):

101 | 10111

011 | 10010

Звідси ми може зробити дві різні хромосоми це 101 10010 і 011 10111. Яку з них вибрати - це ми вирішуємо генератором випадкових чисел. Також можна шукати дві і більше точок перетину.

У даній роботі я схрещую хромосоми типу Double, тому в даному випадку по суті є дві точки перетину - в середині слова і знак числа, тобто спочатку схрещуються хромосоми побітово без урахування знаку, а потім також випадково беремо знак від однієї з хромосоми.

4) Мутація

Мутація - важливий етап генетичного алгоритму, який підтримує різноманітність хромосом особин і таким чином зменшує шанси на те, що рішення швидко зійдеться до якогось локального мінімуму замість глобального. Мутація являє собою випадкову зміну в особини хромосоми, яка була тільки що створена шляхом схрещування.

Як правило, ймовірність мутації встановлюється не дуже великою, щоб мутація не заважала збіжності алгоритму, інакше вона буде псувати особини з вдалими хромосомами.

Так само як і для схрещування, для мутації можна використовувати різні алгоритми. Наприклад, можна замінювати одну хромосому або кілька хромосом на випадкову величину. У даній роботі використовується побітова мутація, коли в хромосомі інвертується один біт, як показано на Рис 2.2.

Рисунок 2.2 – Мутація одного біта

В результаті мутації особини можуть вийти як більш вдалі, так і менш вдалі, але ця операція дозволяє з ненульовою ймовірністю з'явитися вдалій хромосомі з наборами нулів і одиниць, яких не було у батьківських особин.

5) Відбір

В результаті схрещування і подальшої мутації з'являються нові особини. Якби не існувало етапу відбору, то кількість особин росла б за експоненціальним законом, що вимагало б все більшої кількості оперативної пам'яті і часу на обробку кожного нового покоління популяції. Тому після появи нових особин потрібно очищати популяцію від найменш вдалих особин.

Алгоритми відбору також можуть бути різні. Часто в першу чергу відкидаються особини, чиї хромосоми не потрапляють в заданий інтервал пошуку рішення. У даній реалізації я сортую види за значенням їх цільової функції, тобто залишаються найбільш «живучі» види.

6) Умови закінчення алгоритму

Критерій закінчення алгоритму полягає в заданні певної кількості ітерацій (поколінь). Але цей критерій потрібно використовувати обережно, тому що генетичний алгоритм є імовірнісним, і різні запуски алгоритму можуть сходитися з різною швидкістю. Тому в якості порога номера ітерації потрібно задавати досить велике число.

2.1.2 Огляд модифікованого алгоритму косяка риб

Косяком риб (fish school) називають агрегацію риб, які пересуваються приблизно з однієї і тією ж швидкістю та орієнтацією, підтримуючи приблизно постійну відстань між собою. Доведено, що всякого роду об'єднання риб відіграють важливу роль в підвищенні ефективності знаходження ними їжі, захисту від хижаків, і також у зменшенні енергетичних витрат.

В алгоритмі FSS рибки плавають в акваріумі (області пошуку) у пошуках їжі (вирішення задачі оптимізації). Вага кожної з риб формалізує її індивідуальний успіх у пошуку рішення та відіграє роль її пам'яті. Якраз наявність ваги у агентів популяції є основною особливістю парадигми FSS в порівнянні, наприклад, з парадигмою оптимізації рою частинок. Ця особливість алгоритму FSS дозволяє відмовитись від необхідності відшукувати та фіксувати глобально кращі рішення, як це, наприклад, робиться в алгоритмі рою частинок.

FSS використовує такі принципи [25]:

1. Прості обчислення для всіх особин (наприклад, риб).
2. Різні способи зберігання інформації (наприклад, вага риби)
3. Локальні обчислення (тобто плавання складається з окремих компонентів).
4. Низька комунікація між сусідніми особинами (тобто риби повинні думати про місцеві, але також бути соціально обізнаними).
5. Мінімальний централізований контроль (в основному для самоконтролю радіусу).
6. Деякі виразні механізми різноманітності (щоб уникнути небажаної поведінки зграї).
7. Масштабованість.
8. Автономність (тобто здатність до самоконтролю).

Структурна схема алгоритму FSS показана на наступному рисунку 2.3.

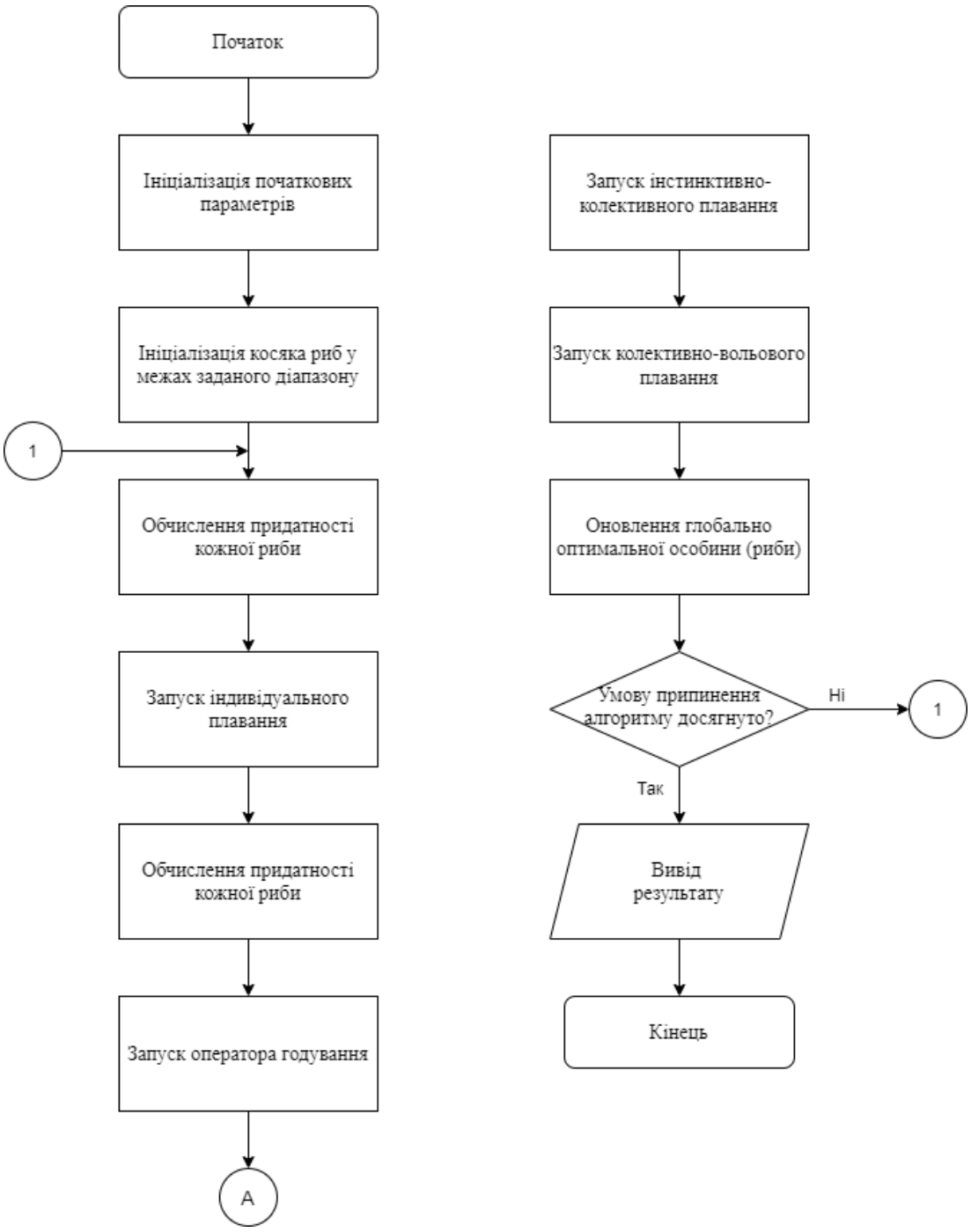


Рисунок 2.3 – Блок-схема алгоритму FSS

FSS - це алгоритм пошуку на основі популяції, заснований на поведінці плаваючих риб, які розширюються і звужуються в пошуках їжі. Кожне місце розташування риби є можливим рішення проблеми оптимізації. Алгоритм використовує вагу для всіх риб, що представляє собою сукупний рахунок того, наскільки успішним був пошук кожної риби в косяку. FSS складається з операторів годування і плавання, які діляться на три підкомпоненти, а саме[26]:

1) Індивідуальне плавання (individual swimming)

Кожна риба в косяку виконує локальний пошук в пошуках перспективних регіонів. Це робиться так, як показано нижче:

$$x_i(t+1) = x_i(t) + rand(-1,1)step_{ind},$$

де $x_i(t)$ та $x_i(t+1)$ представляють положення риби i до і після окремого оператора руху відповідно. $rand(-1,1)$ - це рівномірно розподілене випадкове число, що варіюється від -1 до 1 і $step_{ind}$ - параметр, який визначає максимальне переміщення для цього руху. Нова позиція $x_i(t+1)$ приймається лише в тому випадку, якщо фізична придатність риби покращується зі зміною положення. Якщо це не так, риба залишається в тому ж положенні, і $x_i(t+1) = x_i(t)$.

В оригінальній версії алгоритму стадія індивідуального плаваня допускає рух риби лише тоді, коли її фізична придатність покращується. Однак у дуже гладкому просторі пошуку було б багато рухливих випробувань без успіху, і алгоритм міг би не зійтись. Розроблювана у даній роботі модифікація створена з метою покращення розвідувальної здатності алгоритму.

Для вирішення ціє проблем я ввід параметр α, для якого $0 \leq \alpha \leq 1$, в окремій складовій руху. α розпадається експоненціально разом з повтореннями і вимірює ймовірність погіршення норми для кожної риби. Це означає, що кожного разу, коли риба намагається перейти в положення, яке не покращує її придатність, вибирається випадкове число, і якщо воно менше за α, рух дозволяється. Однак лише ті риби, які виявили покращення своєї придатності в межах індивідуального плавання, можуть внести свій внесок у розрахунок значення I формули 2.1, що використовується в рамках інстинктивно-колективного плавання.

Ця модифікація призначена для поліпшення здатності досліджувати алгоритм, дозволяючи стохастичні погіршувальні рухи. Однак, оскільки

параметр α експоненційно розпадається уздовж ітерацій, цей ефект інтенсивний лише на початку процесу пошуку і стає неактуальним після деяких ітерацій.

2) Інстинктивно-колективне плавання (collective-instinct swimming)

Середнє значення окремих переміщень розраховується на основі наступного:

$$I = \frac{\sum_{i=1}^{N} \Delta x_i \Delta f_i}{\sum_{i=1}^{N} \Delta f_i} \qquad (2.1)$$

Вектор I являє собою середньозважене переміщення кожної риби. Це означає, що риби, у яких було більше високе поліпшення, будуть залучати інших риб на свою позицію. Після обчислення вектора кожна риба буде рухатися відповідно до:

$$x_i(t+1) = x_i(t) + I.$$

3) Колективно-вольове плавання (collective volition swimming)

Цей оператор використовується для регулювання дослідницької / експлуатаційної здатності косяка під час процесу пошуку. Перш за все, баріцентр B косяка розраховується на основі положення x_i та ваги W_i кожної риби:

$$B(t) = \frac{\sum_{i=1}^{N} x_i W_i(t)}{\sum_{i=1}^{N} W_i(t)},$$

а потім, якщо загальна вага косяка $\sum_{i=1}^{N} W_i(t)$ зросла з останньої ітерації до поточної, риби притягуються до баріцентру відповідно до рівняння:

$$x_i(t+1) = x_i(t) - step_{vol} rand(0,1) \frac{x_i(t) - B(t)}{distance(x_i(t), B(t))},$$

Якщо загальна вага косяка не покращилася, риби розподіляються від баріцентра згідно з рівнянням:

$$x_i(t+1) = x_i(t) + step_{vol} rand(0,1) \frac{x_i(t) - B(t)}{distance(x_i(t), B(t))},$$

де $step_{vol}$ визначає розмір максимального переміщення, виконаного за допомогою цього оператора. $distance(x_i(t), B(t))$ - евклідова відстань між

рибами i позиції та баріцентру косяка. $rand(0,1)$ - рівномірно розподілене випадкове число, що варіюється від 0 до 1.

Окрім операторів переміщення, було також визначено оператор годування, який використовується для оновлення ваги кожної риби відповідно до:

$$W_i(t+1) = W_i(t) + \frac{\Delta f_i}{\max(|\Delta f_i|)},$$

де $W_i(t)$ - параметр ваги для риби i, Δf_i - це варіація фітнесу між останньою та новою позицією, а $\max(|\Delta f_i|)$ представляє максимальне абсолютне значення варіації придатності серед усіх риб у косяку. W може змінюватись лише від 1 до $\frac{W_{scale}}{2}$, що є атрибутом визначеним користувачем. Ваги всіх риб ініціалізуються зі значенням $\frac{W_{scale}}{2}$.

2.2 Мова програмування

Для розробки системи в даній роботі було обрано мову програмування C#, чому сприяло багато факторів, які будуть послідовно надаватися в наступних пунктах.

2.2.1 Мова C#

З усіх мов програмування C#, мабуть, одна з найкращих. Ця багатопарадигменна мова універсальна, досить проста у вивченні та об’єктно-орієнтована.

C# - це сучасна мова програмування загального призначення, яка може бути використана для виконання широкого кола задач та завдань, що охоплюють різні професії. C# в основному використовується на платформі Windows .NET, хоча його можна застосувати до платформи з відкритим кодом.

Ця надзвичайно універсальна мова програмування - це об’єктно-орієнтована мова програмування (ООП), яка не дуже поширена, і досить нова, але при цьому вже є надійним інстументом.

Порівняно з давно існуючими мовами, такими як Python та PHP, C# є молодим доповненням до сімейства програмування, якому вже майже двадцять років. Мова була розроблена у 2000 році Андерсом Хейлсбергом з компанії Microsoft, датським інженером-програмістом, який має досвід створення популярних творінь. Андерс брав участь у створенні кількох надійних інструментів і мов програмування, включаючи Microsoft TypeScript і Delphi, відповідну заміну Turbo Pascal [27].

Станом на лютий 2019 року C# зайняла 4-е місце в індексі популярності мов програмування PYPL, відразу після Java, JavaScript та Python. Станом на квітень 2021 року C# й надалі посідає 4-е місце в індексі популярності мов програмування PYPL [28]. Дані, що використовуються для складання цього індексу, базуються на тому, як часто люди шукають підручник з різних мов програмування в Google.

На рис. 2.4 можна наглядно побачити, яким мовам програмування поступається C# у популярності за даними PYPL.

Rank	Change	Language	Share	Trend
1		Python	29.5 %	-1.0 %
2		Java	17.51 %	-0.6 %
3		JavaScript	8.19 %	+0.2 %
4		C#	7.05 %	-0.2 %
5	↑	C/C++	6.73 %	+1.0 %
6	↓	PHP	6.23 %	+0.0 %
7		R	3.86 %	+0.0 %
8		Objective-C	2.77 %	+0.3 %
9	↑	TypeScript	1.87 %	-0.0 %
10	↓	Swift	1.85 %	-0.3 %

Рисунок 2.4 – Популярність мов програмування за даними PYPL [28]

На рис. 2.5 наведено динаміку популярності п’яти мов програмування: Python, Java, JavaScript та C#, за останні 17 років за даними PYPL.

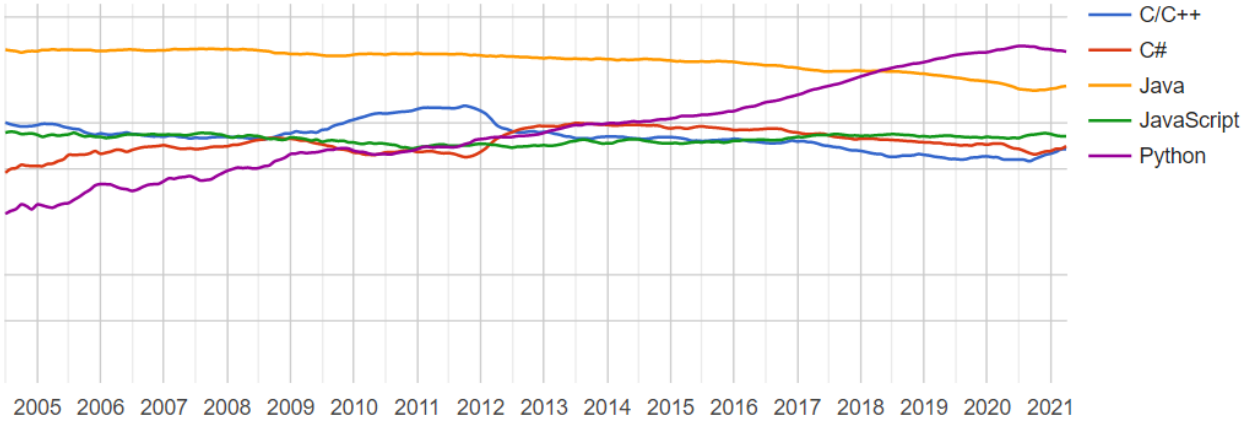


Рисунок 2.5 – Динаміка популярності мов програмування за даними PYPL [28]

В останні роки C# також постійно з'являється в першій десятці мов програмування в TIOBE Index [29], звіті, дані якого взяті з компіляції популярних пошукових систем, включаючи Google, YouTube і Bing.

На рис. 2.6 можна наглядно побачити рейтинг мов програмування за даними TIOBE Index.

Apr 2021	Apr 2020	Change	Programming Language	Ratings	Change
1	2	^	C	14.32%	-2.40%
2	1	v	Java	11.23%	-5.49%
3	3		Python	11.03%	+1.72%
4	4		C++	7.14%	+0.36%
5	5		C#	4.91%	+0.16%
6	6		Visual Basic	4.55%	-0.18%
7	7		JavaScript	2.44%	+0.06%
8	14	^^	Assembly language	2.32%	+1.16%
9	8	v	PHP	1.84%	-0.54%
10	9	v	SQL	1.83%	-0.34%

Рисунок 2.6 – Популярність мов програмування за даними TIOBE Index [29]

На рис. 2.7 наведено динаміку популярності мов програмування за останні 19 років за даними TIOBE Index.

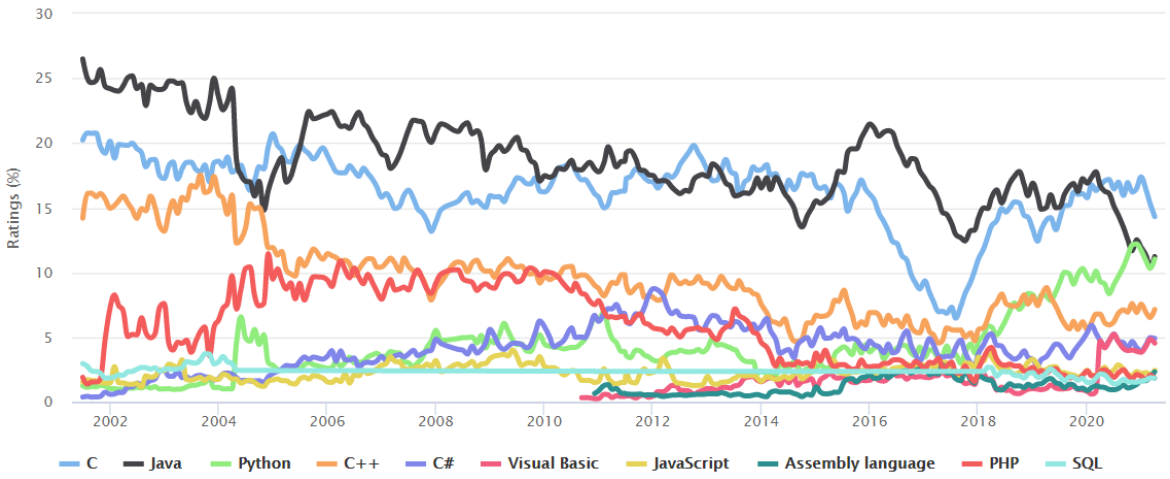


Рисунок 2.7 – Динаміка популярності мов програмування за даними TIOBE Index [29]

Спочатку C# називався COOL - абревіатура, що означає «об'єктно-орієнтована мова, подібна C. На жаль, Microsoft не змогла утримати забавну назву з причин, пов'язаних з законом про товарні знаки.

C# спочатку була розроблена, щоб конкурувати з Java. Судячи по швидкому зростанню популярності та позитивній реакції як новачків, так і досвідчених розробників, можна з упевненістю сказати, що мета була досягнута.

C# - відмінний вибір для розробників з досвідом написання коду від середнього до високого. Хоча експерти визнають, що ця мова відрізняється помірною складністю, вони згодні з тим, що її досить просто зрозуміти і досягти успіху в ній.

Це пов'язано з тим, що C# є мовою високого рівня, що означає, що її відносно легко читати і писати, що робить її надійним вибором для новачків і зручним варіантом для експертів. Крім зручності читання, C# також можна використовувати для автоматизації складних завдань, що вимагають багато часу для досягнення незначних результатів.

Ця мова програмування також є статистично типізованою, що означає, що помилки виявляються ще до запуску програми. Це значно спрощує виявлення

дрібних недоліків у вашому стеку, які в іншому випадку були б майже непомітними, не говорячи вже про те, що вони дуже неприємні.

Хоча C# може знайти хороше застосування в руках програмістів всіх мастей, велика частина призначеної для користувача бази мови складається з тих, хто небайдужий до платформи Microsoft.

Як і інші мови програмування загального призначення, C# можна використовувати для створення ряду різних програм і додатків: мобільних додатків, настільних додатків, хмарних сервісів, веб-сайтів, корпоративного програмного забезпечення та ігор. Хоча C# надзвичайно універсальна, найбільш часто вона використовується в трьох областях [30].

1. Розвиток веб-сайту

   C# часто використовується для розробки професійних динамічних веб-сайтів на платформі .NET або програмного забезпечення з відкритим вихідним кодом. Таким чином, навіть якщо ви не є шанувальником архітектури Microsoft, ви все одно можете використовувати C# для створення повнофункціонального веб-сайту. Оскільки ця мова є об'єктно-орієнтованою, її часто використовують для розробки неймовірно ефективних, легко масштабованих і простих в обслуговуванні веб-сайтів.

2. Windows-додатки

   C# була створена компанією Microsoft для Microsoft, тому легко зрозуміти, чому вона найбільш широко використовується для розробки настільних додатків Windows. Додаткам C# для найкращого функціонування потрібна платформа Windows .NET, тому найбільш сильним варіантом використання цієї мови є розробка додатків та програм, специфічних для архітектури платформи Microsoft.

3. Ігри

   C# може бути однією з кращих мов програмування для ігор. Ця мова широко використовується для створення улюблених ігор. Unity -

безумовно, найпопулярний ігровий движок, на якому було створено понад третини кращих і найбільш популярних ігор в галузі. C# легко інтегрується з движком Unity і може використовуватися практично на будь-якому сучасному мобільному пристрої або консолі завдяки кросплатформенним технологіям, таким як Xamarin.

2.2.2 Переваги мови C#

Є багато переваг та особливостей мови C#, які роблять її більш корисною мовою програмування, ніж інші мови програмування, такі як Java, C, C ++ тощо. Отож є багато переваг мови C#, але деякі важливі переваги мови C# описані нижче [31, 32].

1. Об'єктно-орієнтована

   Мова програмування C# є чисто об'єктно-орієнтованою мовою, тому вона дозволяє створювати модульні підтримувані додатки і повторно використовувані коди.

2. Кросплатформеність

   Найважливішою вимогою для програмування на C# є середовище NET. На вашому комп'ютері повинна бути встановлений NET Framework для правильної роботи вашого додатка.

3. Автоматичний збір сміття

   У програмуванні на C# встановлена дуже ефективна система, яка автоматично збирає і стирає сміття, присутнє в системі. C# дуже ефективна в управлінні системою, тому що вона не створює безладдя в системі, і система не зависає під час виконання.

4. Уникання проблеми витоку пам'яті.

   Основною перевагою мови C# є надійне резервне копіювання пам'яті. C# містить резервне копіювання великої пам'яті, тому проблема

витоку пам'яті та інші подібні проблеми не виникають, як у випадку з мовою C ++.

5. Легкість в розробці

   Мова C# має багатий клас бібліотек, які спрощують реалізацію багатьох функцій. Мова програмування C# впливає на більшість програмістів в світі і має великий досвід в світі програмування.

6. Краща інтеграція

   Додаток, написаний на .NET, матиме кращу інтеграцію і можливості інтерпретації в порівнянні з іншими NET-технологіями.

7. Вигідність

   Вартість обслуговування менша і безпечніше у використанні в порівнянні з іншими мовами.

8. Підтримка програмування

   Ви можете придбати підтримку від Microsoft в програмуванні на C#. Якщо виникне якась проблема, ви можете вирішити її за допомогою підтримки Microsoft.

9. Властивості і індексатори (Properties and Indexers)

   У програмуванні на C# є такі функції, як властивості і індексатори, які недоступні в мові Java.

10. Найбільш корисна

    На C# можна розробляти власні програми для iOS, Android і Windows Phone за допомогою платформи Xamarin.

11. Найбільш потужна

    Мова C# - найпотужніший мова програмування для .NET Framework.

12. Знайомий синтаксис

    Досить легко освоїти і продуктивно працювати, володіючи практичним знанням таких мов, як C, C++, Java, тому що її основний синтаксис аналогічний мовам в стилі C, див. рисунок 2.8

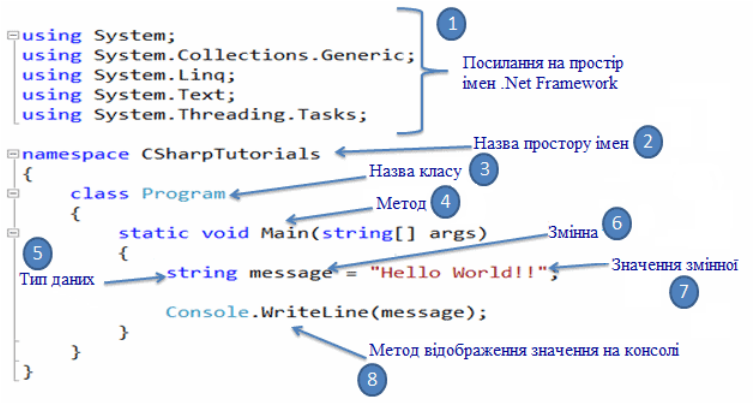


Рисунок 2.8 – Структура базової програми на C#

2.2.3 Недоліки мови C#

Звичайно, є багато переваг мови C#, які було описано вище, але також мова має деякі недоліки, які описано нижче [33, 34].

1. Не є гнучкою мовою

   C # повністю заснований на платформі Microsoft .Net, тому це не є гнучкою мовою.

2. Працює повільно

   Якщо ми щось змінюємо в написаному коді на C#, то спочатку нам потрібно скомпілювати.

3. Неможливо здійснити дії низького рівня

   Програміст не може робити такі речі низького рівня, як безпосередня взаємодія з апаратним забезпеченням за допомогою драйверів та прошивки.

4. Підтримка старих платформ .NET

   Корпорація Microsoft припиняє підтримку старих платформ .NET після декількох оновлень операційних систем. Наприклад, старіші сервери Windows 2000 можуть підтримувати лише програми .NET 2.0. Хоча установка старої операційної системи здається помилкою, багато корпоративних організації зберігають старі операційні системи через

безліч проблем, які може викликати оновлення платформи. Основні оновлення серверної архітектури мають бути протестовані та затверджені перед розгортанням, що збільшує час і витрати на розробку.

5. Складність при переході на ядро .NET:

   Перехід на ядро .NET може бути довгим і важким. Додатки, створені з використанням старіших версій .NET, набагато більше, ніж додатки, створені з використанням ядра .NET. Перехід цих додатків на останні версії .NET може бути трохи складним.

6. Прив'язка до постачальника.

   На жаль, оскільки пакет .NET знаходиться у веденні Microsoft, будь-які зміни або обмеження, які може накласти компанія, неминуче вплинуть на проекти, що виконуються в рамках цієї платформи. Це означає, що у розробників буде менше контролю.

2.2.4 Підсумок щодо вибору мови

Узагальнюючи всі зважені аргументи "за" та "проти" для використання C# у нашому проекті, ми можемо зробити висновок, що вона повністю відповідає вимогам майбутньої програми і може бути написана таким чином. Слід додати, що кількість переваг, що надаються при її використанні, набагато перевищує кількість недоліків.

2.3 Технології та бібліотеки

Далі описуються технології та бібліотеки, які будуть використовуватися у проекті.

2.3.1 .NET Framework 4.7.2

.NET - це платформа для розробників, що складається з інструментів, мов програмування та бібліотек для побудови багатьох різних типів додатків.

Існують різні реалізації .NET. Кожна реалізація дозволяє виконувати .NET-код у різних місцях - Linux, macOS, Windows, iOS, Android та багатьох інших [35].

- .NET Framework - оригінальна реалізація .NET. Вона підтримує роботу веб-сайтів, служб, настільних програм тощо в Windows.
- .NET Core - це міжплатформна реалізація для запуску веб-сайтів, служб та консольних програм у Windows, Linux та macOS. .NET Core є відкритим кодом на GitHub.
- Xamarin / Mono - це реалізація .NET для запуску програм на всіх основних мобільних операційних системах, включаючи iOS та Android.

Обрана реалізація .NET для нашого проекту - .NET Framework.

.Net Framework Architecture - це модель програмування для платформи .Net, яка забезпечує середовище виконання і інтеграцію з різними мовами програмування для простої розробки та розгортання різних додатків Windows і настільних додатків. Вона складається з бібліотек класів і компонентів багаторазового використання. Базова архітектура платформи .Net показана нижче.

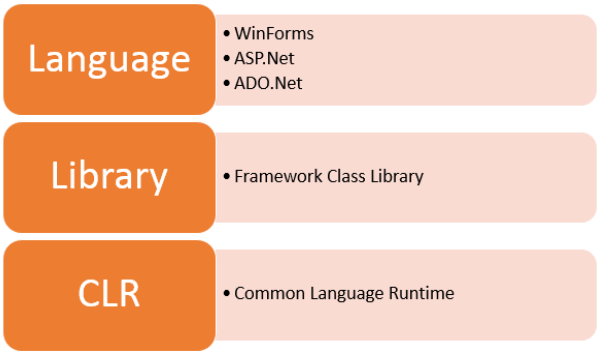


Рисунок 2.9 – Діаграма архітектури .Net Framework

Архітектура .Net Framework заснована на наступних ключових компонентах [36]:

1. Common Language Runtime

   «Інфраструктура спільної мови» або CLI - це платформа в архітектурі .Net, на якій виконуються програми .Net.

   CLI має наступні ключові особливості:

   - Обробка винятків - це помилки, що виникають під час запуску програми.
   - Garbage Collection - це процес видалення непотрібних ресурсів, якщо вони більше не потрібні.
   - Робота з різними мовами програмування.

   Як вже зазначалося, розробник може розробити додаток на різних мовах .Net.

   1. Мова - це мова програмування. У нашому проекті вкористовується C#.
   2. Компілятор - існує компілятор, який буде унікальним для кожної мови програмування. Тож в основі мови C# буде окремий компілятор C#.
   3. Common Language Interpreter - це останній рівень в .Net, який буде використовуватися для запуску .net-програми, розробленої на будь-якій мові програмування. Таким чином, наступний компілятор відправить програму на рівень CLI для запуску програми .Net.

2. Class Library

   .NET Framework включає в себе набір стандартних бібліотек класів. Бібліотека класів - це сукупність різних методів та функцій, які можуть бути використані для розробки.

3. Languages

Типи програм, які можна створити в середовищі .Net, класифікується в основному за такими категоріями:

- WinForms - використовується для розробки програм на основі Forms, які працювали б на машині кінцевого користувача.
- ASP.Net - використовується для розробки веб-додатків, які створені для роботи в будь-якому браузері, наприклад Opera, Chrome або Internet Explorer.
- ADO.Net - ця технологія використовується для розробки додатків для взаємодії з БД, такими як Oracle або Microsoft SQL Server.

Обраний інтерфейс програмування додатків (API) у нашому проекті- WinForms.

Перейдемо до переваг, що надає .Net Framework при своєму використанні. Ось основні з них:

- Взаємодія - структура .Net надає багато зворотної підтримки. Припустимо, якщо у вас був додаток, побудований на старій версії фреймворка .Net, скажімо 2.0., а якщо ви намагалися запустити той самий додаток на машині, яка мала вищу версію .Net framework, скажімо 3.5., то додаток все одно працює. Це тому, що з кожним випуском Microsoft гарантує, що старі версії фреймворку добре поєднуються з останньою версією.
- Переносимість - програми, побудовані на фреймворці .Net, можуть працювати на будь-якій платформі Windows.
- Безпека - .NET Framework має хороший механізм захисту. Вбудований механізм безпеки допомагає як у валідації, так і в верифікації програм. Кожна програма може чітко визначити свій механізм захисту. Кожен

механізм захисту використовується для надання користувачеві доступу до коду або запущеної програми.

- Керування пам'яттю - середовище виконання Common Language виконує всю роботу або управління пам’яттю. Структура .Net має всі можливості бачити ті ресурси, які не використовуються запущеною програмою. Потім вони відповідно звільнять ці ресурси. Це робиться за допомогою програми під назвою «Сміттєвий збирач», яка працює як частина фреймворку .Net.
- Збирач сміття працює через рівні проміжки часу, постійно перевіряє, які системні ресурси не використовуються, і звільняє їх відповідно.
- Спрощене розгортання - .Net Framework також має інструменти, які можна використовувати для упаковки програм, побудованих на .Net framework. Потім ці пакети можна розподілити на клієнтських машинах. Потім пакети автоматично встановлюватимуть програму.

2.3.2 Windows Forms

WPF, Window Forms, Windows Store - різні технології для створення графічних інтерфейсів за допомогою платформи .NET. Однак найпростіша і найзручніша платформа - Window Forms.

Windows Forms - це безкоштовна бібліотека класів з графічним інтерфейсом (GUI) з відкритим вихідним кодом, що входить до складу Microsoft .NET Framework або Mono Framework , що надає платформу для написання багатофункціональних клієнтських додатків для настільних, портативних і планшетних ПК.

Підхід до розробки прикладних програм на Windows Forms грунтується на графічному інтерфейсі GDI (Graphics Device Interface) - це інтерфейс

Windows для представлення графічних об'єктів і передачі їх на пристрої відображення, такі як монітори і принтери.

Windows Forms надає доступ до власних елементів керування користувацького інтерфейсу Windows, укладаючи існуючий API Windows у керований код. За допомогою Windows Forms .NET Framework забезпечує більш повну абстракцію над API, ніж Visual Basic або MFC.

Windows Forms подібний до бібліотеки Microsoft Foundation Class (MFC) при розробці клієнтських додатків. Однак він не надає платформу додатків за замовчуванням, як MFC. Кожен елемент керування у програмі Windows Forms є конкретним екземпляром класу.

Всі візуальні елементи в бібліотеці класів Windows Forms є похідними від класу Control. Це забезпечує мінімальну функціональність елемента користувальницького інтерфейсу, таку як розташування, розмір, колір, шрифт, текст, а також загальні події, такі як клацання і перетягування. Клас Control також має підтримку закріплення, що дозволяє елементу управління перевпорядковувати своє становище під своїм батьківським елементом. Крім надання доступу до власних елементів управління Windows, таким як кнопка, текстове поле, прапорець і список, Windows Forms додала свої власні елементи управління для розміщення ActiveX, компонування, перевірки і прив'язки даних. Ці елементи керування відображаються з використанням GDI +.

Одна з переваг Windows Forms - в тому, що на ній можна писати коссплатформенні додатки. Прості проекти, написані на Windows Forms, можна досить легко перенести на іншу операційну систему, якщо на ній встановлений .Net Framework потрібної моделі, на якому написаний ваш проект. Іншою перевагою є те, що даний момент існує велика кількість готових елементів управління, які можна купити або використовувати безкоштовно. З точки зору написання, Visual Studio краще пристосований до WinForms, так як в WPF більше необхідно робити самому [37].

2.3.3 Net Charting

.Net Charting поєднує в собі візуально приголомшливу графіку та всеосяжний інтерфейс, щоб запропонувати одне з найбільш привабливих графічних рішень, доступних для будь-якої платформи.

.Net Charting використовує платформу .NET та GDI+, забезпечуючи рішення для керованих графіків для розробників C#, що працюють з WinForms. Використання новітніх технологій Microsoft дозволило вийти за межі існуючих стандартів візуального графіку та створити візуальні ефекти, набагато сильніші, ніж типові рішення для динамічного складання графіків.

Протягом багатьох років .Net Charting впроваджував нові технології створення графіків, забезпечуючи клієнтам безпрецедентну простоту використання та автоматизації [38].

Основні особливості та переваги:

- Бібліотека доступна для вільного скачування з веб-сайту Microsoft.
- Простота у використанні.
- Надзвичайна продуктивність у реальному часі.
- Необмежена, множинна та налаштовувана вісь X, Y.
- Масштабування та панорамування.
- Налаштування типу графіків.
- Сучасна веб-естетика.
- Налаштування стилю та тематики.

Основний недолік використання - низька продуктивність на великих обсягах даних. Ще одна проблема з елементом управління Chart полягає в тому, що він так довго перебував в розробці і на різних етапах включення в .NET, що тепер, коли це офіційний елемент управління, документація по ньому дуже неповна і розрізнена.

2.3.4 ZedGraph

ZedGraph - це дуже зручний компонент для малювання графіків під .NET Framework; безкоштовна і дуже багата бібліотека виведення різних видів графіків. Перекриває величезну кількість типів графіків, які можна використовувати в своїх проектах. Відрізняється спрощеною графікою, простотою у використанні. Також є підтримка настроювання через дизайнер форм Visual Studio.

Бібліотека класів ZedGraph дуже гнучка. Майже кожен рівень графіка може бути змінений користувачем. У той же час, щоб забезпечити простоту використання бібліотеки класів, усі властивості графіка надають значення за замовчуванням. Код, що міститься в бібліотеці класів, може вибрати діапазон масштабу та розмір кроку та розмір, який слід адаптувати відповідно до розділених даних [39].

Основні переваги:

- Бібліотека доступна для вільного скачування з сайту SourceForge.
- ZedGraph ліцензований згідно LGPL.
- Дуже проста у використанні.
- Доступна багата документація.
- Можливість масштабування та панорамування.
- Налаштування типу графіків.

Основними недоліками є низька продуктивність на великих обсягах даних та непрезентабельний вигляд.

2.3.5 RegularExpressions

Регулярні вирази є частиною невеликої технічної області і широко використовуються в багатьох програмах. Регулярні вирази – це як невелика

мова програмування, яка має конкретне призначення: знаходити підрядки у великих рядкових виразах.

Це не є новою технологією, спершу вона появилася в середовищі UNIX і в більшості використовується в мові програмування Perl. Розробники Microsoft перенесли цю технологію в Windows, де до недавнього часу вона застосовувалася в основному зі сценарними мовами. Однак багато класів .NET у просторі імен System.Text.RegularExpressions тепер підтримують регулярні вирази. Ви можете знайти регулярні вирази у багатьох частинах середовища .NET Framework.

Регулярні вирази - це стандарт зіставлення зі зразком для синтаксичного аналізу і заміни рядків і спосіб для користувача висловити, як комп'ютерна програма повинна шукати вказаний зразок в тексті, а потім, що програма повинна робити при виявленні кожного зіставлення зі зразком. Іноді їх називають скороченим словом «regex». Вони є потужним способом пошуку і заміни рядків в заданому форматі [40].

Приклад регулярного виразу наведено на рис. 2.10:

```
^(0|([1-9][0-9]*))(\,[0-9]+)?$
```

Рисунок 2.10 – Приклад регулярного виразу

Даний регулярний вираз перевіряє рядок, який повинен відповідати зразку, а саме числу типу double з позитивним знаком.

2.3.6 Microsoft SQL Server

SQL Server - це система управління базами даних, в роботі з якою використовується мова програмування SQL.

СУБД SQL Server використовуються для створення, розміщення, зберігання і управління реляційними (табличними) базами даних на

спеціальних серверах або в хмарі. Вони працюють через настільні додатки і web-сайти. До основних переваг їх функціонування відносяться:

- високошвидкісний доступ до даних, що забезпечується надійною клієнт-серверною архітектурою СУБД;
- простота роботи і адміністрування, обумовлені зрозумілою структурою мови програмування SQL;
- безпека зберігання інформації в БД - завдяки можливості шифрування даних і резервного копіювання.

Специфіка роботи сервера бази даних SQL Server полягає в транзакційній обробці даних. Це означає, що по кожному запиту від СУБД обробляється і зберігається невелика кількість інформації.

Застосування SQL Server дозволяє автоматизувати рішення різних бізнес-задач, підтримувати проведення аналітики даних в режимі онлайн, відслідковувати напрямок ресурсів СУБД [41].

2.3.7 MSTest

Невід'ємною частиною кожного середовища тестування користувальницького інтерфейсу є використання середовища модульного тестування. Один з найпопулярніших в світі .NET - це MSTest.

Фреймворк MSTest підтримує модульне тестування у Visual Studio. У нашому проекті я використовуватиму класи у просторі імен Microsoft.VisualStudio.TestTools.UnitTesting під час кодування модульних тестів.

Фреймворк MSTest забезпечує необхідні інструменти для валідації та верифікації вихідного коду. Структура розпізнає тести через різні атрибути / анотації, в яких присутній код тесту [42].

Основні переваги:

- Швидкість виконання

  MsTest - це власна бібліотека модульного тестування, яка постачається з Visual Studio від Microsoft. Як результат MsTest є швидшим.

- Розпаралелювання

  Основна особливість полягає в здатності MsTest виконуватися паралельно на рівні методу.

- Широкий спектр атрибутів та тверджень

  Бібліотека надає 29 assertions та 20 attributes.

Основним недоліком MSTest є відсутність активності в його підтримці і вдосконаленню. Отож, останнім часом популярність MSTest сильно падає.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі були описані детально алгоритми FSS та GA, а також розглянуті технології, що будуть використані для написання системи – об’єкта розробки. Також було обґрунтовано, навіщо ці технології використовуватимуться та які переваги вони надаватимуть при їх використанні.

Спочатку було описано детально роботу двох еволюційних алгоритмів FSS та GA. Було наведено блок схеми алгоритмів, основні алгоритми роботи, а також описано основні математичні формули, на яких базуються дані алгоритми. Було описано модифікацію алгоритму FSS, яка гарантує кращу сходимість алгоритму у гладкому просторі функцій.

Потім було розглянуто питання вибору мови програмування, яка використовувалася для написання проекту. Найбільш підходящим варіантом стала мова C#, яка була детально розглянута у даному розділі. Зокрема, у розділі розглянуто загальну інформацію про цю мову програмування, її численні переваги та недоліки. На основі чого був сформований висновок, який вказує на те, що мова повністю відповідає вимогам нашого проекту, і було прийнято рішення про її використання, оскільки в цьому випадку розроблена система отримає багато переваг.

Також у другій частині цього розділу обговорюються платформи, фреймворки та бібліотеки, які будуть використані для написання вихідного коду. Всі вони будуть грати різні ролі, але всі ці ролі важливі для нашого проекту.

Спочатку було розглянуто платформу .NET Framework, яка є платформою для розробників, що складається з інструментів, мов програмування та бібліотек для побудови багатьох різних типів додатків. Для нас важливим є те,

що вона підтримує роботу настільних програм в Windows, а також включає в себе набір стандартних бібліотек класів, які для нас є дуже корисними.

Далі було розглянуто технологію Windows Forms, безкоштовну бібліотеку класів з графічним інтерфейсом з відкритим вихідним кодом, що входить до складу Microsoft .NET Framework та надає платформу для написання багатофункціональних клієнтських додатків для настільних ПК. Було розглянуто основні переваги та недоліки використання.

Наступною була розглянута бібліотека Net Charting, яка дозволить нам будувати графіки функцій, необхідні для алгоритму FSS. Було розглянуто основні переваги та недоліки використання даної бібліотеки.

Також було розглянуто бібліотеку ZedGraph, яка як і Net Charting дозволить нам будувати графіки функцій, але вже для алгоритму GA. Ця бібліотека має свої основні переваги, відмінні від Net Charting.

Передостанньою була розглянута бібліотека RegularExpressions, яка дає можливість працювати з регулярними виразами. Ця бібліотека дасть нам можливість перевіряти коректність вводу вхідних параметрів алгоритмів користувачем.

Останнім було розглянуто фреймворк під назвою MSTest, який стане у пригоді нам при тестуванні нашого додатку і який на даний момент є майже незамінним у галузі тестування .NET -додатків.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Відповідно до досліджених матеріалів та обраного способу реалізації ми можемо перейти до фази розробки програмного продукту. Для того, щоб створити гнучку та настроювану систему пошуку екстремуму багатомірної функції на основі еволюційних алгоритмів GA та FSS, необхідно описати основні частини розробленого продукту, функціональні блоки програми та схему їх взаємодії.

3.1 Розробка програмних компонентів GA

Далі буде описано основні класи генетичного алгоритму.

3.1.1 Клас BaseSpecies

Всі види повинні бути похідними від базового класу *BaseSpecies <TSpecies>*, де *TSpecies* - конкретний тип для виду. *TSpecies* повинен бути похідним від класу *BaseSpecies <TSpecies>*.

BaseSpecies - це абстрактний клас, який містить статичні protected-методи для схрещування і мутації. Розберемо деякі методи детальніше. Почнемо з методів, які необхідно реалізувати в похідних класах.

Перший з них це *abstract public void TestChromosomes()*. Цей метод повинен встановлювати внутрішню захищену булеву змінну *m_Dead*, яка показує, чи підходять хромосоми по обмеженню, які на них накладають (наприклад, в нашому прикладі, чи потрапляють значення хромосом в відрізок [-5.0; 5.0]). Якщо значення хромосом нас влаштовують, то *m_Dead* треба

встановити в *false*, інакше в *true*. Завдяки цій змінній (отримується через властивість Dead) популяція відкидає свідомо невдалі види.

Наступний метод, який необхідно реалізувати в похідному класі, це *abstract public TSpecies Cross (TSpecies Species)*. Цей метод створює новий вид, схрещуючи себе та інший вид, який передається як аргумент. Тут треба зробити деякі пояснення. Зазвичай, якщо вид містить кілька хромосом, схрещують хромосоми "одного виду", тобто в нашому випадку схрещуємо *a0* себе (тобто this) і *a0* іншого виду, аналогічно схрещуємо *a1* і *a1*, тобто немає сенсу схрещувати *a0* і *a1* іншого класу.

Сенс схрещування полягає в тому, що беремо одну частину хромосоми одного виду і другу частину іншого виду і створюємо з них нову хромосому, яка і буде міститися в новому вигляді. Часто хромосоми схрещують побітово. Суть його полягає в тому, що спочатку випадково вибираємо точку розриву (схрещування) хромосоми, потім створюємо нову хромосому, яка складається з лівої частини першої хромосоми і правої частини другої.

В класі *BaseSpecies* є публічні статичні методи для схрещування хромосом деяких типів (а саме: *Double, Int64* і *Int32*). Розберемо детальніше перші два методи. В даному випадку по суті є дві точки перетину - в середині слова і знак числа, тобто спочатку схрещуються хромосоми побітово без врахування знаку, а потім також випадково беремо знак від однієї з хромосоми.

Працює цей метод таким чином: спочатку перетворюємо хромосоми з типу *Double* в *Int64*. Це зроблено для того, щоб можна було б без проблем схрещувати побітово. Після схрещування вже типів *Int64* без врахування знаку, переводимо число назад в *Double*, а потім беремо знак однієї з двох хромосом. Реалізація методу схрещування хромосом типу Double на рисунку 3.1.

А тепер давайте подивимося як відбувається схрещування типів *Int64*. Як видно з коду на рисунку 3.2, спочатку схрещуються хромосоми без урахування знаку, а потім вибирається знак однієї з хромосом. Точніше, це працює так, що

порівнюють знак обраної хромосоми з результатом, і, якщо знаки не співпали, то знак результату змінюється на протилежний.

```
/// <summary>
/// Схрестити дві хромосоми типу double
/// </summary>
/// <param name="x">1-а хромосома</param>
/// <param name="y">2-а хромосома</param>
/// <returns>Нова хромосома</returns>
static protected double Cross(double x, double y)
{
    Int64 ix = BitConverter.DoubleToInt64Bits(x);
    Int64 iy = BitConverter.DoubleToInt64Bits(y);

    double res = BitConverter.Int64BitsToDouble(BitCross(ix, iy));

    if (m_Rnd.Next() % 2 == 0)
    {
        if (x * res < 0)
        {
            res = -res;
        }
    }
    else
    {
        if (y * res < 0)
        {
            res = -res;
        }
    }

    return res;
}
```

Рисунок 3.1 – Реалізація методу схрещування хромосом типу Double

```
/// <summary>
/// Схрестити побітово без урахування знаку дві хромосоми типу Int64
/// </summary>
/// <param name="x">1-а хромосома</param>
/// <param name="y">2-а хромосома</param>
/// <returns>Нова хромосома</returns>
static protected Int64 BitCross(Int64 x, Int64 y)
{
    // Число біт, що залишилися зліва від точки перетину хромосом
    int Count = m_Rnd.Next(62) + 1;
    Int64 mask = ~0;

    mask = mask << (64 - Count);

    return (x & mask) | (y & ~mask);
}

/// <summary>
/// Схрестити побітово з урахуванням знака дві хромосоми типу Int64
/// </summary>
/// <param name="x">1-а хромосома</param>
/// <param name="y">2-а хромосома</param>
/// <returns>Нова хромосома</returns>
static protected Int64 Cross(Int64 x, Int64 y)
{
    Int64 res = BitCross(x, y);

    if (m_Rnd.Next() % 2 == 0)
    {
        if (x * res < 0)
        {
            res = -res;
        }
    }
    else
    {
        if (y * res < 0)
        {
            res = -res;
        }
    }

    return res;
}
```

Рисунок 3.2 – Реалізація методу схрещування хромосом типу Int64

Розглянемо наступний метод *public void Mutation()*. Тут все дуже схоже на схрещування. Мутація діє на одну хромосому. У теорії при мутації можуть відбуватися будь-які зміни. Але в даній реалізації мутація змінює один біт в слові. Ось приклад коду наведено на рис. 3.3.

```
/// <summary>
/// Мутація для типу double
/// </summary>
/// <param name="val">Значення, яке мутуємо</param>
/// <returns>Промутоване значення</returns>
static protected double Mutation(double val)
{
    UInt64 x = BitConverter.ToUInt64(BitConverter.GetBytes(val), 0);
    UInt64 mask = 1;
    mask <<= m_Rnd.Next(63);
    x ^= mask;

    double res = BitConverter.ToDouble(BitConverter.GetBytes(x), 0);

    return res;
}

/// <summary>
/// Мутація для типу Int64
/// </summary>
/// <param name="val">Значення, яке мутуємо</param>
/// <returns>Промутоване значення</returns>
static protected Int64 Mutation(Int64 val)
{
    Int64 mask = 1;
    mask <<= m_Rnd.Next(63);

    return val ^ mask;
}
```

Рисунок 3.3 – Реалізація статичного методу для мутації

Мутації відбуваються порівняно рідко. Наприклад, часто ймовірність мутації роблять 5-10%, але іноді варто спробувати зробити її побільше (приблизно 30%). Також досить часто в якості мутації для дрібних чисел краще застосовувати не написану вище функцію, а просто генератор випадкових чисел, який видає випадкове число в потрібному нам інтервалі. У доданому прикладі для дрібних чисел так і зроблено, а для цілих хромосом використовується мутація, описана вище.

На Рис. 3.4 представлено діаграму класу *BaseSpecies*.

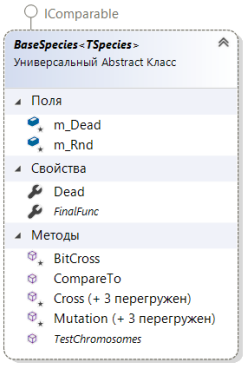


Рисунок 3.4 – UML діаграма класу *BaseSpecies*

3.1.2 Клас Population

Розглянемо клас популяції, де живуть, розмножуються і помирають види. Ми будемо додавати свої види в цей клас (точніше в масив видів, що знаходиться в цьому класі). Клас *Population* оголошений як *public class Population<TSpecies> where TSpecies:BaseSpecies<TSpecies>*.

Як видно, він має generic-параметр, який являє собою вид, який повинен бути похідним від BaseSpecies <TSpecies>

Почнемо з властивостей, які треба налаштувати перед початком роботи алгоритму:

- *MaxSize (Int32)* - максимальне число видів, яке може містити популяція. Це той розмір, до якого "урізається" популяція після схрещування. За замовчуванням 500.
- *CrossPossibility (Double)* - ймовірність схрещування. Вона повинна бути в інтервалі (0,0; 1.0]. Якщо ця умова не виконується, то випадає виняток *ArgumentOutOfRangeException*. За замовчуванням це значення встановлено в 0.95.
- *MutationPossibility (Double)* - ймовірність мутації. Вона повинна бути в інтервалі [0,0; 1.0). Якщо ця умова не виконується, то випадає виняток *ArgumentOutOfRangeException*.

В принципі, можна було б ввести ще ймовірність відбору, але в даному випадку вона вважається рівною 1.0. В іншому випадку треба було б видаляти види з популяції не всі підряд найгірші, а з певною ймовірністю. Але мертві види (нагадаю, що це види, у яких хромосоми не потрапляють в заданий інтервал або не задовольняє іншим умовам) треба видаляти в будь-якому випадку.

А тепер розглянемо публічні методи, які необхідно викликати.

public void Add (TSpecies species) - додати новий вид в популяцію. Ми повинні вручну додавати необхідну кількість видів. Перед початком роботи алгоритму треба, щоб в популяції було хоча б два види. Число видів в популяції може бути менше, ніж встановлене значення *MaxSize*. Якщо після схрещування розмір популяції менше *MaxSize*, то просто не видаляються найгірші види (навіть «мертві»).

Щоб отримати наступну популяцію, необхідно викликати метод *void NextGeneration()*. Робота цього методу може займати досить багато часу, тому краще всього його викликати з окремого потоку. Давайте подивимося що він робить.

Фрагмент коду методу *NextGeneration(),* який відповідає за оновлення популяції наведено на рис. 3.5.

```
/// <summary>
/// Оновити популяцію (отримати наступне покоління)
/// </summary>
virtual public void NextGeneration()
{
    if (m_Generation == 0 && m_Species.Count == 0)
    {
        throw new ZeroSizePopulationException();
    }

    // Спочатку схрестимо
    Cross();

    // Промутуємо і заодно перевіримо всі хромосоми
    foreach (TSpecies species in m_Species)
    {
        // Якщо треба мутувати з урахуванням ймовірност
        if (m_Rnd.NextDouble() <= m_MutationPossibility)
        {
            species.Mutation();
        }

        species.TestChromosomes();
    }

    // Відбераємо найживучіші види
    m_Species.Sort();
    Selection();

    m_Generation++;
}
```

Рисунок 3.5 – Реалізація методу оновлення популяції

Спочатку метод перевіряє, щоб при першому виклику методу (при нульовому поколінні) розмір популяції був би більше 2 (інакше нікого

схрещувати). Якщо це не так, то випадає виняток *SmallSizePopulationException*. Після цього відбувається схрещування видів, реалізація на рис. 3.6.

```
/// <summary>
/// Отримати нові види схрещуванням
/// </summary>
protected void Cross()
{
    // Розмір до початку поповнення популяції (щоб не схрещувати нові види,
    // які додаються в кінець списку)
    Int32 OldSize = m_Species.Count;

    // Номер пари для схрещувати виду
    Int32 Count = m_Species.Count;

    for (int i = 0; i < Count; ++i)
    {
        // Якщо треба схрещувати з урахуванням ймовірності
        if (m_Rnd.NextDouble() <= m_CrossPossibility)
        {
            // Додаємо в список виду, отриманий схрещуванням чергового виду і
            // виду з випадковим номером
            m_Species.Add(m_Species[i].Cross(m_Species[m_Rnd.Next(OldSize)]));
        }
    }
}
```

Рисунок 3.6 – Реалізація методу отримання нових видів схрещування

При схрещуванні перебираємо всі види і схрещуємо їх із установленою ймовірністю схрещування з іншими видами, які вибираються випадково.

Після схрещування відбувається мутація видів також із заданою вірогідністю, а після схрещування йде тест хромосом. Потім сортуємо види за значенням їх цільової функції. Метод *Sort* визначено в класі *BaseSpecies*. Відбір видів також відбувається просто:

```
/// <summary>
/// Провести відбір найбільш "живучих" видів
/// </summary>
protected void Selection()
{
    // Скільки видів треба видалити
    Int32 Count = m_Species.Count - m_MaxSize;

    for (Int32 i = 0; i < Count; ++i)
    {
        m_Species.RemoveAt(m_Species.Count - 1);
    }
}
```

Рисунок 3.7 – Реалізація методу відбору «найживучіших» видів

На Рис. 3.8 представлено діаграму класу *Population*, в якій можна побачити усі поля, властивості та методи властиві даному класу.

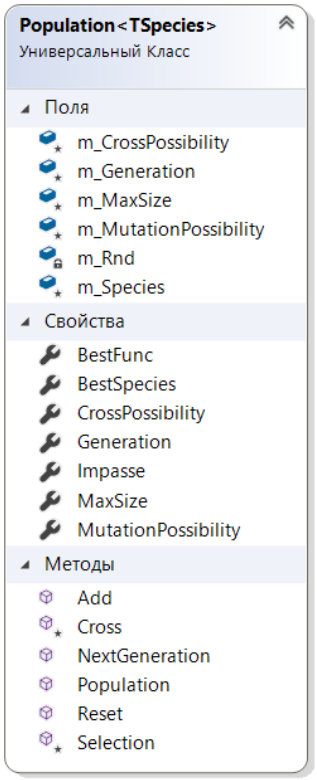


Рисунок 3.8 – UML діаграма класу *Population*

3.1.3 Клас BaseDoubleSpecies

Базовий клас *BaseSpecies* розроблявся таким чином, щоб якомога більше абстрагуватися від конкретної реалізації особин, від того що з себе представляють хромосоми. Це можуть бути дробові і цілі числа, масиви або екземпляри інших класів.

Однак, я використовую генетичний алгоритм щоб мінімізувати якусь математичну функцію, залежну від деякої кількості змінних дрібного типу. В цьому випадку хромосоми зручно представляти у вигляді масиву дрібних чисел, де кожен елемент масиву - це одна хромосома. При цьому будемо вважати, що розмір цього масиву залишається постійним для всіх особин протягом усього часу роботи алгоритму.

Саме для цієї мети і був створений клас *BaseDoubleSpecies*. Завдяки ньому можна істотно скоротити код, який реалізує алгоритм. Розглянемо цей клас.

Оголошення його досить заплутане:

```
public abstract class BaseDoubleSpecies<TSpecies> :
    BaseSpecies<BaseDoubleSpecies<TSpecies>>, ICloneable
    where TSpecies : BaseDoubleSpecies<TSpecies>
```

Рисунок 3.9 – Оголошення класу *BaseDoubleSpecies*

Тут, як і раніше, *TSpecies* - це конкретний тип особин.

Клас *BaseDoubleSpecies* є абстрактним, але він реалізує все, що повинен реалізувати клас особин, проте додає свій абстрактний метод, всередині якого і повинен відбуватися розрахунок цільової функції. Ця функція називається *CalcFinalFunc()*, вона повинна повертати тип *double*.

Щоб при кожному порівнянні особини з іншою особиною не доводилося обчислювати значення цільової функції, вона обчислюється один раз в конструкторі особини. Розраховане значення цільової функції зберігається в змінній *m_FuncVal*.

Також клас *BaseDoubleSpecies* містить наступні масиви:

- *m_Chromosomes* - хромосоми. Для доступу до цього члену передбачено властивість *Cromosomes*.
- *m_Intervals* - статичний масив класів *Interval*. За допомогою нього задаються інтервали зміни кожної хромосоми. Для доступу до цього масиву передбачено властивість *Intervals*. Цю властивість потрібно встановлювати до того як почне заповнюватися популяція. Якщо при створенні особини якась хромосома потрапляє в свій інтервал, то особина позначається як мертва. Розмір цього масиву завжди повинен дорівнювати розміру масиву *m_Chromosomes*. Більш того, при створенні особини з випадковим значенням хромосом, саме з розміру цього масиву обчислюється кількість хромосом.

Клас *BaseDoubleSpecies* реалізує всі етапи генетичного алгоритму, які не були реалізовані в базовому класі *BaseSpecies* такі як схрещування і мутації. Клас *BaseDoubleSpecies* має два конструктора:

```
public BaseDoubleSpecies ()

public BaseDoubleSpecies (double[] chromosomes)
```

Рисунок 3.10 – Конструктори клсу *BaseDoubleSpecies*

Перший з них створює особину з випадковими значеннями хромосом, що зручно використовувати при початковому заповненні популяції. Другий конструктор використовується всередині самого класу *BaseDoubleSpecies* для операцій схрещування і мутації.

На рис. 3.11 представлено діаграму класу *BaseDoubleSpecies.*

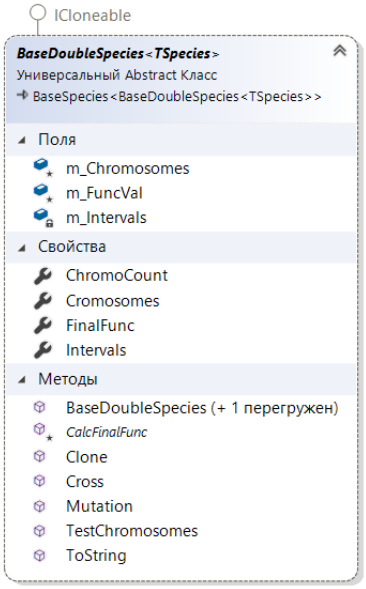


Рисунок 3.11 – UML діаграма класу *BaseDoubleSpecies*

3.1.4 Клас Interval

Клас *Interval* відповідає за допустиму область пошуку та містить наступні атрибути та методи:

- *MinValue* - мінімальне значення інтервалу.
- *MaxValue* - максимальне значення інтервалу.

- *Interval(double minval, double maxval)* – конструктор з двома параметрами.
- *IsInside (double val)* - чи потрапляє значення в заданий інтервал.

На рис. 3.12 представлено діаграму класу *Interval.*

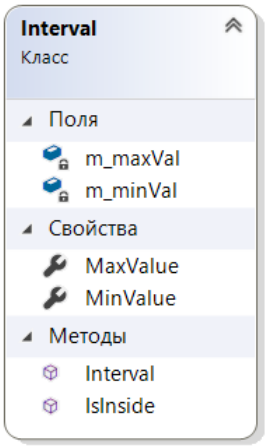


Рисунок 3.12 – UML діаграма класу *Interval*

3.1.5 Клас Analytics

Клас *Analytics* є корисним для спостереження за роботою генетичного алгоритму. Цей клас призначений для збереження кращої особини з кожного покоління. Клас *Analytics* призначений тільки для роботи з класами, похідними від *BaseDoubleSpecies*.

Користуватися класом *Analytics* дуже просто, для цього потрібно виконати наступні дії:

- Створити екземпляр класу *Analytics <TSpecies>*, де *TSpecies* - це клас, похідний від *BaseDoubleSpecies*.
- Перед запуском алгоритму очистити статистику за допомогою методу *Clear()*.
- Потім на кожній ітерації алгоритму (на кожному поколінні) викликати метод *Add()*, передаючи йому примірник кращої особини на даний момент.

- Після цього в процесі розрахунку можна звертатися до списку збережених особин через властивість *BestSpecies*.
- Крім того, використовуючи метод *ToString()*, можна зберегти історію у вигляді текстової таблиці, де кожен рядок буде відповідати одному поколінню, а кожен стовпець - одній дробовій хромосомі. Останній стовпець відповідає значенню цільової функції.

На рис. 3.13 представлено діаграму класу *Analytics*.

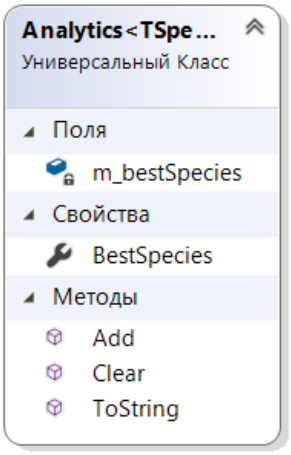


Рисунок 3.13 – UML діаграма класу *Analytics*

3.2 Розробка програмних компонентів FSS

Далі буде описано основні класи алгоритму косяка риб.

3.2.1 Клас FishPointUniversal

Клас *FishPointUniversal* - це клас, який містить статичні методи для знаходження евклідової відстані між особинами (рибами), методи основних операцій над особинами.

Величини, що відповідають за стан агентів (риб), є структурними типами n-мірної точки, для яких визначені наступні операції: додавання / віднімання двох точок, додавання / віднімання точки і числа, множення / ділення точки і

числа, а також операції порівняння точок. Будемо вважати дані величини за векторні типи даних.

Розберемо деякі методи детальніше:

- *FishPointUniversal(params double[] coords)* – констуктор, в який ми передаємо координати особини.
- *FishPointUniversal(int dimensionSize)* – конструктор, в який ми передаємо розмір популяції (косяка риб).
- *GetCoords()* – метод, який повертає масив з координатами особини.
- *EuclidDistance(FishPointUniversal p1, FishPointUniversal p2)* – статичний метод обчислення евклідової відстані між рибами, який нам знадобиться на стадії колективного плавання.
- *FishPointUniversal operator -(FishPointUniversal p1, FishPointUniversal p2)* – статичний метод, що виконує операцію віднімання векторів, які представляють собою рибин. Фрагмент коду наведено на рис. 3.14.

```
public static FishPointUniversal operator -(FishPointUniversal p1, FishPointUniversal p2)
{
    if (p1.vector.Length != p2.vector.Length)
        throw new IndexOutOfRangeException();
    else
    {
        double[] to_return = new double[p1.vector.Length];
        for (int i = 0; i < to_return.Length; i++)
        {
            to_return[i] = p1[i] - p2[i];
        }
        return new FishPointUniversal(to_return);
    }
}
```

Рисунок 3.14 – Реалізація методу *FishPointUniversal operator -*

- *FishPointUniversal operator +(FishPointUniversal p1, double number)*– статичний метод, що виконує операцію додавання векторів, які представляють собою рибин.
- *FishPointUniversal operator *(FishPointUniversal p1, double factor)* – статичний метод, що виконує операцію множення вектора на змінну *factor*. Даний метод нам знадобиться при підрахунку середнього зваженого індивідуальних рухів особин та при установці центру ваги системи.

- *FishPointUniversal operator /(FishPointUniversal p1, double divider)* – статичний метод, що виконує операцію множення вектора на змінну *divider*. Даний метод нам знадобиться при установці центру ваги системи та зменшенні величини кроку індивідуального пошуку.
- *IsInRegion(FishPointUniversal lowerBound, FishPointUniversal higherBound)* – метод, який повертає *true*, якщо особина в межах області пошуку, в іншому випадку – *false*.

На рис. 3.15 представлено діаграму класу *FishPointUniversal*.

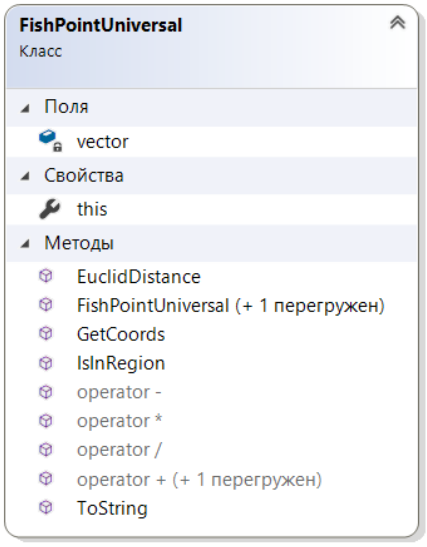


Рисунок 3.15 – UML діаграма класу *FishPointUniversal*

3.2.2 Клас FishUniversal

Клас *FishUniversal* – це клас, який містить методи, що реалізують основну логіку алгоритму. А саме він мітить методи, що реалізують усі стадії плавання риб, пошук максимального значення дельта-функції, пошук та знаходження барицентра системи, знаходження сумарної ваги всіх риб, годування риб та інші.

Розпочнемо з стадії міграції агентів (рибин), яка виконується поітераційно, і в кожній з ітерацій виконуються оператори двох груп:

- Оператори плавання, що забезпечують міграцію агентів в межах акваріума.
- Оператори годування, здатні фіксувати успіх дослідження тих чи інших областей акваріума.

Настав час поговорити про параметри, які має акваріум (іншими словами область пошуку), і його мешканців. Отже, ми ввели наступні змінні, характерні для всього акваріума в цілому:

- *populationSize* – розмір популяції (кількість риб в косяку).
- *iterationCount* – кількість ітерацій в стадії «Міграція агентів».
- *lowerBoundPoint, higherBoundPoint* – верхня і нижня межі пошуку.
- *individStepStart, individStepFinal* – задає початковий і кінцевий радіус пошуку їжі навколо агентів.
- *weightScale* – максимальна вага агента.

Це як раз ті самі параметри, які вводить користувач. З їхньою допомогою регулюється співвідношення точність / час роботи алгоритму.

Самі агенти характеризуються тільки двома величинами:

- *swimStatePos* – позиція агента в різних стадіях плавання.
- *weight* – поточна вага агента.

Ініціалізація агентів здійснюється за допомогою методу *InitializeData()*, який наведено нижче на рис. 3.16.

Як бачимо, при ініціалізації агентів ми випадковим чином вибираємо положення агентів у допустимій області.

Далі наводжу основні дані для ітерацій:

- *individStep* – значення індивідуального кроку зміщення на даній ітерації.
- *individDeltaFitnessMax* – максимальна зміна фітнес-функції.
- *instinctAverage* – середнє зважене після інстинктивної стадії.

- *instinctSumWeightOld* – вага риб після інстинктивного плавання в попередній ітерації.
- *instinctSumWeightNew* – вага риб після інстинктивного плавання в поточній ітерації.
- *willStep* – крок вольового зміщення.
- *barycentre* – центр тяжіння риб, використовується в вольовому плаванні.
- *individDeltaPoint* – зміщення після індивідуальної стадії.
- *individDeltaFitness* – зміна функції після індивідуального плавання.
- *weight* – вагу рибки, за замовчуванням половина від максимуму *weightScale* / 2.

```
public static void InitializeData(UserFunctions.FuncTemplate _function, int _dimensionSize, out FishUniversal[] _fishes,
                                  int _popsize, int _iterationCount, FishPointUniversal _lowerBound, FishPointUniversal _higherBound,
                                  double _individStepStart = 0.5, double _individStepFinish = 0.1, double _weightScale = 10)
{
    funcTemplate = _function;
    dimensionSize = _dimensionSize;
    populationSize = _popsize;
    iterationCount = _iterationCount;
    lowerBoundPoint = _lowerBound;
    higherBoundPoint = _higherBound;
    individStepStart = _individStepStart;
    individStepFinal = _individStepFinish;
    individStep = individStepStart;
    weightScale = _weightScale;
    instinctSumWeightOld = populationSize * weightScale / 2;
    _fishes = new FishUniversal[populationSize];
    for (int fishIndex = 0; fishIndex < _fishes.Length; fishIndex++)
    {
        _fishes[fishIndex] = new FishUniversal();
        // Ініціалізуємо положення риб виходячи з арності функції
        for (int swimState = 0; swimState < 4; swimState++)
            _fishes[fishIndex].swimStatePoint[swimState] = new FishPointUniversal(dimensionSize);
        for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
            // Вибираємо початкове положення для 0-й стадії
            _fishes[fishIndex].swimStatePoint[0][dimensionIndex] = lowerBoundPoint[dimensionIndex] +
                (higherBoundPoint[dimensionIndex] - lowerBoundPoint[dimensionIndex]) * randGen.NextDouble();
    }
}
```

Рисунок 3.16 – Реалізація методу *InitializeData()*

Індивідуальні дані для кожної рибки містяться в масиві *swimStatePoint = new FishPointUniversal [4]*, де:

- 0 – початок індивідуального плавання.
- 1 – положення після індивідуального плавання.
- 2 – після інстинктивного плавання.
- 3 – остаточна позиція ітерації (після колективного плавання).

Отже, як вже було сказано, в першу чергу проводиться ініціалізація всієї популяції: випадковий вибір позиції агента в межах акваріума (*swimStatePos [0])* і установка ваги, яка дорівнює половині від максимального (*weight = weightScale* / 2) для всіх риб.

Далі починається основний цикл алгоритму, який характеризує стадію «Міграція агентів до джерела їжі». Як критерій зупинки в нашому випадку використовується величина кількість ітерацій (*iterationCount*).

Потім настає індивідуальна стадія плавання агентів, яка реалізована у методі *IndividSwim()*, фрагмент коду на рис. 3.17. Вона характеризується тим, що всі риби в деякій області навколо себе (*individStep*) намагаються знайти краще значення функції. Якщо їм це вдається, то цей крок фіксується. В іншому випадку вважаємо, що цього переміщення не було. Як видно з рис. 3.17 було введено параметр α, який розпадається експоненціально разом з повтореннями. Кожного разу, коли риба намагається перейти в положення, яке не покращує її придатність, вибирається випадкове число randomNum, і якщо воно менше за α, рух дозволяється.

```
/// <summary>
/// Стадія індивідуального плавання
/// </summary>
public void IndividSwim()
{
    currentIteration++;
    double alpha = 0.8 * Math.Pow(Math.E, -0.007 * currentIteration);
    Random m_Rnd = new Random();
    double randomNum = m_Rnd.Next(0, 1);
    // Даємо рибкам шанс зробити переміщення
    for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
        this.swimStatePoint[1][dimensionIndex] = this.swimStatePoint[0][dimensionIndex] + (-1 + randGen.NextDouble() * 2) * individStep;
    //Delta X и F(x)
    this.individDeltaPoint = this.swimStatePoint[1] - this.swimStatePoint[0];
    this.individDeltaFitness = Fitness(this.swimStatePoint[1].GetCoords()) - Fitness(this.swimStatePoint[0].GetCoords());
    // Перевіряємо переміщення на ефективність
    if (!this.swimStatePoint[1].IsInRegion(lowerBoundPoint, higherBoundPoint) | this.individDeltaFitness > 0 | randomNum > alpha)
    {
        // Якщо переміщення неефективне або неможливе, то вважаємо, що його не було
        this.swimStatePoint[1] = this.swimStatePoint[0];
        this.individDeltaPoint = new FishPointUniversal(dimensionSize);
        this.individDeltaFitness = 0;
    }
}
```

Рисунок 3.17 – Реалізація методу *IndividSwim()*

Тепер необхідно закріпити успіх в індивідуальній стадії плавання. Для цього використовується характеристика «вага». Вона дорівнює зміні функції пристосованості для даного агента до і після індивідуальної стадії, нормованого максимальним значенням функції серед популяції. Взагалі кажучи, це є

відмінною рисою даного алгоритму, так як нам не треба запам'ятовувати кращих агентів на попередніх ітераціях.

Після цього риби здійснюють наступну стадію плавання - інстинктивно-колективну, яка реалізована у методі *InstinctSwim()*, фрагмент коду на рис. 3.18. Для всього косяка риб вираховується величина «загальний крок міграції». Сенс її наступний: на кожного агента впливає вся популяція в цілому, при цьому вплив окремого агента пропорційно його успіхам в індивідуальній стадії плавання. Потім вся популяція зміщується на нараховану величину *instinctAverage*.

```
/// <summary>
/// Стадія інстинктивного плавання
/// </summary>
public void InstinctSwim()
{
    this.swimStatePoint[2] = this.swimStatePoint[1] + instinctAverage;
}
```

Рисунок 3.18 – Реалізація методу *InstinctSwim()*

Перед наступною операцією плавання необхідно виконати проміжні дії, а саме: підрахувати центр тяжіння всього косяка (метод *SearchBarycentre(FishUniversal[] fishes)*). Перш за все, баріцентр косяка розраховується на основі положення *swimStatePoint[2]* та ваги *weight* кожної риби.

Ми, а точніше, риби, перейшли до останньої стадії плавання: колективно-вольової, яка реалізована у методі *CollectiveSwim()*, фрагмент коду на рис. 3.19. Тут необхідно дізнатися, як змінилася вага популяції в порівнянні з попередньою ітерацією. Якщо вона збільшилася, значить популяція наблизилася до області мінімуму функції, тому необхідно звузити коло пошуку, тим самим виявляються інтесифікаційні властивості. І навпаки: якщо вага косяка зменшилась, значить агенти шукають мінімум не в тому місці, тому необхідно змінити напрямок траєкторії і проявити диверсифікаційні властивості.

Величина *swimStatePoint[3]* в такій формулі відповідає за крок вольового зміщення. Рекомендується використовувати значення, в 2 рази більше індивідуального кроку пошуку. Оператор *euclidD* вираховує відстань між двома точками в евклідовому просторі.

```
/// <summary>
/// Стадія колективного плавання
/// </summary>
public void CollectiveSwim()
{
    double euclidD = FishPointUniversal.EuclidDistance(this.swimStatePoint[2], barycentre);
    if (euclidD == 0)
    {
        this.swimStatePoint[3] = this.swimStatePoint[2];
    }
    else if (instinctSumWeightNew > instinctSumWeightOld)
    {
        this.swimStatePoint[3] = this.swimStatePoint[2] - (this.swimStatePoint[2] - barycentre) * willStep * randGen.NextDouble() / euclidD;
    }
    else
    {
        this.swimStatePoint[3] = this.swimStatePoint[2] + (this.swimStatePoint[2] - barycentre) * willStep * randGen.NextDouble() / euclidD;
    }
}
```

Рисунок 3.19 – Реалізація методу *CollectiveSwim()*

На рис. 3.20 представлено діаграму класу *FishUniversal.*

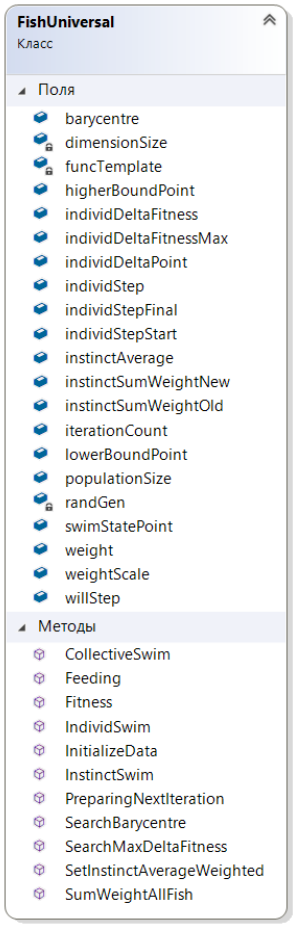


Рисунок 3.20 – UML діаграма класу *FishUniversal*

3.2.3 Клас UserFunctions

Клас *UserFunctions* – це клас, в якому ініціюються багатомірні функції, для яких ми шукатимемо екстремуми.

Функції знаходяться у списку *List<Tuple<FuncTemplate, int, string>> FunctionList*, де *Tuple* – це кортеж, у нашому випадку трьохкратний. Розглянемо детальніше елементи кортежу:

- *FuncTemplate* – багатомірна функція.
- *int* – кількість змінних.
- *string* – рядковий запис багатомірної функції.

У даному класі є ще один метод – *SelectFunction(int number)*, який вибирає функцію за її номером зі списку, а повертає кортеж, що складається з функції і її кількості аргументів (якщо такої функції немає, то повертається null).

На рис. 3.21 представлено діаграму класу *UserFunctions*.

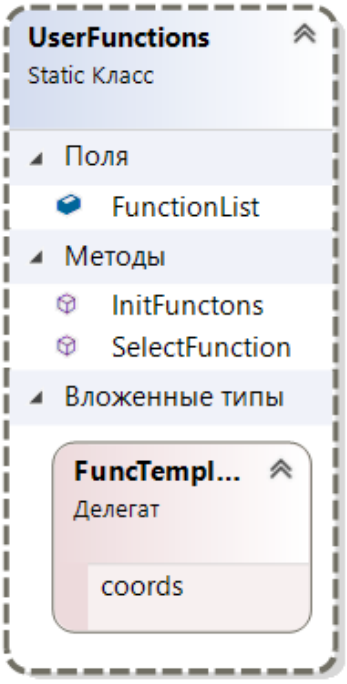


Рисунок 3.21 – UML діаграма класу *UserFunctions*

3.3 Розробка програмних компонентів DatabaseConnection

Далі буде описано основні класи, що відповідають за підключення до БД.

3.3.1 Клас DBSQLServerUtils

Клас *DBSQLServerUtils* призначений для налагодження зв'язку з БД. Даний клас містить один метод *GetDBConnection*, який приймає наступні параметри:

- *datasource* – назва серверу.
- *database* – назва БД.
- *username* – ім'я користувача.
- *password* – пароль користувача.

Метод *GetDBConnection* повертає об'єкт типу *SqlConnection*.

На рис. 3.22 представлено діаграму класу *DBSQLServerUtils.*

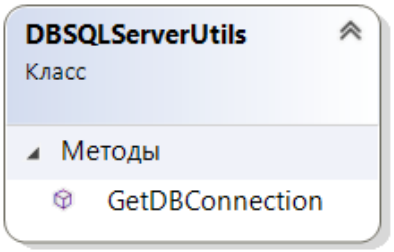


Рисунок 3.22 – UML діаграма класу *DBSQLServerUtils*

3.3.2 Клас DBUtils

Даний клас містить метод *GetDBConnection*, який не приймає жодного параметра. Натомість в методі вказуються основні дані (*datasource*, *database*, *username*, *password*), необхідні для підключення до БД. Реалізація методу представлена на Рис. 3.23.

```
public static SqlConnection GetDBConnection()
{
    string datasource = @"DESKTOP-4UTFR7T\MS_SQL_SERVER";

    string database = "ResearchOfEvolutionaryAlgorithms";
    string username = "sa";
    string password = "12345";

    return DBSQLServerUtils.GetDBConnection(datasource, database, username, password);
}
```

Рисунок 3.23 – Реалізація методу *GetDBConnection*

На рис. 3.24 представлено діаграму класу *DBUtils.*

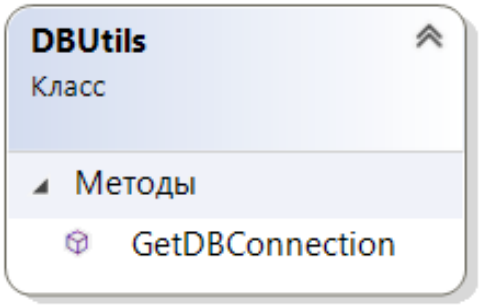


Рисунок 3.24 – UML діаграма класу *DBUtils*

ВИСНОВОК ДО РОЗДІЛУ 3

В результаті проведеної роботи щодо розробки програмного забезпечення системи пошуку екстремумів багатомірних функцій на основі еволюційних алгоритмів (GA та FSS) можна виокремити наступне:

- Реалізовано раніше обгрунтовані (у Розділі II) еволюційні алгоритми GA та модифікований FSS.
- Наведено UML діаграми розроблених класів.
- Детально описано атрибути, методи та властивості класів.
- Були продемонстровані фрагменти коду у вигляді скріншотів розроблюваної системи.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

Базуючись на розробленій системі пошуку екстремуму багатомірної функції метою даного розділу є показати результати її роботи. Також потрібно провести порівняльну характеристику розроблених алгоритмів, дослідити ефективність роботи кожного з них на прикладі задачі глобальної оптимізації.

4.1 Дослідження ефективності роботи модифікованого алгоритму FSS

Вплив розміру популяції на час виконання алгоритму косяка риб наведений у табл. 4.1.

Таблиця 4.1 – Вплив розміру популяції на час виконання алгоритму

Розмір популяції	Час виконання алгоритму, (с)
5	0,013
10	0,008
25	0,007
35	0,007
50	0,007
75	0,008
100	0,005
200	0,01
500	0,012
1000	0,02

Дослідження впливу розміру популяції на час виконання алгоритму здійснювалося на прикладі пошуку екстремуму функції $y = 0{,}1(x^2 + z^2) + 10$, найбільш ефективні значення представлені в таблиці 4.1.1. З неї видно, що оптимальний розмір популяції - 100 особин. Можна помітити, що найбільший час виконання алгоритму становить при малій популяції – 5 особин, та великій – 1000 та 500 особин.

Для вивчення залежності швидкості пошуку рішень від значення максимальної ваги агента була обрана функція $y = 3x^2 + xz + 2z^2 - x - 4z$. Результати досліджень відображені в таблиці 4.2.

Таблиця 4.2 – Вплив максимальної ваги агента на час виконання алгоритму.

Вага агента	Час виконання алгоритму, (с)
5	0,012
10	0,012
25	0,012
35	0,012
50	0,011
75	0,011
100	0,013
200	0,015
500	0,016
1000	0,014

Як видно з таблиці 4.2 оптимальна вага агента – 75.

У таблиці 4.3. показано вплив кількості ітерацій на точність і час виконання алгоритму для функції $y = 0{,}1(x^2 + z^2) + 10$.

Як видно з таблиці 4.3. точність виконання алгоритму збільшується з кількістю ітерацій. Щодо часу виконання алгоритму, то він також збільшується з кількістю ітерацій.

Таблиця 4.3 – Вплив кількості ітерацій на точність і час виконання алгоритму

Кількість ітерацій	Точність виконання алгоритму, (%)	Час виконання алгоритму, (с)
15	99,9412	0,007
25	99,97482	0,045
35	99,987	0,055
50	100	0,056
75	100	0,033
100	100	0,038
200	100	0,045
350	100	0,11
500	100	0,115

Дослідження роботи алгоритму FSS показали, що ефективність алгоритму залежить від обраних параметрів, чим краще вони підібрані, тим точнішим є результат і меншим є час виконання алгоритму.

4.2 Дослідження ефективності роботи алгоритму GA

Вплив розміру популяції на час виконання алгоритму GA.

Дослідження впливу розміру популяції на час виконання алгоритму здійснювалося на прикладі пошуку екстремуму функції $y = 0{,}1(x^2 + z^2) + 10$, найбільш ефективні значення представлені в таблиці 4.4. З неї видно, що оптимальний розмір популяції - 100 особин.

Для вивчення залежності швидкості пошуку рішень від значення імовірності схрещування хромосом була обрана функція $y = 3x^2 + xz + 2z^2 - x - 4z$. Результати досліджень відображені в таблиці 4.5.

Таблиця 4.4 – Вплив розміру популяції на час виконання алгоритму

Розмір популяції	Час виконання алгоритму, (с)
5	0,17
10	0,091
25	0,032
35	0,032
50	0,026
75	0,025
100	0,023
200	0,026
500	0,037
1000	0,077

Таблиця 4.5 – Вплив імовірності схрещування хромосом на час виконання алгоритму

Імовірність схрещування, (%)	Час виконання алгоритму, (с)
95	0,051
90	0,061
85	0,046
80	0,046
75	0,053
70	0,09
65	0,098
60	0,165

Як видно з таблиці 4.5 що оптимальна імовірність схрещування хромосом для даної функції – 80-85%.

У таблиці 4.6. показано вплив кількості ітерацій на точність і час виконання алгоритму для функції $y = 0{,}1(x^2 + z^2) + 10$.

Таблиця 4.6 – Вплив кількості ітерацій на точність і час виконання алгоритму

Кількість ітерацій	Точність виконання алгоритму, (%)	Час виконання алгоритму, (с)
15	99,9998	0,025
25	100	0,03
35	100	0,054
50	100	0,073
75	100	0,094
100	100	0,121
200	100	0,267
350	100	0,399
500	100	0,584

Як видно з таблиці 4.6. точність виконання алгоритму та час збільшується з кількістю ітерацій.

Дослідження роботи алгоритму GA показали, що ефективність алгоритму залежить від обраних параметрів, чим краще вони підібрані, тим точнішим є результат і меншим є час виконання алгоритму.

4.3 Порівняльний аналіз ефективності роботи алгоритмів FSS та GA

Для порівняльного аналізу ефективності роботи алгоритму пошуку косяком риб в рамках виконання завдання глобальної оптимізації було обрано генетичний алгоритм.

У таблиці 4.7 наведені результати обчислення значень оптимуму функції Де Джонга $f(x) = \sum_{i=1}^{n} x_i^2, i = 1{:}\, n$, з різною кількістю змінних.

Таблиця 4.7 – Точність знайденого рішення досліджуваними алгоритмами для функції Де Джонга

Кількість змінних / Точність, (%)	3	4	5
FSS	99,988	99,99	99,993
GA	99,999	100	100

На основі даних таблиці 4.7 можна зробити висновок, що обидва алгоритми є точними, адже точність складає практично 100%, а у генетичного алгоритму таки становить 100%.

У таблиці 4.8 і 4.9 представлені залежності кількості ітерацій, необхідних для знаходження оптимуму функцій Розенброка $f(x) = \sum_{i=1}^{n-1} 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2, i = 1{:}\,n-1$ і Растрігіна $f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)), i = 1{:}\,n$ відповідно.

Таблиця 4.8 – Кількість ітерацій для функції Розенброка з різною кількістю змінних

Кількість змінних / Кількість ітерацій	3	4	5
FSS	170	250	330
GA	130	370	650

Таблиця 4.9 – Кількість ітерацій для знаходження оптимуму функції Растрігіна з різною кількістю змінних

Кількість змінних / Кількість ітерацій	3	4	5
FSS	150	170	200
GA	210	390	735

За даними таблиць 4.8 та 4.9 ми спостерігаємо, що алгоритму косяка риб необхідно менше ітерацій при знаходженні екстремуму багатомірної функції, розмірність якої більше трьох змінних.

У таблиці 4.10 представлені залежності часу виконання алгоритму, необхідного для знаходження оптимуму функції Швефеля $f(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|}), i = 1{:}n.$

Таблиця 4.10 – Час необхідний для знаходження оптимуму функції Растрігіна з різною кількістю змінних

Кількість змінних / Час виконання, (с)	3	4	5
FSS	0,18	0,29	0,404
GA	0,155	0,37	0,64

Як видно з таблиці 4.10, алгоритм FSS є швидшим, ніж GA, тобто рішення знаходить швидше алгоритм FSS.

При аналізі роботи еволюційних алгоритмів для знаходження оптимуму функцій видно, що FSS та GA є універсальними методами для пошуку екстремуму незалежно від складності функцій. Для пошуку оптимуму досить простих функцій краще використовувати класичні методи, так як вони в даному випадку працюють трохи швидше еволюційних алгоритмів. При вирішенні задачі глобальної оптимізації для функції зі складним ландшафтом рекомендується використовувати FSS, так як класичні алгоритми часто призводять до неправильного вирішення, а генетичний алгоритм, як можна помітити, працює повільніше. Завдяки модифікації алгоритму FSS, він сходиться і на функціях з гладкою поверхнею. Також алгоритму FSS необхідна менша кількість ітерацій для функцій багатьох змінних, натомість на функціях з малою кількістю змінних GA має перевагу в кількості ітерацій. Точність обчислення обох алгоритмів є практично стовідсотковою. Варто зазначити що на час виконання алгоритму та точність знайденого рішення впливають обрані параметри, які потрібно намагатись підібрати якомога оптимальніше, опираючись на проведені вище дослідження.

4.4 Огляд інтерфейсу програми

На рисунку 4.1 зображений розроблений програмний інтерфейс користувача для алгоритму FSS. У полях вводу необхідно ввести наступні параметри алгоритму:

- Досліджувана функція – функція для якої відбуватиметься пошук екстремума.
- Верхня границя – верхня межа області пошуку рішення.
- Нижня границя – нижня межа області пошуку рішення.
- Кількість ітерацій
- Розмір популяції – кількість агентів, які здійснюватимуть пошук рішення.
- Початковий індивідуальний крок – початковий крок, з яким рухатимуться риби, починаючи з першої ітерації.
- Кінцевий індивідуальний крок - початковий крок, з яким рухатимуться риби, на останніх ітераціях.
- Максимальна вага риби

На рисунку 4.2 зображений розроблений програмний інтерфейс користувача для алгоритму GA. У полях вводу необхідно ввести наступні параметри алгоритму:

- Досліджувана функція – функція для якої відбуватиметься пошук екстремума.
- Верхня границя – верхня межа області пошуку рішення.
- Нижня границя – нижня межа області пошуку рішення.
- Кількість ітерацій
- Розмір популяції – кількість агентів, які здійснюватимуть пошук рішення.

- Імовірність схрещування – імовірність, з якою відбувається схрещування хромосом (за замовчуванням встановлено 90%).
- Імовірність мутації – імовірність, з якою відбувається мутація хромосом (за замовчуванням встановлено 10%).
- Кількість хромосом – являє собою кількість змінних функції.

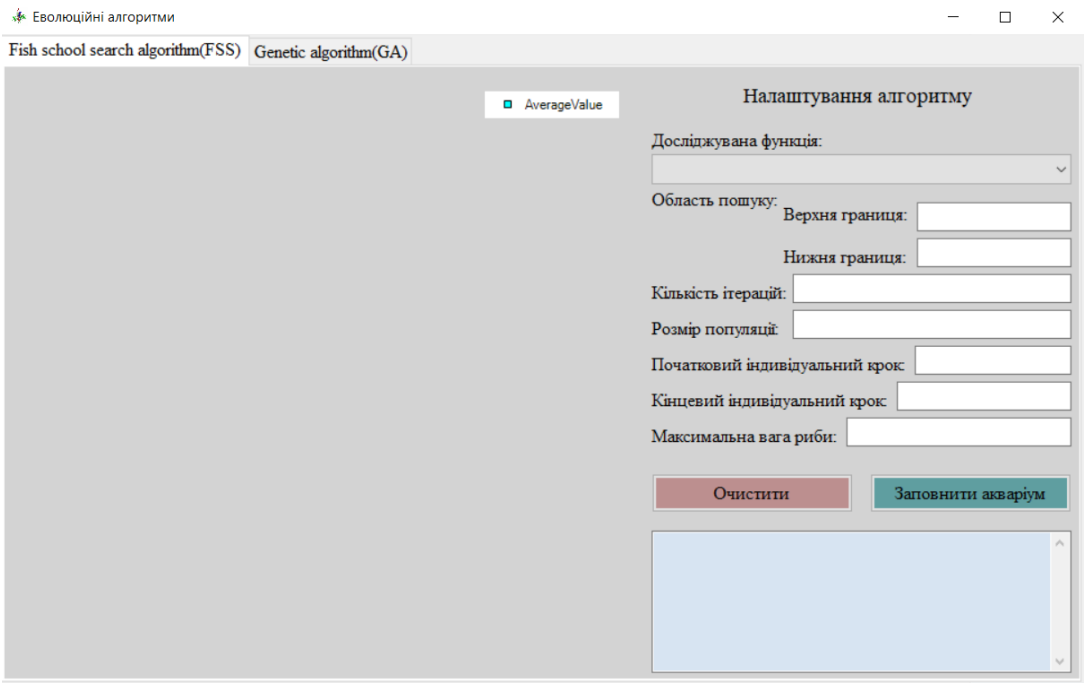


Рисунок 4.1 – Програмний інтерфейс користувача для алгоритму FSS

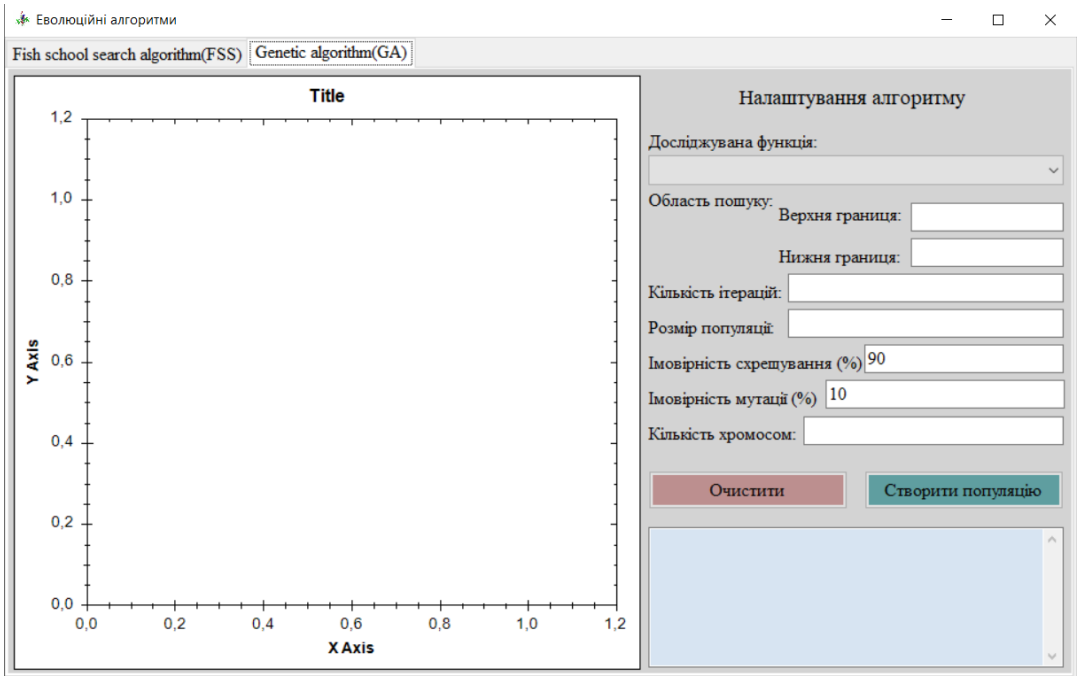


Рисунок 4.2 – Програмний інтерфейс користувача для алгоритму GA

Параметри, які вводить користувач проходять валідацію на коректність вводу. Якщо хоча б один із параметрів не буде введено, або ж він буде введено некоректно, то появиться повідомлення з попередженням «Перевірте чи введено всі дані!» або «Перевірте чи правильно введені всі дані!», а відповідне поле, де некоректно введені дані підсвічуватиметься червоним кольором.

Рисунок 4.3 – Сповіщення некоректного вводу параметрів

Якщо всі параметри введено вірно, то програма розпочне свою роботу. Результатом буде знайдений мінімум функції, час виконання, а також графік значень мінімумів на кожній ітерації.

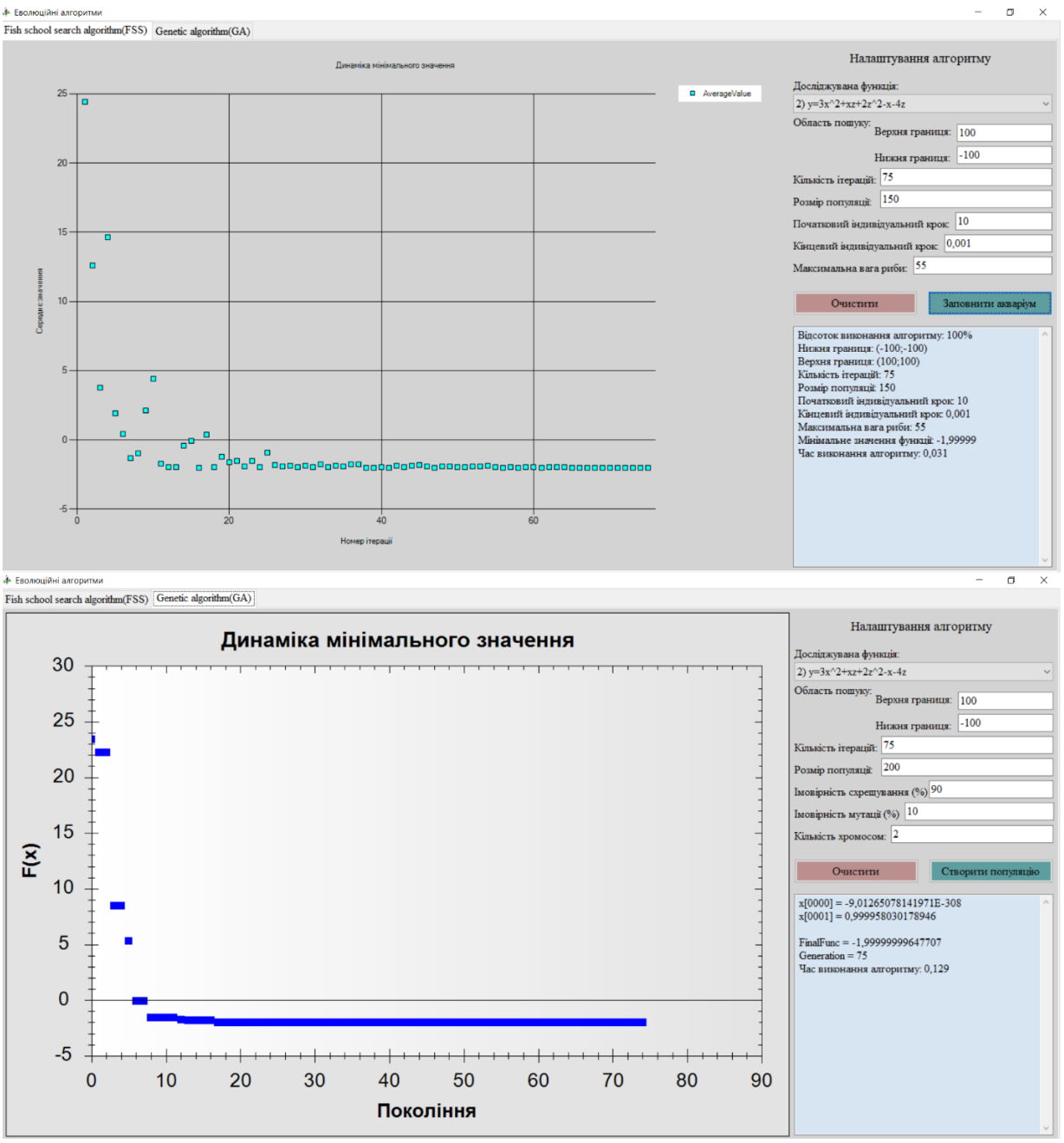


Рисунок 4.4 – Результати роботи програми

4.5 Огляд бази даних

База даних *ResearchOfEvolutionaryAlgorithms* містить дві таблиці:

- *GA_parameters* – таблиця, в якій зберігаються параметри введені користувачем, а також результат роботи алгоритму GA та час виконання.
- *FSS_parameters* – таблиця, в якій зберігаються параметри введені користувачем, а також результат роботи алгоритму FSS та час виконання.

Таблиці містять наступні поля, зображені на рисунку 4.5

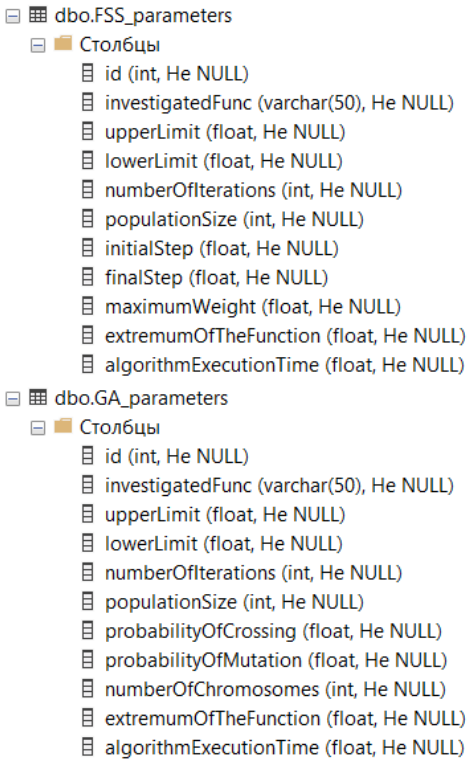


Рисунок 4.5 – Таблиці БД з полями

Вигляд таблиці *GA_parameters* після запуску програми можна побачити на рисунку 4.6

Результаты | Сообщения

	id	investigatedFunc	upperLimit	lowerLimit	numberOfIterations	populationSize	probabilityOfCrossing	probabilityOfMutation	numberOfChromosomes	extremumOfTheFunction	algorithmExecutionTime
1	823	0) y=-x(10-x)	10	-10	100	200	0,9	0,1	1	-25	0,168999999761581
2	824	0) y=-x(10-x)	10	-10	44	200	0,9	0,1	1	-24,999927520752	0,0860000029206276
3	825	2) y=3x^2+xz+2z^2-x-4z	100	-100	44	200	0,9	0,1	2	-1,99999988079071	0,0900000035762787
4	826	4) y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))	100	-100	44	200	0,8	0,2	3	-29,9958629608154	0,0850000008940697
5	827	4) y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))	100	-100	440	200	0,8	0,2	3	-29,9999942779541	0,778999984264374
6	828	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	100	-100	440	200	0,8	0,2	4	100,003913879395	0,828999996185303
7	829	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	107	-109	1000	200	0,8	0,2	4	100,000465393066	0,805999994277954
8	830	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	10	-10	10	500	0,8	0,2	4	100	0,0410000011324883
9	817	15) Функція Швефеля (3 змінні)	500	-500	100	100	0,9	0,1	3	-1260,04931640625	0,180000007152557
10	818	15) Функція Швефеля (3 змінні)	500	-500	500	400	0,9	0,1	3	-1262,193359375	0,398000001907349
11	819	16) Функція Швефеля (4 змінні)	500	-500	500	400	0,9	0,1	4	-1682,94079589844	0,509999990463257
12	820	16) Функція Швефеля (4 змінні)	500	-500	1000	400	0,9	0,1	4	-1682,88598632813	0,941999971866608
13	821	17) Функція Швефеля (5 змінних)	500	-500	1000	400	0,9	0,1	5	-2103,67651367188	0,129999995231628
14	822	17) Функція Швефеля (5 змінних)	500	-500	400	500	0,7	0,3	5	-2103,671875	0,458999991416931

Рисунок 4.6 – Таблиця *GA_parameters*

Вигляд таблиці *FSS_parameters* після запуску програми можна побачити на рисунку 4.7.

Результаты | Сообщения

	id	investigatedFunc	upperLimit	lowerLimit	numberOfIterations	populationSize	initialStep	finalStep	maximumWeight	extremumOfTheFunction	algorithmExecutionTime
1	486	0) y=-x(10-x)	100	-100	100	100	10	1	50	-24,9999980926514	0,046000000089407
2	487	0) y=-x(10-x)	100	-100	100	100	10	0,001	50	-24,9999904632568	0,0410000011324883
3	488	1) y=0,1(x^2+z^2)+10	100	-100	100	100	10	0,001	50	10,0000066757202	0,0359999984502792
4	489	2) y=3x^2+xz+2z^2-x-4z	1000	-1000	100	1000	10	0,001	50	2,61325216293335	0,250999987125397
5	490	2) y=3x^2+xz+2z^2-x-4z	100	-100	100	100	10	0,01	50	-1,99974954128265	0,0370000004768372
6	491	2) y=3x^2+xz+2z^2-x-4z	100	-100	100	100	10	0,01	50	-1,99998688697815	0,0410000011324883
7	492	3) y=x^2-xz+z^2+3x-2z+1	100	-100	100	100	10	0,01	50	-1,33333218097687	0,0350000001490116
8	493	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	100	-100	100	100	10	0,01	50	100,000183105469	0,0480000004172325
9	494	8) Функція Де Джонга (5 змінних)	10	-10	100	100	1	0,01	50	7,08668085280806E-05	0,063000001013279
10	495	9) Функція Розенброка (3 змінні)	10	-10	100	100	1	0,01	50	0,459905475378037	0,0560000017285347
11	496	9) Функція Розенброка (3 змінні)	10	-10	400	100	1	0,01	50	0,106386631727219	0,193000003695488
12	497	9) Функція Розенброка (3 змінні)	10	-10	400	100	1	0,001	50	0,0854343846440315	0,188999995589256
13	498	8) Функція Де Джонга (5 змінних)	10	-10	400	100	1	0,001	50	9,68388349065208E-07	0,150999993085861
14	499	4) y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))	10	-10	400	100	1	0,001	50	-24,6504001617432	0,155000001192093
15	500	0) y=-x(10-x)	10	-10	400	100	1	0,001	50	-24,9999332427979	0,0750000029802322

Рисунок 4.7 – Таблиця *FSS_parameters*

4.6 Тестування програми

Щоб перевірити коректну роботу еволюційних алгоритмів FSS та GA було написано тести за допомогою бібліотеки MSTest. Робота алгоритмів перевірялась на різних багатомірних функція, як складних та і простих, а також з різними параметрами алгоритмів.

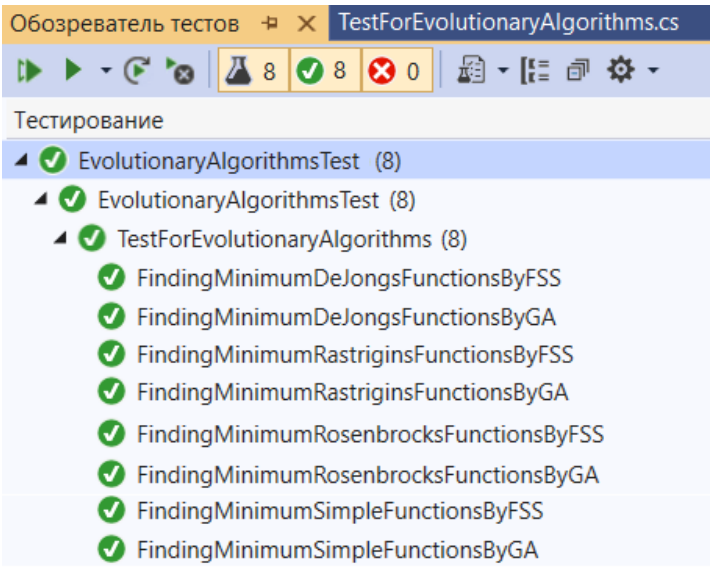


Рисунок 4.8 – Тестування роботи алгоритмів

Як видно з Рисунка 4.8 алгоритми FSS та GA працюють коректно як на функціях з простим ландшафтом, так і на функціях з складним ландшафтом, таких як функція Де Джогана, Растрігіна та Розенброка.

4.7 Рекомендації щодо розвитку та вдосконалення додатка

У подальшому для покращення і поглиблення функціональності можна додати декілька напрямків змін. По-перше, можна додати опцію пошуку не тільки глобального мінімуму, але й максимуму. По-друге, можна додати опцію пошуку не тільки глобальних екстремумів, але й локальних. Також іншим напрямком розширення функціональності застосунку може стати збільшення кількості алгоритмів, що використовуються для обчислень. Якщо ж говорити про розширення з точки зору технологій, можна, наприклад, додати можливість шифрування даних, які відправляються у базу даних. У будь-якому разі, варто зазначити, що завдяки тому, що у нас є інтеграційні тести, буде легко вводити нові функції, при цьому точно впевнюючись, що старі не ламатимуться.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі представлений готовий застосунок, розроблюваний протягом всієї дипломної роботи. Надано загальні вказівки щодо використання застосунка, а також було розглянуто випадки неправильного користування і до яких ситуацій це може призвести.

Було проведено дослідження ефективності роботи алгоритмів GA та FSS, досліджувались оптимальні параметри для алгоритмів на прикладі різних багатомірних функцій.

Далі були наведені тест-кейси тестування, які є дуже важливими для подібних застосунків, адже саме вони даватимуть нам впевненість у тому, що бізнес-логіка працює, як потрібно.

В кінці цього розділу пропонуються можливі ідеї щодо розширення та вдосконалення функціональних можливостей програми, які можуть бути реалізовані в майбутньому.

ВИСНОВКИ

Під час виконання дипломної роботи було створено систему для пошуку екстемумів багатомірних функцій з різною складністю ландшафту на основі еволюційних алгоритмів. Дана система надає можливість своїм користувачам за допомогою настільного застосунку знаходити глобальні екстремуми багатомірних функцій.

У першому розділі були розглянуті задачі оптимізації та методи їх вирішення. Також у першому розділі було розглянуто наступні еволюційні алгоритми: генетичний алгоритм, алгоритм світлячків, алгоритм бур'янистої оптимізації, зозулин пошук, алгоритм заснований на поведінці кажанів та алгоритм косяка риб. Були розглянуті схеми їх роботи, а також принцип роботи кожного із цих алгоритмів. Були встановлені їх недоліки та переваги.

У другому розділі були описані детально алгоритми FSS та GA, а також розглянуті технології, що будуть використані для написання системи – об’єкта розробки. Також було обґрунтовано, навіщо ці технології використовуватимуться та які переваги вони надаватимуть при їх використанні. Також у другій частині цього розділу обговорюються платформи, фреймворки та бібліотеки, які будуть використані для написання вихідного коду.

У третьому розділі було реалізовано раніше обгрунтовані (у Розділі II) еволюційні алгоритми GA та FSS, здійснено модифікацію алгоритму FSS, наведено UML діаграми розроблених класів, детально описано атрибути, методи та властивості класів. Були продемонстровані фрагменти коду у вигляді скріншотів розроблюваної системи.

У четвертому розділі ми провели детальний аналіз розробленої системи. У розділі представлений готовий застосунок, розроблюваний протягом всієї дипломної роботи. Надано загальні вказівки щодо використання застосунка, а

також було розглянуто випадки неправильного користування і до яких ситуацій це може призвести. Було проведено дослідження ефективності роботи алгоритмів GA та FSS, досліджувались оптимальні параметри для алгоритмів на прикладі різних багатомірних функцій. Також були наведені тест-кейси тестування, які є дуже важливими для подібних застосунків. В розділу пропонуються можливі ідеї щодо розширення та вдосконалення функціональних можливостей програми, які можуть бути реалізовані в майбутньому.

Слід також зазначити, що під час розробки застосунку були враховані всі зазначені на початку вимоги, яким він мав відповідати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глобальная оптимизация (Overall optimization) [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.loginom.ru/articles/overall-optimization.html.
2. What Is Global Optimization? [Електронний ресурс] // MathWorks – Режим доступу до ресурсу: https://www.mathworks.com/help/gads/what-is-global-optimization.html.
3. Christodoulos A. F. State of the Art in Global Optimization / A. F. Christodoulos, M. P. Panos // Computational Methods and Applications / A. F. Christodoulos, M. P. Panos. – Dordrecht, Netherlands: Springer US, 1996. – С. 139–180.
4. Гребенникова И. В. Методы оптимизации / И. В. Гребенникова – Екатеринбург: УрФУ, 2017. – С. 6–30.
5. Посыпкин М. А. Мультплатформенный программный комплекс для решения задач оптимизации в распределенной вычислительной среде / М. А. Посыпкин. – Челябинск: ЮУрГУ, 2009.
6. Little J.D.C., Murty K.G., Sweeney D.W., and Karel C. / An algorithm for the traveling salesman problem. Operations Research, 1963. С. 972-989.
7. Леонова М. В. Алгоритм розв’язування задачі про оптимальні призначення методом гілок та меж / М. В. Леонова. – Полтава: Полтавський національний педагогічний університет імені В.Г. Короленка, 2013.
8. Генетический алгоритм — наглядная реализация [Електронний ресурс] // Habr. – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/254759/.

9. Симанков В.С. Генетические алгоритмы и поиск оптимальных решений / В.С. Симанков, В.А. Частикова // Автоматизация и современные технологии, 2003. – № 6. – С. 39–45.
10. Yang Xin-She. Firefly Algorithm, Stochastic Test Functions and Design optimization // International Journal of Bio-Inspired Computation. 2010. V. 2. N 2. P. 78—84.
11. Mehrabiana A.R., Lucase C. A novel numerical optimization algorithm inspired from weed colonization // Ecological informatics. 2006. No 1. P. 355—366.
12. Henryk Josiński, Daniel Kostrzewa, Agnieszka Michalczuk, Adam Świtoński, "The Expanded Invasive Weed Optimization Metaheuristic for Solving Continuous and Discrete Optimization Problems", The Scientific World Journal, vol. 2014, Article ID 831691, 14 pages, 2014.
13. Yang X.-S., Deb S. Cuckoo search via Levy flights // Proc. of the world Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, India. IEEE Publications, USA, pp. 210—214.
14. Oliver C. Ibe. Levy Processes / Oliver C. Ibe // Markov Processes for Stochastic Modeling (Second Edition) / Oliver C. Ibe., 2013. – С. 329–347.
15. Tuba M., Subotic M., Stanarevic N. Modified cuckoo search algorithm for unconstrained optimization problems // Proc. of the 5th European Computing Conference (ECC'I1), Paris, France, April 28—30, 2011. pp. 263—268.
16. Yang X.-S. A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NISCO 2010). Studies in Computational Intelligence. Berlin: Springer, 2010. Vol. 284. P.65-74.
17. Doppler effect [Електронний ресурс] // The Editors of Encyclopaedia Britannica. – 2019. – Режим доступу до ресурсу: https://www.britannica.com/science/Doppler-effect.

18. Mucherino A., Seref O. Monkey Search: A Novel Meta-Heuristic Search for Global Optimization // Data Mining, System Analysis and Optimization in Biomedicine, AIP Conference Proceedings. 2007. P. 162—173.
19. Herbert Robbins. Sutton Monro //A Stochastic Approximation Method, 400 - 407, September, 1951
20. Zhao R., Tang W. Monkey Algorithm for Global Numerical Optimization // Journal of Uncertain Systems. 2008. V. 2. N 3. P. 165—176.
21. Bastos-Filho C.J.A., Lima-Neto F. B., Lins A., Nascimento A., Lima M. Fish school search // Nature-Inspired Algorithms for Optimisation. SCI. Springer. Heidelberg. 2009. Vol. 193. P. 261—277.
22. Cavaicanti-J'unior G. M., Bastos-Fitho C. J. A., Lima-Neto F. B., Castro R. M. C.S. A Hybrid Algorithm Based on Fish School Search and Particle Swarm Optimization for Dynamic Problems // Internauional Conference on Swarm Intelligence 2011 (ICSI). V. 2. P. 543—552.
23. Генетические алгоритмы [Електронний ресурс] – Режим доступу до ресурсу: https://prog-cpp.ru/genetic/.
24. Vijini Mallawaarachchi. Introduction to Genetic Algorithms — Including Example Code [Електронний ресурс] / Vijini Mallawaarachchi. – 2017. – Режим доступу до ресурсу: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.
25. Fish School Search [Електронний ресурс]. – 2007. – Режим доступу до ресурсу: https://fbln.me/fss/.
26. Carmelo J. A. Bastos-Filho. Multi Objective Fish School Search / Carmelo J. A. Bastos-Filho, Augusto C. S. Guimarães. – Brazil: University of Pernambuco, 2017.
27. A tour of the C# language [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
28. PYPL PopularitY of Programming Language [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://pypl.github.io/PYPL.html.

29. TIOBE Index for April 2021 [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.
30. MATT WATSON. What is C# used for? [Електронний ресурс] / MATT WATSON // STACKIFY PRODUCT & COMPANY UPDATES. – 2020. – Режим доступу до ресурсу: https://stackify.com/what-is-c-used-for/.
31. Top 10 Most Important Features Of C# [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/top-10-most-important-features-of-C-Sharp-programming/.
32. Advantages of C# language. [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://blogs.brainsmiths.com/post/2019/12/10/advantages-of-c-language.aspx.
33. C Sharp – Features, Advantages and Disadvantages [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.urbannaturale.com/c-sharp-features-advantages-and-disadvantages/.
34. Java vs C# - 10 Key Differences between Java and C# [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.guru99.com/java-vs-c-sharp-key-difference.html.
35. What is .NET Framework? [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework.
36. What is .NET Framework? Explain Architecture & Components [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/net-framework.html.
37. Desktop Guide (Windows Forms .NET) [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0.

38. Getting started with .NET Charts [Електронний ресурс] – Режим доступу до ресурсу: https://www.i-programmer.info/programming/uiux/2756-getting-started-with-net-charts.html.
39. A flexible charting library for .NET [Електронний ресурс]. – 2007. – Режим доступу до ресурсу: https://www.codeproject.com/Articles/5431/A-flexible-charting-library-for-NET.
40. NET regular expressions [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions.
41. Adam Hughes. Microsoft SQL Server [Електронний ресурс] / Adam Hughes, Craig Stedman. – 2019. – Режим доступу до ресурсу: https://searchdatamanagement.techtarget.com/definition/SQL-Server.
42. Use the MSTest framework in unit tests [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/test/using-microsoft-visualstudio-testtools-unittesting-members-in-unit-tests?view=vs-2019.

ДОДАТОК 1

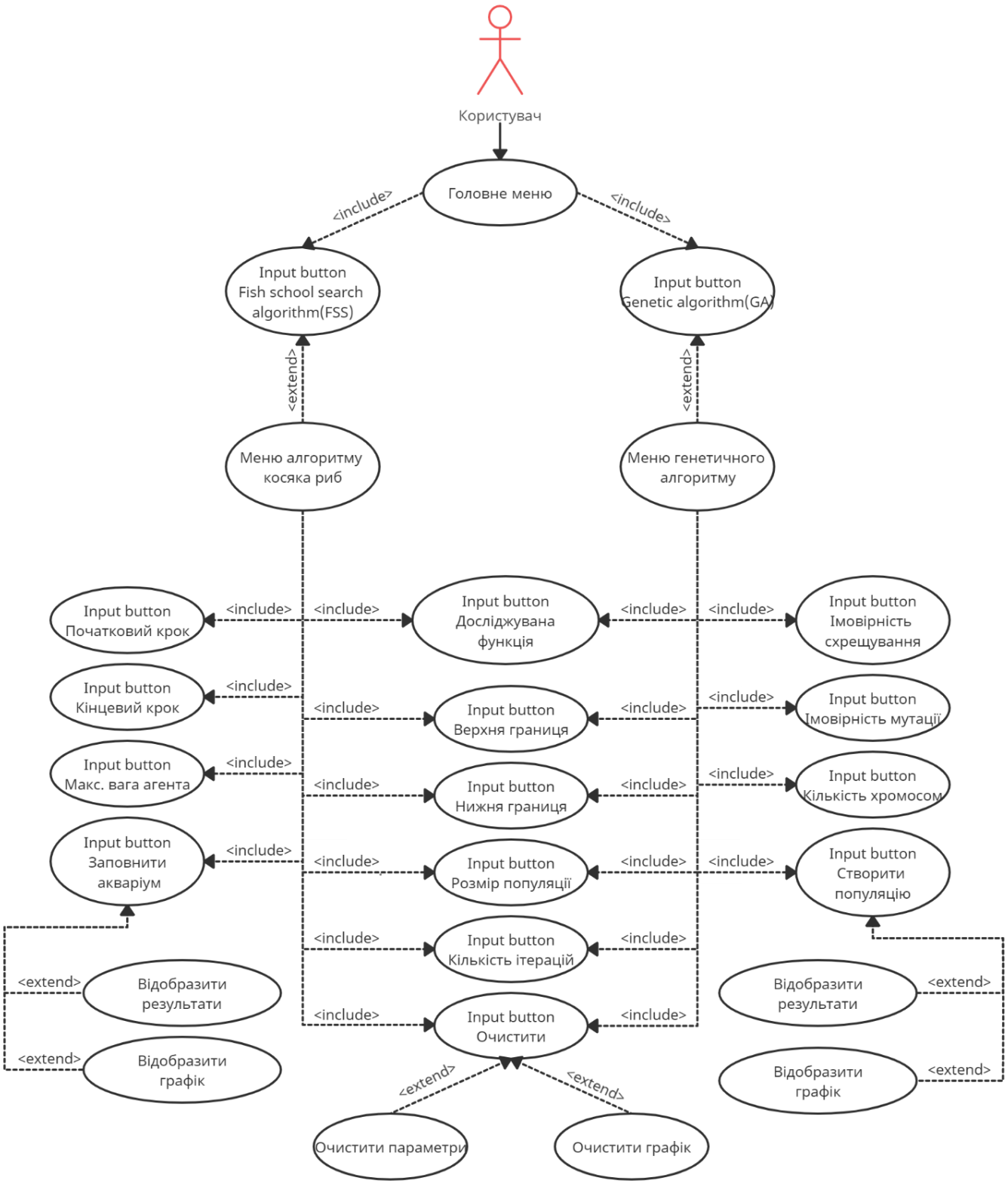
Еволюційні алгоритми глобальної пошукової оптимізації

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2021 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Герега Б. Д.			**Еволюційні алгоритми глобальної пошукової оптимізації** Структурна схема системи	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-73
Затвердив						

ДОДАТОК 2

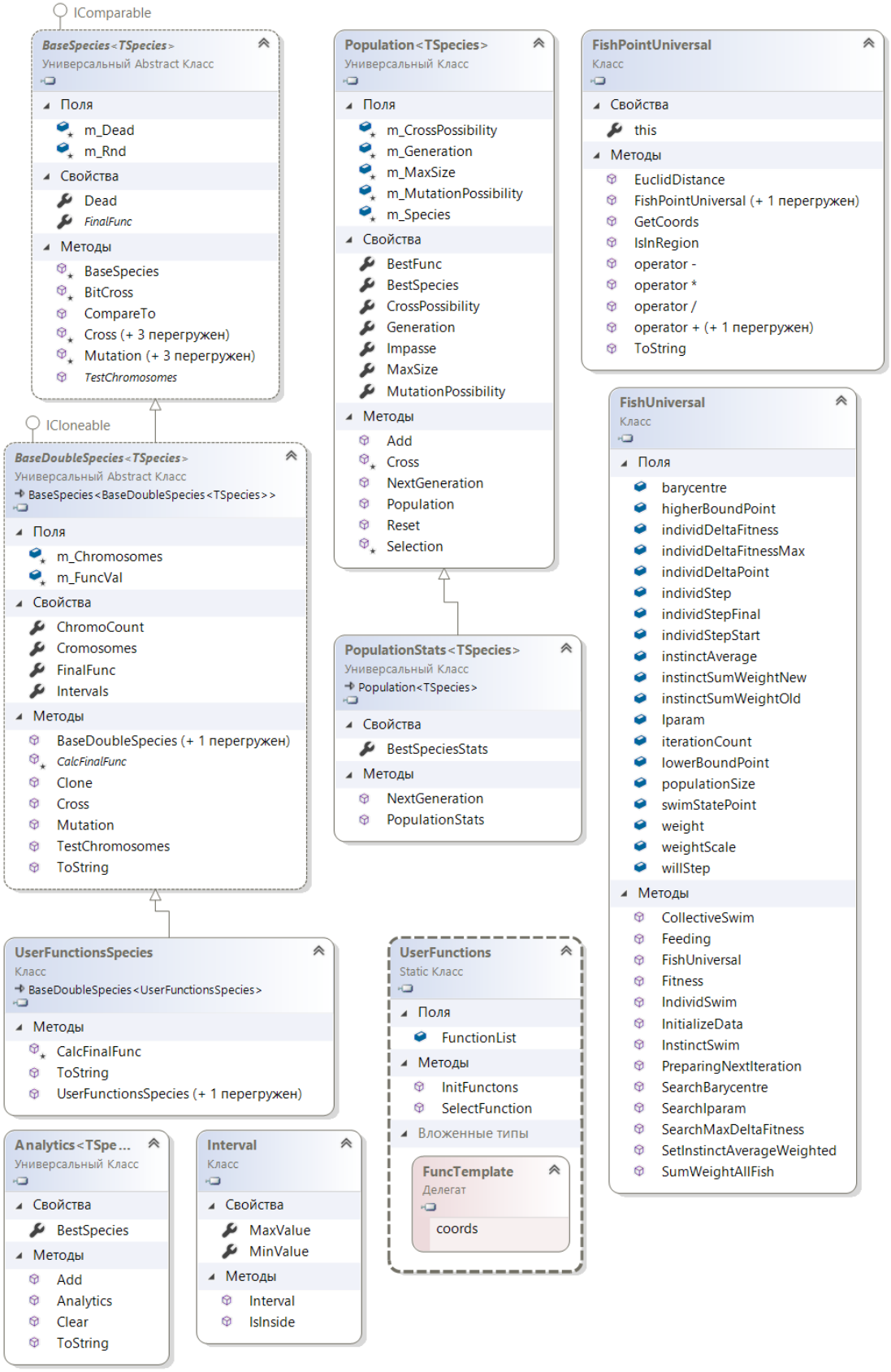
Еволюційні алгоритми глобальної пошукової оптимізації

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2021 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Герега Б. Д.			**Еволюційні алгоритми глобальної пошукової оптимізації** Функціональна схема (діаграма класів)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-73
Н. Контр.		Сімоненко В. П.				
Затвердив						

ДОДАТОК 3

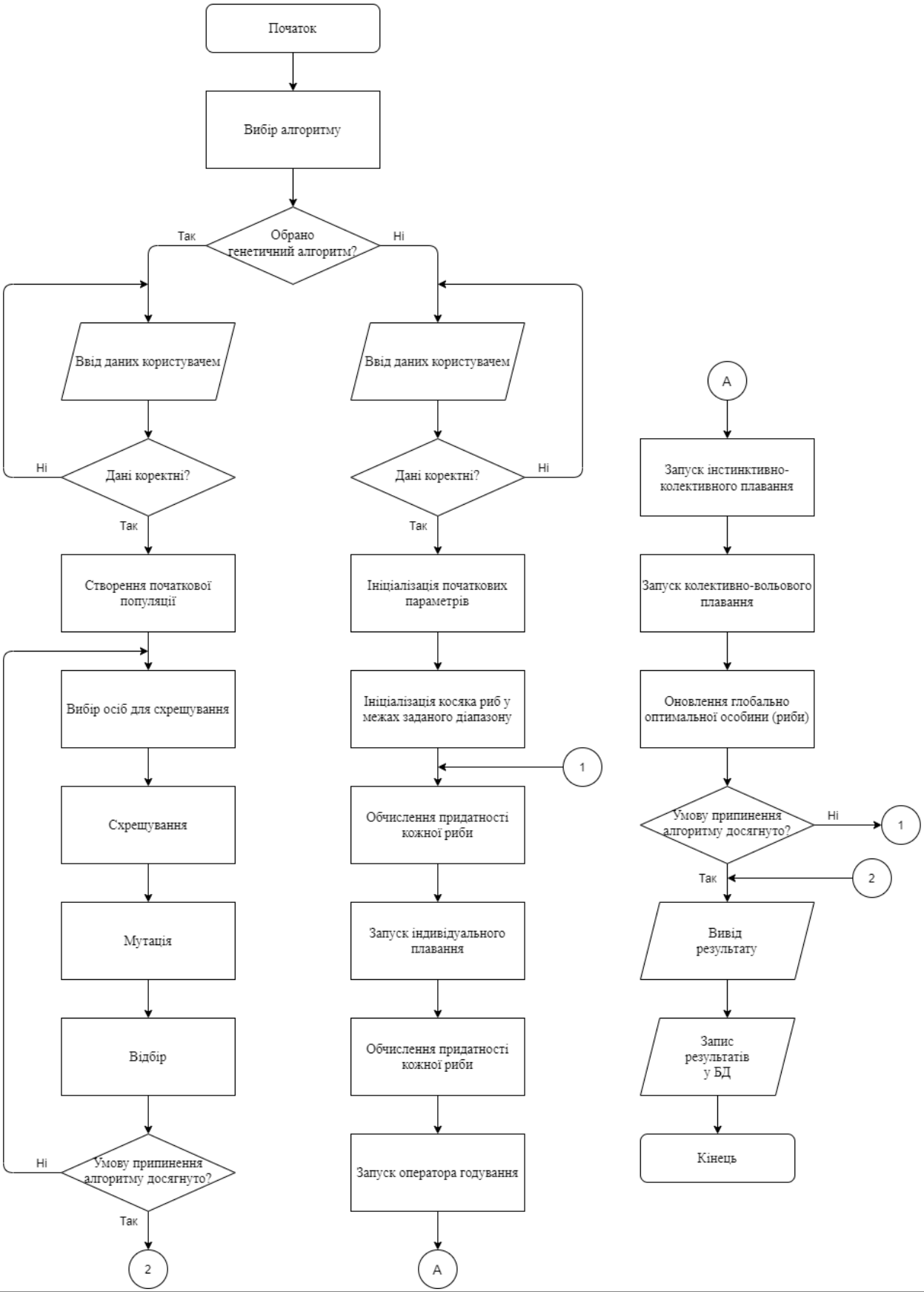
Еволюційні алгоритми глобальної пошукової оптимізації

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2021 р



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Герега Б. Д.			**Еволюційні алгоритми глобальної пошукової оптимізації** Алгоритм дій програмного забезпечення	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-73	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 4

Еволюційні алгоритми глобальної пошукової оптимізації

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 41

Київ 2021 р

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GA
{
	public class Analytics<TSpecies>
		where TSpecies : BaseDoubleSpecies<TSpecies>
	{
		/// <summary>
		/// Здесь храним лучших особей в каждом поколении
		/// </summary>
		List<BaseDoubleSpecies<TSpecies>> m_bestSpecies = new
List<BaseDoubleSpecies<TSpecies>>();

		public List<BaseDoubleSpecies<TSpecies>> BestSpecies
		{
			get { return m_bestSpecies; }
		}

		/// <summary>
		/// Добавить особь в список
		/// </summary>
		/// <param name="species"></param>
		public void Add(BaseDoubleSpecies<TSpecies> species)
		{
			m_bestSpecies.Add((BaseDoubleSpecies<TSpecies>)species.Clone());
		}

		/// <summary>
		/// Очистить список особей
		/// </summary>
		public void Clear()
		{
			m_bestSpecies.Clear();
		}

		/// <summary>
		/// Оформить статистику в виде столбцов
		/// </summary>
		/// <remarks>Столбцы: Хромосома1, Хромосома2, ..., Целевая функция</remarks>
		/// <returns></returns>
		public override string ToString()
		{
			StringBuilder sb = new StringBuilder();

			foreach (BaseDoubleSpecies<TSpecies> species in m_bestSpecies)
			{
				foreach (double chromosome in species.Cromosomes)
				{
					sb.AppendFormat("{0}    ", chromosome);
				}

				sb.AppendFormat("{0}\r\n", species.FinalFunc);
			}

			return sb.ToString();
		}
```

```
}
}
using System;
using System.Collections.Generic;

namespace GA
{
	/// <summary>
	/// Summary description for DoubleSpecies.
	/// </summary>
	public abstract class BaseDoubleSpecies<TSpecies> :
		BaseSpecies<BaseDoubleSpecies<TSpecies>>, ICloneable
		where TSpecies : BaseDoubleSpecies<TSpecies>
	{
		public BaseDoubleSpecies()
		{
			m_Chromosomes = new double[m_Intervals.Length];

			// Заполним массив хромосом случайными числами в заданном интервале
			for (int i = 0; i < m_Intervals.Length; ++i)
			{
				Interval interval = m_Intervals[i];

				m_Chromosomes[i] = m_Rnd.NextDouble() * (
					interval.MaxValue - interval.MinValue) +
interval.MinValue;
			}

			m_FuncVal = CalcFinalFunc();
		}

		public BaseDoubleSpecies(double[] chromosomes)
		{
			m_Chromosomes = (double[])chromosomes.Clone();
			m_FuncVal = CalcFinalFunc();
		}


		public int ChromoCount
		{
			get { return m_Chromosomes.Length; }
		}

		/// <summary>
		/// Значение целевой функции
		/// </summary>
		protected double m_FuncVal = double.MaxValue;

		/// <summary>
		/// Расчет целевой функции.
		/// Возвращаемое значение сохраняется в m_FuncVal
		/// </summary>
		/// <returns></returns>
		abstract protected double CalcFinalFunc();

		/// <summary>
		/// Хромосомы
		/// </summary>
		protected double[] m_Chromosomes;

		public double[] Cromosomes
		{
			get
			{
```

```
                        return m_Chromosomes;
                }
        }

        static private Interval[] m_Intervals;

        /// <summary>
        /// Интервалы изменения хромосом
        /// </summary>
        public static Interval[] Intervals
        {
                get { return m_Intervals; }
                set { m_Intervals = value; }
        }



        public override BaseDoubleSpecies<TSpecies> Cross(BaseDoubleSpecies<TSpecies>
Species)
        {
                if (this == Species)
                {
                        //return new BaseDoubleSpecies<TSpecies> (m_Chromosomes);
                        //Створює екземпляр вказаного типу за допомогою конструктора,
який найкраще відповідає зазначеним параметрам
                        return (TSpecies)Activator.CreateInstance(typeof(TSpecies),
                                new object[] { m_Chromosomes });
                }

                TSpecies Other = (TSpecies)Species;

                //В данном конкретном случае лучше работает скрещивание сразу всех
хромосом

                double[] chromosomes = new double[m_Chromosomes.Length];
                for (int i = 0; i < chromosomes.Length; ++i)
                {
                        chromosomes[i] = Cross(m_Chromosomes[i], Other.Cromosomes[i]);
                }

                return (TSpecies)Activator.CreateInstance(typeof(TSpecies),
                                new object[] { chromosomes });
        }


        override public double FinalFunc
        {
                get
                {
                        return m_FuncVal;
                }
        }

        public override void Mutation()
        {
                Int32 n = m_Rnd.Next(m_Chromosomes.Length);
                Interval interval = m_Intervals[n];

                m_Chromosomes[n] = m_Rnd.NextDouble() *
                        (interval.MaxValue - interval.MinValue) + interval.MinValue;

                m_FuncVal = CalcFinalFunc();
        }
```

```
            public override void TestChromosomes()
            {
                  Boolean res = false;
                  for (int i = 0; i < m_Chromosomes.Length; i++)
                  {
                        double chromosome = m_Chromosomes[i];

                        if (Double.IsNaN(chromosome) ||
                              !m_Intervals[i].IsInside(chromosome))
                        {
                              res = true;
                              break;
                        }
                  }

                  m_Dead = res;
            }

            public override string ToString()
            {
                  return base.ToString();
            }


            #region ICloneable Members

            public object Clone()
            {
                  return Activator.CreateInstance(typeof(TSpecies),
                        new object[] { m_Chromosomes }); ;
            }

            #endregion
      }
}
using System;
//using System.Diagnostics;

namespace GA
{
      /// <summary>
      /// Базовый класс для вида
      /// </summary>
      abstract public class BaseSpecies<TSpecies> : IComparable
            where TSpecies : BaseSpecies<TSpecies>
      {
            /// <summary>
            /// Используется для получения позиции пересечения хромосом и мутации
            /// </summary>
            static protected Random m_Rnd = new Random();

            #region Статические функции для скрещивания хромосом базовых типов
            /// <summary>
            /// Схрестити дві хромосоми типу double
            /// </summary>
            /// <param name="x">1-а хромосома</param>
            /// <param name="y">2-а хромосома</param>
            /// <returns>Нова хромосома</returns>
            static protected double Cross(double x, double y)
            {
                  Int64 ix = BitConverter.DoubleToInt64Bits(x);
                  Int64 iy = BitConverter.DoubleToInt64Bits(y);

                  double res = BitConverter.Int64BitsToDouble(BitCross(ix, iy));
```

```
                if (m_Rnd.Next() % 2 == 0)
                {
                        if (x * res < 0)
                        {
                                res = -res;
                        }
                }
                else
                {
                        if (y * res < 0)
                        {
                                res = -res;
                        }
                }

                return res;
        }

        /// <summary>
        /// Схрестити дві хромосоми типа int
        /// </summary>
        /// <param name="x">1-а хромосома</param>
        /// <param name="y">2-а хромосома</param>
        /// <returns>Нова хромосома</returns>
        static protected Int32 Cross(Int32 x, Int32 y)
        {
                // Число біт, що залишилися зліва від точки перетину хромосом
                Int32 Count = m_Rnd.Next(1, 31);
                Int32 mask = ~0;

                mask = mask << (32 - Count);

                //Debug.Assert(mask != 0 && mask != ~0 ,String.Format("mask = {0}",
mask));

                Int32 res = (x & mask) | (y & ~mask);

                if (m_Rnd.Next() % 2 == 0)
                {
                        if (x * res < 0)
                        {
                                res = -res;
                        }
                }
                else
                {
                        if (y * res < 0)
                        {
                                res = -res;
                        }
                }

                return res;
        }

        /// <summary>
        /// Схрестити побітово без урахування знаку дві хромосоми типу Int64
        /// </summary>
        /// <param name="x">1-а хромосома</param>
        /// <param name="y">2-а хромосома</param>
        /// <returns>Нова хромосома</returns>
        static protected Int64 BitCross(Int64 x, Int64 y)
        {
                // Число біт, що залишилися зліва від точки перетину хромосом
                int Count = m_Rnd.Next(62) + 1;
```

```
                Int64 mask = ~0;

                mask = mask << (64 - Count);

                return (x & mask) | (y & ~mask);
            }

            /// <summary>
            /// Схрестити побітово з урахуванням знака дві хромосоми типу Int64
            /// </summary>
            /// <param name="x">1-а хромосома</param>
            /// <param name="y">2-а хромосома</param>
            /// <returns>Нова хромосома</returns>
            static protected Int64 Cross(Int64 x, Int64 y)
            {
                /*// Число бит, оставшиеся слева от точки пересечения хромосом
                int Count = m_Rnd.Next(62) + 1;
                Int64 mask = ~0;

                mask = mask << (64 - Count);

           return (x & mask) | (y & ~mask);*/

                Int64 res = BitCross(x, y);

                if (m_Rnd.Next() % 2 == 0)
                {
                    if (x * res < 0)
                    {
                        res = -res;
                    }
                }
                else
                {
                    if (y * res < 0)
                    {
                        res = -res;
                    }
                }

                return res;
            }

            /*protected Int64 Cross2 (Int64 x, Int64 y)
            {
                Int64 a = x;
                Int64 b = y;

                int Count1 = 0;
                for (Count1 = 0; a != 0; a /= 10, Count1++);

                int Count2 = 0;
                for (Count2 = 0; b != 0; b /= 10, Count2++);

                Int32 n1 = Math.Max(Count1, Count2) - 1;
                Int32 n2 = m_Rnd.Next(n1 + 1);
                Int64 t1 = (Int64)(a / Math.Pow(10, n2));
                Int64 t2 = (Int64)(Math.Abs(b) - (Int64)(Math.Abs(b) / Math.Pow(10,
n2)) * Math.Pow(10, n2));

                Int64 res = (Int64)(Math.Abs(t1) * Math.Pow(10, n2) + t2);
                if (res * x < 0)
                {
                    res = -res;
                }
```

```
            return res;
        }*/
        #endregion

        #region Методи для мутацій в базових типах
        /// <summary>
        /// Мутація для типу double
        /// </summary>
        /// <param name="val">Значення, яке мутуємо</param>
        /// <returns>Промутоване значення</returns>
        static protected double Mutation(double val)
        {
            UInt64 x = BitConverter.ToUInt64(BitConverter.GetBytes(val), 0);
            UInt64 mask = 1;
            mask <<= m_Rnd.Next(63);
            x ^= mask;

            double res = BitConverter.ToDouble(BitConverter.GetBytes(x), 0);

            return res;
        }

        /// <summary>
        /// Мутація для типу Int32
        /// </summary>
        /// <param name="val">Мутируемое значение</param>
        /// <returns>Промутирующее значение</returns>
        static protected Int32 Mutation(Int32 val)
        {
            Int32 mask = (1 << m_Rnd.Next(31));

            return val ^ mask;
        }

        /// <summary>
        /// Мутація для типу Int64
        /// </summary>
        /// <param name="val">Значення, яке мутуємо</param>
        /// <returns>Промутоване значення</returns>
        static protected Int64 Mutation(Int64 val)
        {
            Int64 mask = 1;
            mask <<= m_Rnd.Next(63);

            return val ^ mask;
        }
        #endregion

        /// <summary>
        /// Мертв ли вид.
        /// </summary>
        /// <remarks>Например, когда хромосовы не попадают в заданные
интервал</remarks>
        protected Boolean m_Dead = false;

        /// <summary>
        /// Мертвый ли вид?
        /// </summary>
        public bool Dead
        {
            get { return m_Dead; }
        }

        /// <summary>
```

```
		/// Проверяет, чтобы все хромосомы попали бы в заданные интервалы.
		/// В противном случае помечает вид как "мертвый".
		/// </summary>
		abstract public void TestChromosomes();

		/// <summary>
		/// Метод для скрещивания себя с другим видом
		/// </summary>
		/// <param name="Species">Другой вид</param>
		/// <returns>Полученный вид</returns>
		abstract public TSpecies Cross(TSpecies Species);

		/// <summary>
		/// Произвести мутацию
		/// </summary>
		abstract public void Mutation();

		/// <summary>
		/// Целевая функция. Должна в случае удачного набора хромосом стремиться к
своему минимуму
		/// </summary>
		/// <returns>Значение целевой функции</returns>
		abstract public double FinalFunc
		{
			get;
		}

		#region IComparable Members

		/// <summary>
		/// Вид считается больше, если он мертв или у него больше целевая функция
		/// </summary>
		/// <param name="obj"></param>
		/// <returns></returns>
		public Int32 CompareTo(object obj)
		{
			TSpecies Other = (TSpecies)obj;

			Int32 res = 0;

			// Если сам мертв, а другой - нет
			if (this.Dead && !Other.Dead)
			{
				res = 1;
			}
			else if (!this.Dead && Other.Dead)
			{
				// Если сам жив, а другой вид мертв
				res = -1;
			}
			else
			{
				// "Там все живы..." (c) :)
				// Все определяет целевая функция

				double ThisFunc = this.FinalFunc;
				double OtherFunc = Other.FinalFunc;
				if (ThisFunc > OtherFunc)
				{
					res = 1;
				}
				else if (ThisFunc < OtherFunc)
				{
					res = -1;
				}
```

```
                }

                return res;
            }   // public Int32 CompareTo(object obj)

            #endregion
        }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace GA
{
        public class Interval
        {
            double m_minVal = double.MinValue;

            /// <summary>
            /// Минимальное значение интервала
            /// </summary>
            public double MinValue
            {
                get { return m_minVal; }
                set { m_minVal = value; }
            }


            double m_maxVal = double.MaxValue;

            /// <summary>
            /// Максимальное значение интервала
            /// </summary>
            public double MaxValue
            {
                get { return m_maxVal; }
                set { m_maxVal = value; }
            }

            public Interval(double minval, double maxval)
            {
                m_minVal = minval;
                m_maxVal = maxval;
            }

            /// <summary>
            /// Попадает ли значение в заданный интервал
            /// </summary>
            /// <param name="val"></param>
            /// <returns></returns>
            public bool IsInside(double val)
            {
                return val >= m_minVal && val <= m_maxVal;
            }
        }
}
using System;
using System.Collections.Generic;

namespace GA
{
        /// <summary>
        /// Бросается, когда пытаемся получить новое поколение, а готовых видов нет
        /// </summary>
```

```
class ZeroSizePopulationException : Exception
{
}

/// <summary>
/// Класс популяции. Все виды размножаются и живут в популяции
/// </summary>
public class Population<TSpecies> where TSpecies : BaseSpecies<TSpecies>
{
	/// <summary>
	/// Номер текущего поколения
	/// </summary>
	protected Int32 m_Generation = 0;

	public Int32 Generation
	{
		get
		{
			return m_Generation;
		}
	}

	/// <summary>
	/// Здесь храним все виды
	/// </summary>
	protected List<TSpecies> m_Species = new List<TSpecies>();

	/// <summary>
	/// Максимальное число видов, которое может содержать популяция
	/// </summary>
	protected Int32 m_MaxSize = 500;

	/// <summary>
	/// Максимальное число видов, которое может содержать популяция
	/// </summary>
	public Int32 MaxSize
	{
		get
		{
			return m_MaxSize;
		}
		set
		{
			if (value < 2)
			{
				throw new ArgumentOutOfRangeException("MaxSize", value,
					"Размер популяции должен быть больше 2");
			}

			m_MaxSize = value;
		}
	}

	/// <summary>
	/// Вероятность скрещивания
	/// </summary>
	protected double m_CrossPossibility = 0.95;

	/// <summary>
	/// Вероятность скрещивания
	/// </summary>
	public double CrossPossibility
	{
		get
		{
```

```
                        return m_CrossPossibility;
                }

                set
                {
                        if (value <= 0 || value > 1.0)
                        {
                                throw new ArgumentOutOfRangeException("CrossPossibility",
value,
                                        "Вероятность скрещивания должна быть положительной
и меньше 1.0");
                        }

                        m_CrossPossibility = value;
                }
        }

        /// <summary>
        /// Вероятность мутации
        /// </summary>
        protected double m_MutationPossibility = 0.1;

        /// <summary>
        /// Вероятность мутации
        /// </summary>
        public double MutationPossibility
        {
                get
                {
                        return m_MutationPossibility;
                }

                set
                {
                        if (value < 0 || value > 1.0)
                        {
                                throw new
ArgumentOutOfRangeException("MutationPossibility", value,
                                        "Вероятность мутации должна быть положительной и
меньше 1.0");
                        }

                        m_MutationPossibility = value;
                }
        }

        public Population()
        {
        }

        Random m_Rnd = new Random();

        /// <summary>
        /// Оновити популяцію (отримати наступне покоління)
        /// </summary>
        virtual public void NextGeneration()
        {
                if (m_Generation == 0 && m_Species.Count == 0)
                {
                        throw new ZeroSizePopulationException();
                }

                // Спочатку схрестимо
                Cross();
```

```
                // Промутуємо і заодно перевіримо всі хромосоми
                foreach (TSpecies species in m_Species)
                {
                        // Якщо треба мутувати з урахуванням ймовірності
                        if (m_Rnd.NextDouble() <= m_MutationPossibility)
                        {
                                species.Mutation();
                        }

                        species.TestChromosomes();
                }

                // Відбераємо найживучіші види
                m_Species.Sort();
                Selection();

                m_Generation++;
        }

        #region Вспомогательные функции для получения нового поколения

        /// <summary>
        /// Отримати нові види схрещуванням
        /// </summary>
        protected void Cross()
        {
                // Розмір до початку поповнення популяції (щоб не схрещувати нові види,
                // які додаються в кінець списку)
                Int32 OldSize = m_Species.Count;

                // Номер пари для схрещувати виду
                Int32 Count = m_Species.Count;

                for (int i = 0; i < Count; ++i)
                {
                        // Якщо треба схрещувати з урахуванням ймовірності
                        if (m_Rnd.NextDouble() <= m_CrossPossibility)
                        {
                                // Додаємо в список виду, отриманий схрещуванням
чергового виду i
                                // виду з випадковим номером

      m_Species.Add(m_Species[i].Cross(m_Species[m_Rnd.Next(OldSize)]));
                        }
                }
        }

        /// <summary>
        /// Получить наилучшее значение целевой функции
        /// </summary>
        public double BestFunc
        {
                get
                {
                        return m_Species[0].FinalFunc;
                }
        }

        /// <summary>
        /// Получить лучший вид
        /// </summary>
        public TSpecies BestSpecies
        {
                get
                {
```

```
                        return m_Species[0];
                }
        }

        /// <summary>
        /// Провести відбір найбільш "живучих" видів
        /// </summary>
        protected void Selection()
        {
                // Скільки видів треба видалити
                Int32 Count = m_Species.Count - m_MaxSize;

                for (Int32 i = 0; i < Count; ++i)
                {
                        m_Species.RemoveAt(m_Species.Count - 1);
                }
        }

        #endregion

        /// <summary>
        /// Возвращает True, если популяция в тупике (все виды одинаковые)
        /// </summary>
        public Boolean Impasse
        {
                get
                {
                        Boolean res = true;
                        TSpecies FirstSpecies = m_Species[0];

                        foreach (TSpecies species in m_Species)
                        {
                                if (species.CompareTo(FirstSpecies) != 0)
                                {
                                        res = false;
                                        break;
                                }
                        }

                        return res;
                }
        }

        /// <summary>
        /// Добавить вид в популяцию
        /// </summary>
        /// <param name="species">Новый вид</param>
        public void Add(TSpecies species)
        {
                m_Species.Add(species);
        }

        public void Reset()
        {
                m_Generation = 0;
                m_Species.Clear();
        }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace GA
```

```
{
	public class PopulationStats<TSpecies> : Population<TSpecies>
		where TSpecies : BaseSpecies<TSpecies>, ICloneable
	{
		/// <summary>
		/// Лучшие решения. Первый индекс - номер популяции, второй - номер запуска.
		/// </summary>
		List<List<TSpecies>> m_bestSpeciesStats = new List<List<TSpecies>>();

		public List<List<TSpecies>> BestSpeciesStats
		{
			get { return m_bestSpeciesStats; }
		}

		public PopulationStats()
			: base()
		{
			// Добавим в 0-й элемент пустой список. 0-е поколение нас интересовать
не будет
			m_bestSpeciesStats.Add(new List<TSpecies>());
		}

		public override void NextGeneration()
		{
			base.NextGeneration();

			if (m_bestSpeciesStats.Count == m_Generation)
			{
				m_bestSpeciesStats.Add(new List<TSpecies>());
			}


	m_bestSpeciesStats[m_Generation].Add((TSpecies)this.BestSpecies.Clone());
		}
	}
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FSS
{
    public class FishPointUniversal
    {
        private double[] vector;
        public double this[int index]
        {
            get
            {
                if (index < vector.Length)
                    return vector[index];
                throw new IndexOutOfRangeException();
            }
            set
            {
                if (index < vector.Length)
                    vector[index] = value;
                else
                    throw new IndexOutOfRangeException();
            }
        }
        public FishPointUniversal(params double[] coords)
```

```
        {
            vector = coords;
        }
        public FishPointUniversal(int dimensionSize)
        {
            vector = new double[dimensionSize];
        }
        public double[] GetCoords()
        {
            return vector;
        }
        public static double EuclidDistance(FishPointUniversal p1, FishPointUniversal p2)
        {
            if (p1.vector.Length != p2.vector.Length)
                throw new IndexOutOfRangeException();
            else
            {
                double sum = 0;
                for (int i = 0; i < p1.vector.Length; i++)
                    sum += Math.Pow((p2[i] - p1[i]), 2);
                return Math.Sqrt(sum);
            }
        }
        public static FishPointUniversal operator -(FishPointUniversal p1,
FishPointUniversal p2)
        {
            if (p1.vector.Length != p2.vector.Length)
                throw new IndexOutOfRangeException();
            else
            {
                double[] to_return = new double[p1.vector.Length];
                for (int i = 0; i < to_return.Length; i++)
                {
                    to_return[i] = p1[i] - p2[i];
                }
                return new FishPointUniversal(to_return);
            }
        }
        public static FishPointUniversal operator +(FishPointUniversal p1,
FishPointUniversal p2)
        {
            if (p1.vector.Length != p2.vector.Length)
                throw new IndexOutOfRangeException();
            else
            {
                double[] to_return = new double[p1.vector.Length];
                for (int i = 0; i < to_return.Length; i++)
                {
                    to_return[i] = p1[i] + p2[i];
                }
                return new FishPointUniversal(to_return);
            }
        }
        public static FishPointUniversal operator +(FishPointUniversal p1, double number)
        {
            double[] to_return = new double[p1.vector.Length];
            for (int i = 0; i < to_return.Length; i++)
            {
                to_return[i] = p1[i] + number;
            }
            return new FishPointUniversal(to_return);
        }
        public static FishPointUniversal operator *(FishPointUniversal p1, double factor)
        {
            double[] to_return = new double[p1.vector.Length];
```

```
            for (int i = 0; i < to_return.Length; i++)
            {
                to_return[i] = p1[i] * factor;
            }
            return new FishPointUniversal(to_return);
        }
        public static FishPointUniversal operator /(FishPointUniversal p1, double divider)
        {
            double[] to_return = new double[p1.vector.Length];
            for (int i = 0; i < to_return.Length; i++)
            {
                to_return[i] = p1[i] / divider;
            }
            return new FishPointUniversal(to_return);
        }
        public bool IsInRegion(FishPointUniversal lowerBound, FishPointUniversal
higherBound)
        {
            if (lowerBound.vector.Length != higherBound.vector.Length ||
lowerBound.vector.Length != this.vector.Length)
                throw new IndexOutOfRangeException();
            else
            {
                for (int i = 0; i < lowerBound.vector.Length; i++)
                    if (this[i] < lowerBound[i] || this[i] > higherBound[i])
                        return false;
                return true;
            }

        }
        public override string ToString()
        {
            string to_return = "(";
            for (int i = 0; i < this.vector.Length; i++)
                to_return += i == this.vector.GetUpperBound(0) ? this[i].ToString() + ")" :
this[i].ToString() + ";";
            return to_return;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace FSS
{
    public class FishUniversal
    {
        #region Общие данные для всего аквариума
        private static Random randGen = new Random();
        public static double Fitness(params double[] coords)
        {
            return funcTemplate(coords);
        }
```

```
        #region Вводимые пользователем данные (с конструктором)

        private static UserFunctions.FuncTemplate funcTemplate;
        /// <summary>
        /// Арность функции (количество аргументов)
        /// </summary>
        private static int dimensionSize;
        /// <summary>
        /// Численность популяции
        /// </summary>
        public static int populationSize;
        /// <summary>
        /// Число итераций (как одно из условий окончания плавания)
        /// </summary>
        public static int iterationCount;
        /// <summary>
        /// Границы аквариума (область исследования)
        /// </summary>
        public static FishPointUniversal lowerBoundPoint, higherBoundPoint;
        //OLD: public static double lowerBoundX, higherBoundX;//Границы аквариума (область
исследования)
        /// <summary>
        /// Ограничение на максимальный вес
        /// </summary>
        public static double weightScale;
        /// <summary>
        /// Границы шагов смещения в индивидуальном плавании
        /// </summary>
        public static double individStepStart, individStepFinal;

        public static void InitializeData(UserFunctions.FuncTemplate _function, int
_dimensionSize, out FishUniversal[] _fishes,
                                          int _popsize, int _iterationCount,
FishPointUniversal _lowerBound, FishPointUniversal _higherBound,
                                          double _individStepStart = 0.5, double
_individStepFinish = 0.1, double _weightScale = 10)
        {
            funcTemplate = _function;
            dimensionSize = _dimensionSize;
            populationSize = _popsize;
            iterationCount = _iterationCount;
            lowerBoundPoint = _lowerBound;
            higherBoundPoint = _higherBound;
```

```
            individStepStart = _individStepStart;
            individStepFinal = _individStepFinish;
            individStep = individStepStart;
            weightScale = _weightScale;
            instinctSumWeightOld = populationSize * weightScale / 2;
            _fishes = new FishUniversal[populationSize];
            for (int fishIndex = 0; fishIndex < _fishes.Length; fishIndex++)
            {
                _fishes[fishIndex] = new FishUniversal();
                // Ініціалізуємо положення риб виходячи з арності функції
                for (int swimState = 0; swimState < 4; swimState++)
                    _fishes[fishIndex].swimStatePoint[swimState]            =            new
FishPointUniversal(dimensionSize);
                for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
                    // Вибираємо початкове положення для 0-й стадії
                    _fishes[fishIndex].swimStatePoint[0][dimensionIndex]                    =
lowerBoundPoint[dimensionIndex] +
                        (higherBoundPoint[dimensionIndex] - lowerBoundPoint[dimensionIndex])
* randGen.NextDouble();
            }
        }
        #endregion
        #region Общие данные для итерации
        /// <summary>
        /// Значение индивидуального шага смещения на данной итерации
        /// </summary>
        public static double individStep;//(затем содержит шаг предыдущей итерации)
        //Изменяемые в каждой итерации
        /// <summary>
        /// Максимальное изменение фитнесс-функции
        /// </summary>
        public static double individDeltaFitnessMax;
        /// <summary>
        /// Среднее взвешенное после инстинктивной стадии
        /// </summary>
        public static FishPointUniversal instinctAverage;
        /// <summary>
        /// Вес рыб после инстинктивного плавания в пред. итерации
        /// </summary>
        public static double instinctSumWeightOld;
        /// <summary>
        /// Вес рыб после инстинктивного плавания в текущей итерации
        /// </summary>
```

```
public static double instinctSumWeightNew;
/// <summary>
/// Шаг волевого смещения (желательна зависимость от индивид. шага смещения)
/// </summary>
public static double willStep;
/// <summary>
/// Центр тяжести рыб, используется в волевом плавании
/// </summary>
public static FishPointUniversal barycentre;
public static FishPointUniversal Iparam;
#endregion
#endregion
#region Индивидуальные данные для каждой рыбки
//Члены объектов
/// <summary>
/// 0 - начало индивидуального плавания
/// 1 -  положение после индвид. плавания,
///2 - после инстинктивного плавания,
/// 3- окончательная позиция итерации (после коллективного плавания)
/// </summary>
public FishPointUniversal[] swimStatePoint = new FishPointUniversal[4];
/// <summary>
/// Смещение после индивидуальной стадии
/// </summary>
public FishPointUniversal individDeltaPoint;
/// <summary>
/// Изменение функции после индивид. плавания
/// </summary>
public double individDeltaFitness;
public double weight = weightScale / 2;//вес рыбки, по умолчанию половина от максимума
//Методы рыбок
/// <summary>
/// Пошук мінімального значення дельта-функції
/// </summary>
/// <param name="fishes"></param>
public static void SearchMaxDeltaFitness(FishUniversal[] fishes)
{
    individDeltaFitnessMax = fishes.Min(a => a.individDeltaFitness);
}
/// <summary>
/// Подсчет среднего взвешенного индивидуальных движений
/// </summary>
/// <param name="fishes"></param>
```

```
public static void SetInstinctAverageWeighted(FishUniversal[] fishes)
{
    FishPointUniversal numerator = new FishPointUniversal(dimensionSize);
    double denumerator = 0;
    foreach (var fish in fishes)
    {
        numerator += fish.individDeltaPoint * fish.individDeltaFitness;
        denumerator += fish.individDeltaFitness;
    }
    if (denumerator == 0) instinctAverage = new FishPointUniversal(dimensionSize);
    else
        instinctAverage = numerator / denumerator;
}
/// <summary>
/// Пошук і установка центру ваги системи
/// </summary>
/// <param name="fishes"></param>
public static void SearchBarycentre(FishUniversal[] fishes)
{
    FishPointUniversal numerator = new FishPointUniversal(dimensionSize);
    double denumerator = 0;
    foreach (var fish in fishes)
    {
        numerator += fish.swimStatePoint[2] * fish.weight;
        denumerator += fish.weight;
    }
    if (denumerator == 0) barycentre = new FishPointUniversal(dimensionSize);
    else
        barycentre = numerator / denumerator;
}
/// <summary>
/// Пошук і установка центру ваги системи
/// </summary>
/// <param name="fishes"></param>
public static void SearchIparam(FishUniversal[] fishes)
{
    FishPointUniversal numerator = new FishPointUniversal(dimensionSize);
    double denumerator = 0;
    foreach (var fish in fishes)
    {
        //double n += individDeltaPoint * individDeltaFitness;
        numerator += fish.individDeltaPoint * fish.individDeltaFitness;
        denumerator += fish.individDeltaFitness;
```

```
            }
            if (denumerator == 0) barycentre = new FishPointUniversal(dimensionSize);
            else
                Iparam = numerator / denumerator;
        }
        /// <summary>
        /// Поиск суммарного веса всех рыб
        /// </summary>
        /// <param name="fishes"></param>
        public static void SumWeightAllFish(FishUniversal[] fishes)
        {
            instinctSumWeightNew = fishes.Sum(fish => fish.weight);
        }
        public static void PreparingNextIteration(FishUniversal[] fishes)
        {
            instinctSumWeightOld = instinctSumWeightNew;
            foreach (var fish in fishes)
            {
                fish.swimStatePoint[0] = fish.swimStatePoint[3];
            }
            //Линейно уменьшаем величину шага индвидуального поиска
            individStep -= (individStepStart - individStepFinal) / iterationCount;
        }
        //Индивидуальные для рыбок методы
        /// <summary>
        /// Кормление рыбки
        /// </summary>
        public void Feeding()
        {
            if (individDeltaFitnessMax != 0)//Избегаем NaN
                this.weight += this.individDeltaFitness / individDeltaFitnessMax;
            if (this.weight > weightScale) this.weight = weightScale;
            if (this.weight < 1) this.weight = 1;
        }
        /// <summary>
        /// Стадія індивідуального плавання
        /// </summary>
        public void IndividSwim()
        {
            // Даємо рибкам шанс зробити переміщення
            for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
                this.swimStatePoint[1][dimensionIndex]                                    =
this.swimStatePoint[0][dimensionIndex] + (-1 + randGen.NextDouble() * 2) * individStep;
```

```
            //Delta X и F(x)
            this.individDeltaPoint = this.swimStatePoint[1] - this.swimStatePoint[0];
            this.individDeltaFitness    =    Fitness(this.swimStatePoint[1].GetCoords())    -
Fitness(this.swimStatePoint[0].GetCoords());
            // Перевіряємо переміщення на ефективність
            if   (!this.swimStatePoint[1].IsInRegion(lowerBoundPoint,   higherBoundPoint)   |
this.individDeltaFitness > 0)
            {
                // Якщо переміщення неефективне або неможливе, то вважаємо, що його не було
                this.swimStatePoint[1] = this.swimStatePoint[0];
                this.individDeltaPoint = new FishPointUniversal(dimensionSize);
                this.individDeltaFitness = 0;
            }
        }
        /// <summary>
        /// Стадія інстинктивного плавання
        /// </summary>
        public void InstinctSwim()
        {
            this.swimStatePoint[2] = this.swimStatePoint[1] + instinctAverage;
        }
        /// <summary>
        /// Стадія колективного плавання
        /// </summary>
        public void CollectiveSwim()
        {
            double   euclidD   =   FishPointUniversal.EuclidDistance(this.swimStatePoint[2],
barycentre);
            if (euclidD == 0)
            {
                this.swimStatePoint[3] = this.swimStatePoint[2];
            }
            else if (instinctSumWeightNew > instinctSumWeightOld)
            {
                this.swimStatePoint[3] = this.swimStatePoint[2] - (this.swimStatePoint[2] -
barycentre) * willStep * randGen.NextDouble() / euclidD;
            }
            else
            {
                this.swimStatePoint[3] = this.swimStatePoint[2] + (this.swimStatePoint[2] -
barycentre) * willStep * randGen.NextDouble() / euclidD;
            }
        }
```

```
        #endregion
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FSS
{
    public static class UserFunctions
    {
        public delegate double FuncTemplate(params double[] coords);
        public static List<Tuple<FuncTemplate, int, string>> FunctionList;
        public static void InitFunctons()
        {
            FunctionList = new List<Tuple<FuncTemplate, int, string>>();
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => -x[0] * (10 - x[0]),
1, "y=-x(10-x)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 0.1 * (x[0] * x[0] +
x[1] * x[1])+10, 2, "y=0,1(x^2+z^2)+10"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 3 * x[0] * x[0] +
x[0] * x[1] + 2 * x[1] * x[1] - x[0] - 4 * x[1], 2, "y=3x^2+xz+2z^2-x-4z"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => x[0] * x[0] - x[0] *
x[1] + x[1] * x[1] + 3 * x[0] - 2 * x[1] + 1, 2, "y=x^2-xz+z^2+3x-2z+1"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 10 * (Math.Sin(0.1 *
x[0]) + Math.Sin(0.1 * x[1]) + Math.Sin(0.1 * x[2])), 3,
"y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 0.1 * (x[0] * x[0] +
x[1] * x[1] + x[2] * x[2] + x[3] * x[3] + x[0] * x[3] + x[1] * x[3] + x[1] * x[2]) + 100, 4,
"y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => x[0] * x[0] + x[1] *
x[1] + x[2] * x[2], 3, "Функція Де Джонга (3 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => x[0] * x[0] + x[1] *
x[1] + x[2] * x[2] + x[3] * x[3], 4, "Функція Де Джонга (4 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => x[0] * x[0] + x[1] *
x[1] + x[2] * x[2] + x[3] * x[3] + x[4] * x[4], 5, "Функція Де Джонга (5 змінних)"));

            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 100 * Math.Pow((x[1]
- Math.Pow(x[0], 2)), 2) + Math.Pow((1 - x[0]), 2) + 100 * Math.Pow((x[2] - Math.Pow(x[1],
2)), 2) + Math.Pow((1 - x[1]), 2), 3, "Функція Розенброка (3 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 100 * Math.Pow((x[1]
- Math.Pow(x[0], 2)), 2) + Math.Pow((1 - x[0]), 2) + 100 * Math.Pow((x[2] - Math.Pow(x[1],
2)), 2) + Math.Pow((1 - x[1]), 2) + 100 * Math.Pow((x[3] - Math.Pow(x[2], 2)), 2) +
Math.Pow((1 - x[2]), 2), 4, "Функція Розенброка (4 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 100 * Math.Pow((x[1]
- Math.Pow(x[0], 2)), 2) + Math.Pow((1 - x[0]), 2) + 100 * Math.Pow((x[2] - Math.Pow(x[1],
2)), 2) + Math.Pow((1 - x[1]), 2) + 100 * Math.Pow((x[3] - Math.Pow(x[2], 2)), 2) +
Math.Pow((1 - x[2]), 2) + 100 * Math.Pow((x[4] - Math.Pow(x[3], 2)), 2) + Math.Pow((1 -
x[3]), 2), 5, "Функція Розенброка (5 змінних)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 10 * 3 +
(Math.Pow(x[0], 2) - 10 * Math.Cos(2 * Math.PI * x[0])) + (Math.Pow(x[1], 2) - 10 *
Math.Cos(2 * Math.PI * x[1])) + (Math.Pow(x[2], 2) - 10 * Math.Cos(2 * Math.PI * x[2])), 3,
"Функція Растрігіна (3 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 10 * 4 +
(Math.Pow(x[0], 2) - 10 * Math.Cos(2 * Math.PI * x[0])) + (Math.Pow(x[1], 2) - 10 *
Math.Cos(2 * Math.PI * x[1])) + (Math.Pow(x[2], 2) - 10 * Math.Cos(2 * Math.PI * x[2])) +
(Math.Pow(x[3], 2) - 10 * Math.Cos(2 * Math.PI * x[3])), 4, "Функція Растрігіна (4
змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => 10 * 5 +
(Math.Pow(x[0], 2) - 10 * Math.Cos(2 * Math.PI * x[0])) + (Math.Pow(x[1], 2) - 10 *
Math.Cos(2 * Math.PI * x[1])) + (Math.Pow(x[2], 2) - 10 * Math.Cos(2 * Math.PI * x[2])) +
```

```
(Math.Pow(x[3], 2) - 10 * Math.Cos(2 * Math.PI * x[3])) + (Math.Pow(x[4], 2) - 10 *
Math.Cos(2 * Math.PI * x[4])), 5, "Функція Растрігіна (5 змінних)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => -x[0] *
Math.Sin(Math.Pow(Math.Abs(x[0]), 1 /2 )) - x[1] * Math.Sin(Math.Pow(Math.Abs(x[1]), 1 / 2))
- x[2] * Math.Sin(Math.Pow(Math.Abs(x[2]), 1 / 2)), 3, "Функція Швефеля (3 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => -x[0] *
Math.Sin(Math.Pow(Math.Abs(x[0]), 1 / 2)) - x[1] * Math.Sin(Math.Pow(Math.Abs(x[1]), 1 / 2))
- x[2] * Math.Sin(Math.Pow(Math.Abs(x[2]), 1 / 2)) - x[3] *
Math.Sin(Math.Pow(Math.Abs(x[3]), 1 / 2)), 4, "Функція Швефеля (4 змінні)"));
            FunctionList.Add(new Tuple<FuncTemplate, int, string>(x => -x[0] *
Math.Sin(Math.Pow(Math.Abs(x[0]), 1 / 2)) - x[1] * Math.Sin(Math.Pow(Math.Abs(x[1]), 1 / 2))
- x[2] * Math.Sin(Math.Pow(Math.Abs(x[2]), 1 / 2)) - x[3] *
Math.Sin(Math.Pow(Math.Abs(x[3]), 1 / 2)) - x[4] * Math.Sin(Math.Pow(Math.Abs(x[4]), 1 /
2)), 5, "Функція Швефеля (5 змінних)"));

        }
        /*
         public static void ShowFunctionList()
         {
             InitFunctons();
             for (int i = 0; i < FunctionList.Count; i++)
                 Console.WriteLine("{0}. {1}", i, FunctionList[i].Item3);
         }
        */
        /// <summary>
        /// Выбор функции по его номеру
        /// </summary>
        /// <param name="number">Номер функции</param>
        /// <returns>Возвращает кортеж, состоящий из функции и ее количества аргументов.
Если такой функции нет, то возвращается null</returns>
        public static Tuple<FuncTemplate, int> SelectFunction(int number)
        {
            if (number >= FunctionList.Count || number < 0) return null;
            return new Tuple<FuncTemplate, int>(FunctionList[number].Item1,
FunctionList[number].Item2);
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using FSS;
using GA;
using DatabaseConnection;
using System.Data.SqlClient;
using System.Text.RegularExpressions;
using System.Threading;
using System.Windows.Forms.DataVisualization.Charting;
using ZedGraph;
```

```
using System.Globalization;

namespace FormsApp
{
    public partial class Form1 : Form
    {
        //GA
        Population<BaseDoubleSpecies<UserFunctionsSpecies>>      m_Population      =      new
Population<BaseDoubleSpecies<UserFunctionsSpecies>>();
        Analytics<UserFunctionsSpecies> m_Analytics =  new Analytics<UserFunctionsSpecies>();

        //FSS
        static UserFunctions.FuncTemplate selectedFunction;
        static int dimensionSize, iterationCount, populationSize;
        static double indivStepBegin, indivStepEnd, maxWeight;
        static FishPointUniversal beginArea, endArea;
        static FishUniversal[] fishes;
        static int iterationNumber;
        static BackgroundWorker bgWorker;
        static Stopwatch timer = new Stopwatch();
        static Stopwatch timer2 = new Stopwatch();
        static double[] nums;


        public Form1()
        {
            InitializeComponent();

            UserFunctions.InitFunctons();
            for (int i = 0; i < UserFunctions.FunctionList.Count; i++)
                comboBox1.Items.Add(i + ") " + UserFunctions.FunctionList[i].Item3);

            for (int i = 0; i < UserFunctions.FunctionList.Count; i++)
                comboBox2.Items.Add(i + ") " + UserFunctions.FunctionList[i].Item3);
        }

        void InputData()
        {
            int selectedNumber;
            Tuple<UserFunctions.FuncTemplate, int> selectedItem;
            //int.TryParse(comboBox1.SelectedItem.ToString().Substring(0,      1),      out
selectedNumber);
            selectedNumber = comboBox1.SelectedIndex;
```

```
            selectedItem = UserFunctions.SelectFunction(selectedNumber);
            selectedFunction = selectedItem.Item1;
            dimensionSize = selectedItem.Item2;

            beginArea = new FishPointUniversal(dimensionSize);
            endArea = new FishPointUniversal(dimensionSize);
            double varForOut = 0;
            //Ввод вверхней границы области поиска
            for (int i = 0; i < dimensionSize; i++)
            {
                var  matches  =  Regex.IsMatch(textBox1.Text,  @"^(-?)(0|([1-9][0-9]*))(\,[0-
9]+)?$", RegexOptions.IgnoreCase);

                if (matches)
                {
                    double.TryParse(textBox1.Text, out varForOut);
                    endArea[i] = varForOut;
                }
                else
                {
                    textBox1.Text = "Error";
                }
            }
            //Ввод нижней границы области поиска
            for (int i = 0; i < dimensionSize; i++)
            {
                var  matches  =  Regex.IsMatch(textBox2.Text,  @"^(-?)(0|([1-9][0-9]*))(\,[0-
9]+)?$", RegexOptions.IgnoreCase);

                if (matches)
                {
                    double.TryParse(textBox2.Text, out varForOut);
                    beginArea[i] = varForOut;
                }
                else
                {
                    textBox2.Text = "Error";
                }
            }

            //ввід к-сті ітерацій
            var matches3 = Regex.IsMatch(textBox3.Text, @"^\d+$", RegexOptions.IgnoreCase);
```

```
            if (matches3)
            {
                iterationCount = Convert.ToInt32(textBox3.Text);
            }
            else
            {
                textBox3.Text = "Error";
            }
            //ввід розміру популяції
            var matches4 = Regex.IsMatch(textBox4.Text, @"^\d+$", RegexOptions.IgnoreCase);

            if (matches4)
            {
                populationSize = Convert.ToInt32(textBox4.Text);
            }
            else
            {
                textBox4.Text = "Error";
            }
            //ввід початкового індивідуального кроку
            var matches5 = Regex.IsMatch(textBox5.Text, @"^(-?)(0|([1-9][0-9]*))(\,[0-
9]+)?$", RegexOptions.IgnoreCase);

            if (matches5)
            {
                indivStepBegin = Convert.ToDouble(textBox5.Text);
            }
            else
            {
                textBox5.Text = "Error";
            }
            //ввід кінцевого індивідуального кроку
            var matches6 = Regex.IsMatch(textBox6.Text, @"^(-?)(0|([1-9][0-9]*))(\,[0-
9]+)?$", RegexOptions.IgnoreCase);

            if (matches6)
            {
                indivStepEnd = Convert.ToDouble(textBox6.Text);
            }
            else
            {
                textBox6.Text = "Error";
            }
```

```
            //ввід максимальної ваги риби
            var matches7 = Regex.IsMatch(textBox7.Text, @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase);

            if (matches7)
            {
                maxWeight = Convert.ToDouble(textBox7.Text);
            }
            else
            {
                textBox7.Text = "Error";
            }
            FishUniversal.InitializeData(selectedFunction, dimensionSize, out fishes,
populationSize, iterationCount, beginArea, endArea, indivStepBegin, indivStepEnd, maxWeight);
        }

        static void NextIteration()
        {
            if (iterationNumber <= FishUniversal.iterationCount)
            {
                foreach (var fish in fishes)
                    fish.IndividSwim();
                FishUniversal.SearchMaxDeltaFitness(fishes);
                foreach (var fish in fishes)
                    fish.Feeding();
                FishUniversal.SetInstinctAverageWeighted(fishes);
                foreach (var fish in fishes)
                    fish.InstinctSwim();
                FishUniversal.SearchBarycentre(fishes);
                FishUniversal.willStep = FishUniversal.individStep;
                FishUniversal.SumWeightAllFish(fishes);
                foreach (var fish in fishes)
                    fish.CollectiveSwim();
                FishUniversal.PreparingNextIteration(fishes);
            }
            nums[iterationNumber - 1] = fishes.Min(fish =>
FishUniversal.Fitness(fish.swimStatePoint[3].GetCoords()));
        }



        private void tabPage1_Click(object sender, EventArgs e)
        {
```

```
        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox4_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox5_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox6_TextChanged(object sender, EventArgs e)
        {
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void button2_Click(object sender, EventArgs e)
        {
            bgWorker = new BackgroundWorker() { WorkerReportsProgress = true };
            textBox1.BackColor = Color.White;
            textBox2.BackColor = Color.White;
            textBox3.BackColor = Color.White;
            textBox4.BackColor = Color.White;
            textBox5.BackColor = Color.White;
            textBox6.BackColor = Color.White;
            textBox7.BackColor = Color.White;
            textBox8.ForeColor = Color.Black;
            if (comboBox1.SelectedIndex == -1 || textBox1.Text == "" || textBox2.Text == ""
|| textBox3.Text == "" ||
                textBox4.Text == "" || textBox5.Text == "" || textBox6.Text == "" ||
textBox7.Text == "")
            {
                textBox8.ForeColor = Color.Red;
```

```
                textBox8.Text = "Перевірте чи введено всі дані!";
            }
            else  if  (!Regex.IsMatch(textBox1.Text,  @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox2.Text,           @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox3.Text, @"^\d+$", RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox4.Text, @"^\d+$", RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox5.Text,           @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox6.Text,           @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox7.Text,               @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
            {
                textBox8.ForeColor = Color.Red;
                textBox8.Text = "Перевірте чи правильно введено всі дані!";
                if   (!Regex.IsMatch(textBox1.Text,   @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
                {
                    textBox1.BackColor = Color.IndianRed;
                }
                if   (!Regex.IsMatch(textBox2.Text,   @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
                {
                    textBox2.BackColor = Color.IndianRed;
                }
                if (!Regex.IsMatch(textBox3.Text, @"^\d+$", RegexOptions.IgnoreCase))
                {
                    textBox3.BackColor = Color.IndianRed;
                }
                if (!Regex.IsMatch(textBox4.Text, @"^\d+$", RegexOptions.IgnoreCase))
                {
                    textBox4.BackColor = Color.IndianRed;
                }
                if   (!Regex.IsMatch(textBox5.Text,   @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
                {
                    textBox5.BackColor = Color.IndianRed;
                }
                if   (!Regex.IsMatch(textBox6.Text,   @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
                {
```

```
                textBox6.BackColor = Color.IndianRed;
            }
            if     (!Regex.IsMatch(textBox7.Text,     @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
            {
                textBox7.BackColor = Color.IndianRed;
            }
        }
        else
        {
            bgWorker.DoWork += (bg_sender, bg_e) =>
            {
                timer.Restart();
                for   (iterationNumber   =   1;   iterationNumber   <=   iterationCount;
iterationNumber++)
                {
                    bgWorker.ReportProgress((int)((float)iterationNumber                /
iterationCount * 100));
                    NextIteration();
                }

            };
            bgWorker.ProgressChanged += (progress_sender, progress_e) =>
            {
                textBox8.Text     =     "Відсоток     виконання     алгоритму:     "     +
progress_e.ProgressPercentage + "%" + Environment.NewLine;
            };
            bgWorker.RunWorkerCompleted += (completed_sender, completed_e) =>
            {
                timer.Stop();
                textBox8.Text += "Нижня границя: " + beginArea + Environment.NewLine +
                                 "Верхня границя: " + endArea + Environment.NewLine +
                                 "Кількість    ітерацій:    "    +    iterationCount    +
Environment.NewLine +
                                 "Розмір    популяції:    "    +    populationSize    +
Environment.NewLine +
                                 "Початковий індивідуальний крок: " + indivStepBegin +
Environment.NewLine +
                                 "Кінцевий  індивідуальний  крок:  "  +  indivStepEnd  +
Environment.NewLine +
                                 "Максимальна    вага    риби:    "    +    maxWeight    +
Environment.NewLine;
```

```
                    float         minFunc         =         (float)fishes.Min(fish         =>
FishUniversal.Fitness(fish.swimStatePoint[3].GetCoords()));
                    textBox8.Text   +=   "Мінімальне   значення   функції:   "   +   minFunc   +
Environment.NewLine;
                    float endTime = (float)timer.Elapsed.Milliseconds/1000;
                    textBox8.Text   +=   "Час   виконання   алгоритму:   "   +   endTime   +
Environment.NewLine;

                    SqlConnection connection = DBUtils.GetDBConnection();
                    connection.Open();
                    try
                    {
                        // Команда Insert.
                        string sql = "Insert into [dbo].[FSS_parameters] ([investigatedFunc],
[upperLimit], [lowerLimit], [numberOfIterations], " +
                                     "[populationSize],     [initialStep],     [finalStep],
[maximumWeight], [extremumOfTheFunction], [algorithmExecutionTime]) " +
                                     "values (@_investigatedFunc, @_upperLimit, @_lowerLimit,
@_numberOfIterations, @_populationSize, @_initialStep, " +
                                     "@_finalStep, @_maximumWeight, @_extremumOfTheFunction,
@_algorithmExecutionTime) ";

                        SqlCommand cmd = connection.CreateCommand();
                        cmd.CommandText = sql;

                        // Добавити параметр @investigatedFunc
                        cmd.Parameters.Add("@_investigatedFunc",  SqlDbType.VarChar).Value  =
comboBox1.Text;
                        // Добавити параметр @upperLimit
                        cmd.Parameters.Add("@_upperLimit",      SqlDbType.Float).Value      =
float.Parse(textBox1.Text, CultureInfo.InvariantCulture.NumberFormat);
                        // Добавити параметр @lowerLimit
                        cmd.Parameters.Add("@_lowerLimit",      SqlDbType.Float).Value      =
float.Parse(textBox2.Text, CultureInfo.InvariantCulture.NumberFormat);
                        // Добавити параметр @numberOfIterations
                        cmd.Parameters.Add("@_numberOfIterations",   SqlDbType.Int).Value   =
int.Parse(textBox3.Text, CultureInfo.InvariantCulture.NumberFormat);
                        // Добавити параметр @populationSize
                        cmd.Parameters.Add("@_populationSize",     SqlDbType.Int).Value     =
int.Parse(textBox4.Text, CultureInfo.InvariantCulture.NumberFormat);
                        // Добавити параметр @initialStep
                        cmd.Parameters.Add("@_initialStep",     SqlDbType.Float).Value     =
indivStepBegin;
```

```
                        // Добавити параметр @finalStep
                        cmd.Parameters.Add("@_finalStep",      SqlDbType.Float).Value      =
indivStepEnd;
                        // Добавити параметр @maximumWeight
                        cmd.Parameters.Add("@_maximumWeight",    SqlDbType.Float).Value    =
maxWeight;
                        // Добавити параметр @extremumOfTheFunction
                        cmd.Parameters.Add("@_extremumOfTheFunction", SqlDbType.Float).Value
= minFunc;
                        // Добавити параметр @algorithmExecutionTime
                        cmd.Parameters.Add("@_algorithmExecutionTime",
SqlDbType.Float).Value = endTime;
                        // Выполнить Command (Используется для delete, insert, update).
                        int rowCount = cmd.ExecuteNonQuery();

                        //Console.WriteLine("Row Count affected = " + rowCount);
                    }
                    catch (Exception exep)
                    {
                        Console.WriteLine("Error: " + exep);
                        Console.WriteLine(exep.StackTrace);
                    }
                    finally
                    {
                        connection.Close();
                        connection.Dispose();
                        connection = null;
                    }


                    System.Windows.Forms.DataVisualization.Charting.Axis     ax     =     new
System.Windows.Forms.DataVisualization.Charting.Axis();
                    ax.Title = "Номер ітерації";
                    chart1.ChartAreas[0].AxisX = ax;
                    System.Windows.Forms.DataVisualization.Charting.Axis     ay     =     new
System.Windows.Forms.DataVisualization.Charting.Axis();
                    ay.Title = "Середнє значення";
                    chart1.Titles.Add("Динаміка мінімального значення");
                    chart1.ChartAreas[0].AxisY = ay;
                    /*
                    textBox8.Text += "Динаміка мінімального значення:" + Environment.NewLine;
                    for (int i = 1; i <= iterationCount; i++)
                    {
```

```
                textBox8.Text += i + " ітерація: " + nums[i - 1] + Environment.NewLine;
            }
            */
            for (int i = 1; i <= iterationCount; i++)
            {
                chart1.Series["AverageValue"].Points.AddXY(i, nums[i - 1]);
            }

        };

        textBox8.Clear();
        chart1.Titles.Clear();
        chart1.Series[0].Points.Clear();
        InputData();
        nums = new double[iterationCount];
        bgWorker.RunWorkerAsync();
    }

}

private void button1_Click(object sender, EventArgs e)
{
    textBox1.Clear();
    textBox2.Clear();
    textBox3.Clear();
    textBox4.Clear();
    textBox5.Clear();
    textBox6.Clear();
    textBox7.Clear();
    textBox8.Clear();
    comboBox1.SelectedIndex = -1;
    textBox1.BackColor = Color.White;
    textBox2.BackColor = Color.White;
    textBox3.BackColor = Color.White;
    textBox4.BackColor = Color.White;
    textBox5.BackColor = Color.White;
    textBox6.BackColor = Color.White;
    textBox7.BackColor = Color.White;
    chart1.Series[0].Points.Clear();
}

private void button3_Click(object sender, EventArgs e)
{
```

```
            textBox9.Clear();
            textBox10.BackColor = Color.White;
            textBox11.BackColor = Color.White;
            textBox12.BackColor = Color.White;
            textBox13.BackColor = Color.White;
            textBox14.BackColor = Color.White;
            textBox15.BackColor = Color.White;
            textBox16.BackColor = Color.White;
            textBox9.ForeColor = Color.Black;
            if (comboBox2.SelectedIndex == -1 || textBox10.Text == "" || textBox11.Text == ""
|| textBox12.Text == "" ||
                textBox13.Text == "" || textBox14.Text == "" || textBox15.Text == "" ||
textBox16.Text == "")
            {
                textBox9.ForeColor = Color.Red;
                textBox9.Text = "Перевірте чи введено всі дані!";
            }
            else if (!Regex.IsMatch(textBox16.Text, @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox15.Text,          @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox14.Text, @"^\d+$", RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox13.Text, @"^\d+$", RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox12.Text,              @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox11.Text,              @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase) ||
              !Regex.IsMatch(textBox10.Text, @"^\d+$", RegexOptions.IgnoreCase))
            {
                textBox9.ForeColor = Color.Red;
                textBox9.Text = "Перевірте чи правильно введено всі дані!";
                if  (!Regex.IsMatch(textBox16.Text,  @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
                {
                    textBox16.BackColor = Color.IndianRed;
                }
                if  (!Regex.IsMatch(textBox15.Text,  @"^(-?)(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
                {
                    textBox15.BackColor = Color.IndianRed;
                }
                if (!Regex.IsMatch(textBox14.Text, @"^\d+$", RegexOptions.IgnoreCase))
                {
```

```
                textBox14.BackColor = Color.IndianRed;
            }
            if (!Regex.IsMatch(textBox13.Text, @"^\d+$", RegexOptions.IgnoreCase))
            {
                textBox13.BackColor = Color.IndianRed;
            }
            if    (!Regex.IsMatch(textBox12.Text,    @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
            {
                textBox12.BackColor = Color.IndianRed;
            }
            if    (!Regex.IsMatch(textBox11.Text,    @"^(0|([1-9][0-9]*))(\,[0-9]+)?$",
RegexOptions.IgnoreCase))
            {
                textBox11.BackColor = Color.IndianRed;
            }
            if (!Regex.IsMatch(textBox10.Text, @"^\d+$", RegexOptions.IgnoreCase))
            {
                textBox10.BackColor = Color.IndianRed;
            }
        }
        else
        {
            timer2.Restart();
            int count = Convert.ToInt32(textBox10.Text);

            // Зададим интервалы изменения хромосом
            Interval[] intervals = new Interval[count];

            for (int i = 0; i < count; i++)
            {
                intervals[i]    =    new    Interval(Convert.ToDouble(textBox15.Text),
Convert.ToDouble(textBox16.Text));
            }

            m_Analytics.Clear();

            // Зададим параметры алгоритма
            m_Population.Reset();
            m_Population.MaxSize = Convert.ToInt32(textBox13.Text);
            m_Population.MutationPossibility = Convert.ToDouble(textBox11.Text) / 100.0;
            m_Population.CrossPossibility = Convert.ToDouble(textBox12.Text) / 100.0;
```

```
                // Установим свойства видов
                UserFunctionsSpecies.Intervals = intervals;

                // Добавим виды
                for (int i = 0; i < m_Population.MaxSize; ++i)
                {
                    m_Population.Add(new UserFunctionsSpecies());
                }

                ShowData();

                for (int i = 0; i < Convert.ToInt32(textBox14.Text); ++i)
                {
                    m_Population.NextGeneration();

                    m_Analytics.Add(m_Population.BestSpecies);

                    ShowData();
                    this.Update();
                }



                timer2.Stop();
                float endTime = (float)timer2.Elapsed.Milliseconds / 1000;
                textBox9.Text += Environment.NewLine + "Час виконання алгоритму: " + endTime
+ Environment.NewLine;

                SqlConnection connection = DBUtils.GetDBConnection();
                connection.Open();
                try
                {
                    // Команда Insert.
                    string sql = "Insert into [dbo].[GA_parameters] ([investigatedFunc],
[upperLimit], [lowerLimit], [numberOfIterations], " +
                                 "[populationSize],                [probabilityOfCrossing],
[probabilityOfMutation],           [numberOfChromosomes],           [extremumOfTheFunction],
[algorithmExecutionTime]) " +
                                 "values (@_investigatedFunc, @_upperLimit, @_lowerLimit,
@_numberOfIterations, @_populationSize, @_probabilityOfCrossing, " +
                                 "@_probabilityOfMutation,            @_numberOfChromosomes,
@_extremumOfTheFunction, @_algorithmExecutionTime) ";

                    SqlCommand cmd = connection.CreateCommand();
```

```
                    cmd.CommandText = sql;

                    // Добавити параметр @investigatedFunc
                    cmd.Parameters.Add("@_investigatedFunc",    SqlDbType.VarChar).Value    =
comboBox2.Text;
                    // Добавити параметр @upperLimit
                    cmd.Parameters.Add("@_upperLimit",        SqlDbType.Float).Value        =
float.Parse(textBox16.Text, CultureInfo.InvariantCulture.NumberFormat);
                    // Добавити параметр @lowerLimit
                    cmd.Parameters.Add("@_lowerLimit",        SqlDbType.Float).Value        =
float.Parse(textBox15.Text, CultureInfo.InvariantCulture.NumberFormat);
                    // Добавити параметр @numberOfIterations
                    cmd.Parameters.Add("@_numberOfIterations",     SqlDbType.Int).Value     =
int.Parse(textBox14.Text, CultureInfo.InvariantCulture.NumberFormat);
                    // Добавити параметр @populationSize
                    cmd.Parameters.Add("@_populationSize",       SqlDbType.Int).Value       =
int.Parse(textBox13.Text, CultureInfo.InvariantCulture.NumberFormat);
                    // Добавити параметр @probabilityOfCrossing
                    cmd.Parameters.Add("@_probabilityOfCrossing",  SqlDbType.Float).Value  =
float.Parse(textBox12.Text, CultureInfo.InvariantCulture.NumberFormat) / 100.0;
                    // Добавити параметр @probabilityOfMutation
                    cmd.Parameters.Add("@_probabilityOfMutation",  SqlDbType.Float).Value  =
float.Parse(textBox11.Text, CultureInfo.InvariantCulture.NumberFormat) / 100.0;
                    // Добавити параметр @maximumWeight
                    cmd.Parameters.Add("@_numberOfChromosomes",    SqlDbType.Int).Value    =
int.Parse(textBox10.Text, CultureInfo.InvariantCulture.NumberFormat);
                    // Добавити параметр @extremumOfTheFunction
                    cmd.Parameters.Add("@_extremumOfTheFunction",  SqlDbType.Float).Value  =
(float)m_Analytics.BestSpecies[m_Analytics.BestSpecies.Count-1].FinalFunc;
                    // Добавити параметр @algorithmExecutionTime
                    cmd.Parameters.Add("@_algorithmExecutionTime",  SqlDbType.Float).Value  =
endTime;
                    // Выполнить Command (Используется для delete, insert, update).
                    int rowCount = cmd.ExecuteNonQuery();

                    //Console.WriteLine("Row Count affected = " + rowCount);
                }
                catch (Exception exep)
                {
                    Console.WriteLine("Error: " + exep);
                    Console.WriteLine(exep.StackTrace);
                }
                finally
```

```
        {
            connection.Close();
            connection.Dispose();
            connection = null;
        }


        GraphPane pane = zedGraphControl1.GraphPane;

        pane.Title.Text = "Динаміка мінімального значення";
        pane.XAxis.Title.Text = "Покоління";
        pane.YAxis.Title.Text = "F(x)";
        pane.Chart.Fill.Color = SystemColors.ControlLight;
        pane.Fill.Color = SystemColors.ControlLight;

        pane.CurveList.Clear();

        PointPairList list = new PointPairList();

        for (int i = 0; i < m_Analytics.BestSpecies.Count; i++)
        {
            list.Add(i, m_Analytics.BestSpecies[i].FinalFunc);
        }

        LineItem myCurve = pane.AddCurve("", list, Color.Blue, SymbolType.Square);
        myCurve.Line.IsVisible = false;
        myCurve.Symbol.Fill.Color = Color.Blue;
        myCurve.Symbol.Fill.Type = FillType.Solid;
        myCurve.Symbol.Size = 5;

        zedGraphControl1.AxisChange();

        zedGraphControl1.Invalidate();
    }


}

private void ShowData()
{
    StringBuilder builder = new StringBuilder();
    builder.AppendLine(m_Population.BestSpecies.ToString());
    builder.AppendFormat("Generation = {0}", m_Population.Generation);
```

```
    textBox9.Text = builder.ToString();
}

private void button4_Click(object sender, EventArgs e)
{
    textBox10.Clear();
    textBox11.Clear();
    textBox12.Clear();
    textBox13.Clear();
    textBox14.Clear();
    textBox15.Clear();
    textBox16.Clear();
    textBox9.Clear();
    comboBox2.SelectedIndex = -1;
    textBox10.BackColor = Color.White;
    textBox11.BackColor = Color.White;
    textBox12.BackColor = Color.White;
    textBox13.BackColor = Color.White;
    textBox14.BackColor = Color.White;
    textBox15.BackColor = Color.White;
    textBox16.BackColor = Color.White;
    zedGraphControl1.GraphPane.CurveList.Clear();
    zedGraphControl1.GraphPane.GraphObjList.Clear();
    zedGraphControl1.Refresh();
}

private void Form1_Load(object sender, EventArgs e)
{

}

private void tabPage2_Click(object sender, EventArgs e)
{

}

private void label1_Click(object sender, EventArgs e)
{

}

private void label6_Click(object sender, EventArgs e)
```

```
        {

        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            comboBox1.SelectedIndexChanged += comboBox1_SelectedIndexChanged;
        }

        private void textBox7_TextChanged(object sender, EventArgs e)
        {

        }

    }
}
```