Довідка
про відсутність плагіату у випускній кваліфікаційній роботі

Міністерство освіти і науки України
Український державний університет науки та технологій

Кафедра «Комп'ютерні інформаційні технології»

ДОВІДКА

За результатами перевірки випускної кваліфікаційної роботи здобувача вищої освіти
Галушки Олександра Валентиновича
(прізвище, ім'я, по батькові)
на тему: Засоби підвищення часової ефективності генетичних алгоритмів

в роботі не виявлено порушень академічної доброчесності.

Керівник ВКР _____ Шинкаренко В.І.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет науки і технологій

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри

_____ /Вадим ГОРЯЧКІН/

«17» 12 2021 р.

ДИПЛОМНА РОБОТА

на здобуття освітнього ступеня «магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Засоби підвищення часової ефективності генетичних алгоритмів**

Theme **Methods for enhacing the perfomance of genetic algorithms**

Керівник дипломної роботи проф. _____ Віктор ШИНКАРЕНКО

Нормоконтролер доц. _____ Олена КУРОП'ЯТНИК

Студент групи ПЗ2021 _____ Олександр ГАЛУШКА

Student Oleksandr HALUSHKA

Дніпро – 2021

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет Комп'ютерних технологій і систем кафедра Комп'ютерні інформаційні технології
Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ В. І. Шинкаренко
(підпис)
«17» 12 2021 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС магістр
(освітній ступінь)
студента групи ПЗ2021 Галушки Олександра Валентиновича
(номер групи) (ПІБ)

1 Тема дипломної роботи: Засоби підвищення часової ефективності генетичних алгоритмів
затверджена наказом по університету від «18» листопада 2020 р. № 690 ст.

2 Термін подання студентом закінченої роботи «3» грудня 2021р.

3 Вихідні дані до дипломної роботи _____
_____
_____

4 Зміст пояснювальної записки (перелік питань до розробки) проведення дослідження часової ефективності генетичних алгоритмів відновлення конструктивної моделі заданого часового ряду з наявною фрактальною самоподібністю.

5 Перелік демонстраційного матеріалу презентація на тему використання мультиагентного підходу в контексті генетичних алгоритмів для методу конструктивно-продукційного моделювання фрактальних часових рядів, результати експериментів, висновки; відео демонстрації роботи розробленої мультиагентної системи для проведення досліджень.

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Техніко-економічні розрахунки	доц. Гненний М.В.	18.10.21	24.10.21
Охорона праці	доц. Саблін О.І.	18.10.21	24.10.21

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Назва розділів дипломної роботи	Термін виконання розділів роботи	Примітка
1	Вступ		
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами		
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження		
4	Постановка задачі, технічне завдання	11.10.21 – 17.10.21	30%
5	Техніко-економічні показники		
6	Розробка інструментальних засобів дослідження		
7	Виконання досліджень	08.11.21 – 14.11.21	60%
8	Оформлення тез доповідей		
9	Оформлення статті у фаховий журнал		
10	Оформлення пояснювальної записки		
11	Розробка демонстраційних матеріалів	29.11.21 – 05.12.21	100%

Дата видачі завдання «18» листопада 2020 р.

Керівник дипломної роботи _____ В.І. Шинкаренко
(підпис) (ПІБ)

Завдання прийняв до виконання _____ О.В. Галушка
(підпис) (ПІБ)

РЕФЕРАТ

Об’єктом дослідження є процеси технічні, соціальні, біологічні та інші процеси які моделюються часовими рядами.

Предметом дослідження є методи підвищення часової ефективності генетичних алгоритмів відновлення конструктивної моделі заданого часового ряду з наявною фрактальною самоподібністю.

Методи дослідження: випробування підходу з різними часовими рядами з метою підвищення часової ефективності методу моделювання.

Метою дослідження є покращення ефективності процесу обчислень в контексті відновлення фрактальної моделі часового ряду та вдосконалення процесу відновлення фрактальних складових.

Результатами та їх новизною виступає адаптація мультиагентного підходу в контексті конструктивно-продукційного підходу для моделювання часових рядів, формування теоретичної бази для опису процесу роботи, створення мультиагентної системи для проведення випробувань та її вдосконалення.

Пояснювальна записка складається зі вступу, п’яти розділів, висновків, бібліографічного списку та чотирьох додатків.

Вступ – описує суть, актуальність дослідження, об’єкт та предмет дослідження, мету та завдання досліджень (3 сторінки).

Перший розділ – описує проблематику, огляд аналогічних програмних додатків та визначення інструментів для розробки (13 сторінок).

Другий розділ – описує обґрунтування напрямку досліджень (14 сторінок).

Третій розділ – описує процес проектування і розробки ПЗ (13 сторінок).

Четвертий розділ – дослідження процесу моделювання часоаих рядів з використанням мультиагентного підходу та вдосконалення програмної системи (34 сторінки).

П’ятий розділ – розкриті питання охорони та безпеки праці в надзвичайних ситуаціях (9 сторінок).

Додатки – технічне завдання, робочий проект, тези, проект наукової статті.

ЗМІСТ

ВСТУП

На даний момент придумано чимало методів, моделей, представлень за допомогою яких можна описати стан та поведінку або зміни цих характеристик в межі деякого інтервалу часу. За допомогою цих методів можна проводити моделювання складових системи як у точний момент часу так і інтервально у потрібному для подальшого аналізу та роботи форматі. Результуючий набір даних може бути використано для визначення необхідних показників в роботі системи задля аналізу поведінки системи та прогнозу подальших змін стану та поведінки.

Одна з моделей за допомогою якої можна описати стан та поведінку деякої системи є часовий ряд. Дана модель формується на основі замірів показників системи в через деякі проміжки часу на заданому інтервалі. У відповідності до значень ряду виставляється конкретний момент часу в який були отримано це значення.

Приклади моделі часового ряду:

– електрокардіограма серцебиття;
– діаграма заміру потужності двигуна на динамометричному стенді;
– діаграма динаміки зміни ціни активу на фондовій біржі.

Для візуального представлення як правило використовується лінійна діаграма. Модель часового ряду може використовуватися як для ведення статистики та аналітичної роботи так і для прогнозу подальших змін деякого показника або його складових.

Часовий ряд як аналітична модель може бути сконструйований за допомогою деякої моделі яка будується на основі його самого. Якщо якість формування такої моделі доволі висока, тоді це дозволяє якнайкраще відтворити вхідний ряд але вже з продовженням, що дає змогу якісно проводити прогноз подальших значень ряду.

До таких моделей можна віднести фрактальну модель часового ряду яка формується на основі фрактальних властивостей деяких часових рядів і несе в собі принцип фрактальної самоподібності [1], який закладений в основу фракталів.

Актуальність роботи. Моделювання, аналіз та прогнозування характеристик та показників різних процесів залишається актуальною задачею вже дуже довгий час. Використання моделювання для статистичних даних дозволяє оптимізувати велику кількість процесів в різних сферах діяльності людини. Аналіз часових рядів проводиться для виявлення тенденцій та певних закономірностей які можна зв'язати з зовнішніми обставинами.

Опираючись на модель деякого процесу можливе проведення прогнозування майбутніх значень характеристик цього процесу. Прогнозування дозволяє ефективно мінімізувати ризики та допомагає приймати рішення які націлені на оптимальність проведення того чи іншого процесу виходячи з результатів аналізу.

На даний момент було розроблено велику кількість методів для моделювання та прогнозування часових рядів. При цьому, слідує відмітити, що кожний конкретний метод, як правило, розроблявся для моделювання часових рядів певної природі. З цього можна зробити висновок, що кожен з розроблених методів не є універсальним і не може використовуватись для часових рядів будь-якої природи.

Метод конструктивно-продукційного моделювання можна класифікувати як універсальний метод моделювання процесів, так як в основі даного методу закладено використання фрактальної самоподібності, що дозволяє використовувати його для моделювання часових рядів різної природи.

На даний момент дослідження моделювання процесів проводяться в наступних галузях:

- моделювання в інформаційних та телекомунікаційних технологіях [2];
- стохастичне моделювання і прикладне дослідження технологій [3];
- моделювання складних процесів та систем [4];
- моделювання в інформаційних технологіях та нанотехнологіях [5].

Об'єкт дослідження. Об'єктом дослідження є процеси технічні, соціальні, біологічні та інші процеси які моделюються часовими рядами.

Предмет дослідження. Предметом дослідження є методи підвищення часової ефективності генетичних алгоритмів відновлення конструктивної моделі заданого часового ряду з наявною фрактальною самоподібністю.

Мета і завдання дослідження. Метою дослідження в даній роботі є формалізація методів покращення формування фрактальної моделі методами конструктивно-продукційного моделювання та підвищення часової ефективності роботи системи. Зазначена мета передбачає рішення наступних завдань:

- визначення структури мультиагентної системи для масштабування обчислювальної потужності системи;
- адаптація розподілених еволюційних обчислень на базі генетичного алгоритму в контексті методу формування фрактальної моделі часового ряду;
- створення програмної системи для автоматизації конструювання фрактальної моделі та порівняння отриманих результатів відносно базової версії додатку.

Наукова новизна. Використання методів розподілених еволюційних обчислень для конструктивно-продукційного моделювання фрактальної моделі часових рядів.

Практичне значення. За допомогою формування фрактальної моделі складного часового ряду, його дослідження та прогнозування відповідного процесу з’являються додаткові можливості аналізу та управління системами та процесами, які представлені часовими рядами.

Апробація результатів досліджень. Процес та результати роботи над дослідженнями доповідалися на засіданні кафедри КІТ 22.02.21, тези подано на конференцію: «81 Всеукраїнська науково-технічна конференція «Наука і сталий розвиток транспорту». Інформаційно-телекомунікаційні технології та комп’ютерне моделювання», ведеться робота над проектом наукової статті.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ МОДЕЛЮВАННЯ ТА ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ ЗА ФРАКТАЛЬНИМИ ВЛАСТИВОСТЯМИ ТА РОЗПОДІЛЕНИХ ЕВОЛЮЦІЙНИХ ОБЧИСЛЕНЬ

1.1 Призначення та область застосування

Результатом даного дослідження має бути програмна система, яка за допомогою методу конструктивно-продукційного моделювання автоматично визначає фрактальну конструктивну модель вхідного часового ряду. Дана модель описує поведінку ряду і дозволяє реконструювати вхідний ряд з продовженням що дає змогу прогнозування подальших значень ряду.

Конструктивно-продукційний підхід успішно використовується для конструювання об’єктів та процесів, а також часових рядів.

Даний метод моделювання також дозволяє управляти та аналізувати поведінку процесів та систем, яка представлена часовими рядами.

1.2 Формулювання задачі та її складових

Результатом роботи повинна бути програмна система для автоматизації знаходження фрактальної моделі, яка описує вхідний часовий ряд та може бути використана для реконструювання ряду з продовженням.

Система повинна відповідати наступним вимогам:

- мінімальна потреба дій зі сторони користувача під формування фрактальної моделі, що описує заданий часовий ряд;
- реалізація механізму протоколювання для спостереження над проміжними результатами роботи системи;
- інтеграція системи зі сторонніми сервісами відправки та отримання програмних та користувацьких повідомлень;
- фонова робота системи на робочих машинах;
- забезпечення використання не більше 50% відсотків обчислювальних ресурсів кожної машини для можливості роботи інших додатків.

1.3 Аналіз існуючих програмних аналогів та формулювання сильних та слабких сторін

На даний момент не було знайдено програмного забезпечення яке використовує метод конструктивно-продукційного моделювання для відтворення фрактальної моделі вхідного часового ряду.

Але існує значна кількість додатків та модулів для роботи з часовими послідовностями. Вони використовуються для обробки часових рядів та аналізу.

1.3.1 Використання Microsoft Excel в роботі з часовими послідовностями

Excel являється одним з найпопулярніших редакторів таблиць з обширним набором інструментів для математичного опрацювання даних. До таких інструментів входять й інструменти для аналізу та прогнозу часових послідовностей.

Для аналізування часового ряду використовується інструмент «Аналіз даних» (рис. 1.1 – 1.2). Для прогнозування часової послідовності використовується інструмент «Експоненціальне зглажування» (рис. 1.3).

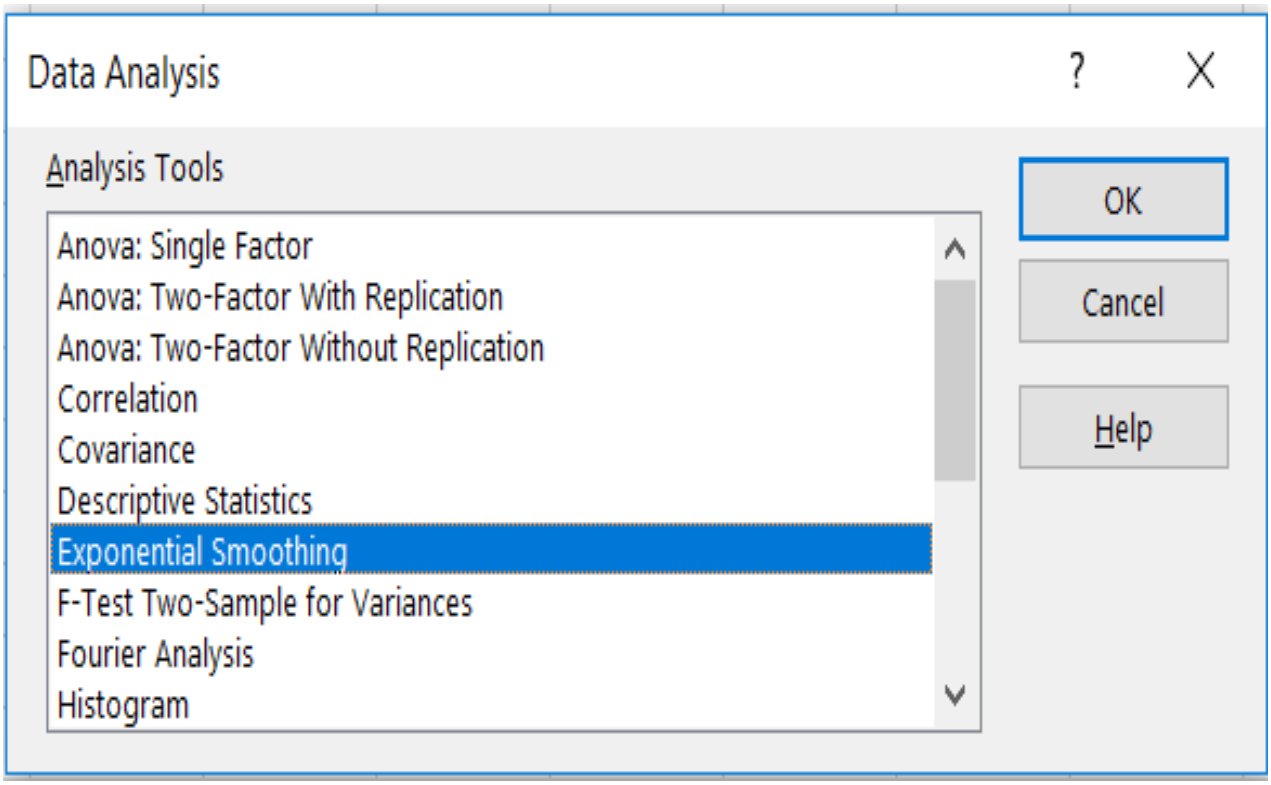


Рисунок 1.1 – Інструменти для аналізу даних в Excel

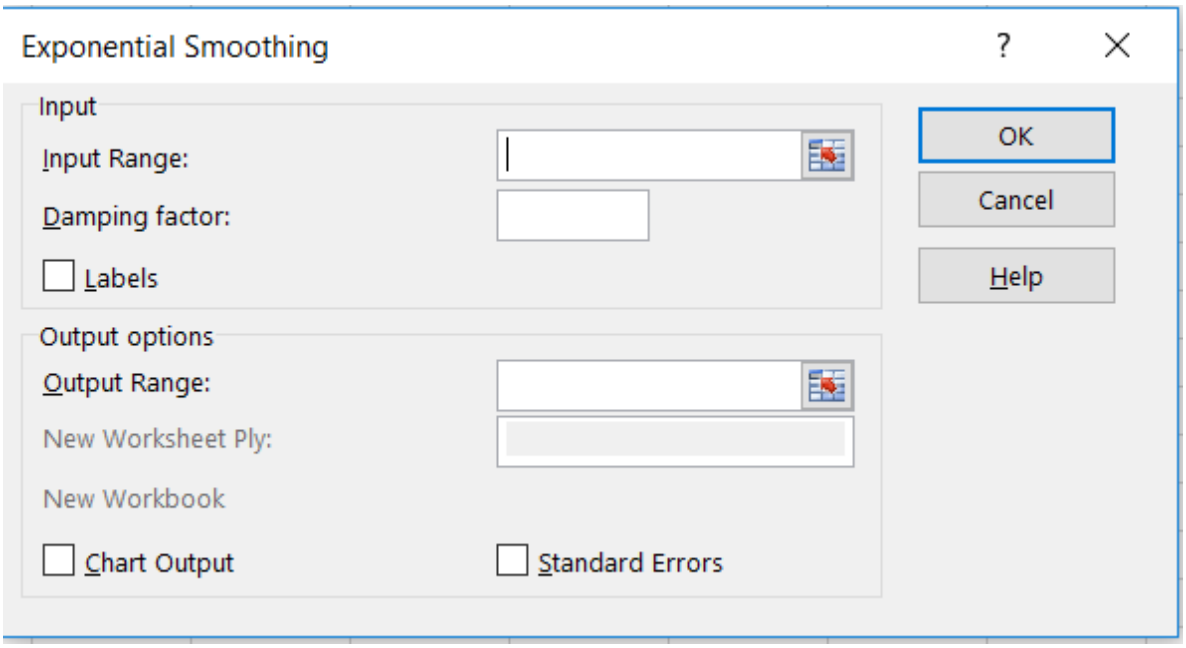


Рисунок 1.2 – Експоненціальне зглажування

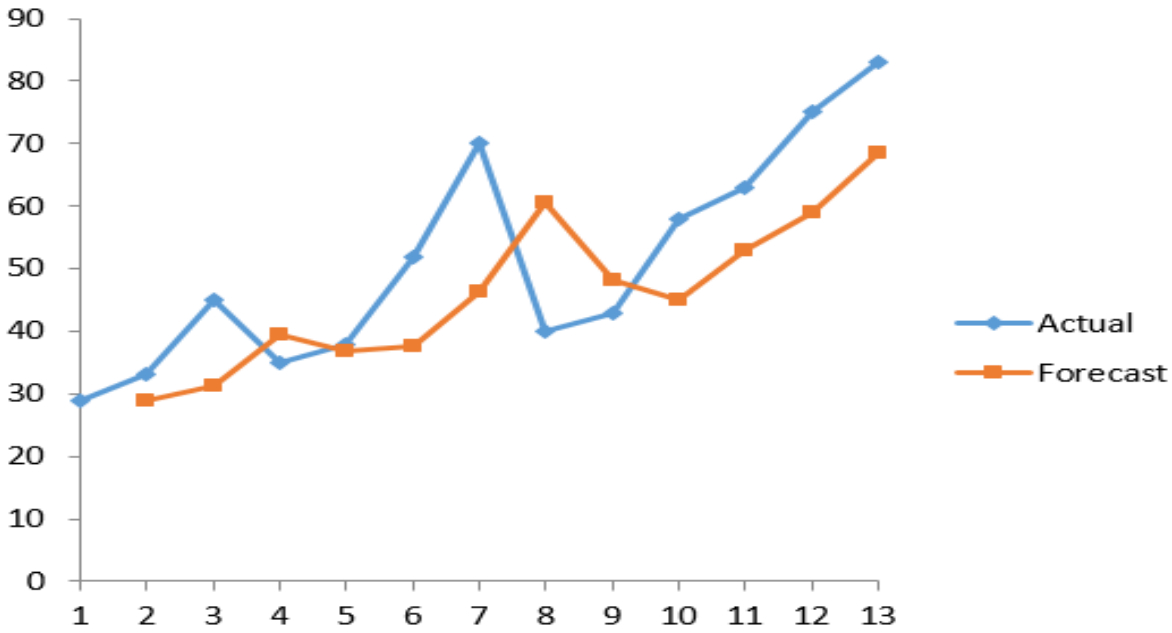


Рисунок 1.3 – Погнозування методом експоненціального зглажування

Переваги Excel наступні:

– багатий функціонал для вирішення різних задач;
– необхідний високий рівень професійної кваліфікації для ефективної роботи з додатком;
– потребує тісної взаємодії користувача з додатком, що збільшує вірогідність помилки.

1.3.2 Використання LGenetic для моделювання фрактальних часових рядів

Програмний додаток LGenetic є результатом досліджень щодо адаптації методу конструктивно-продукційного моделювання для моделювання фрактальних часових рядів [6]. В ході досліджень було розроблено метод для моделювання часових рядів який буде використовуватися в роботі мультиагентної системи.

На рис 1.4 зображено додаток LGenetic в процесі виконання обчислень.

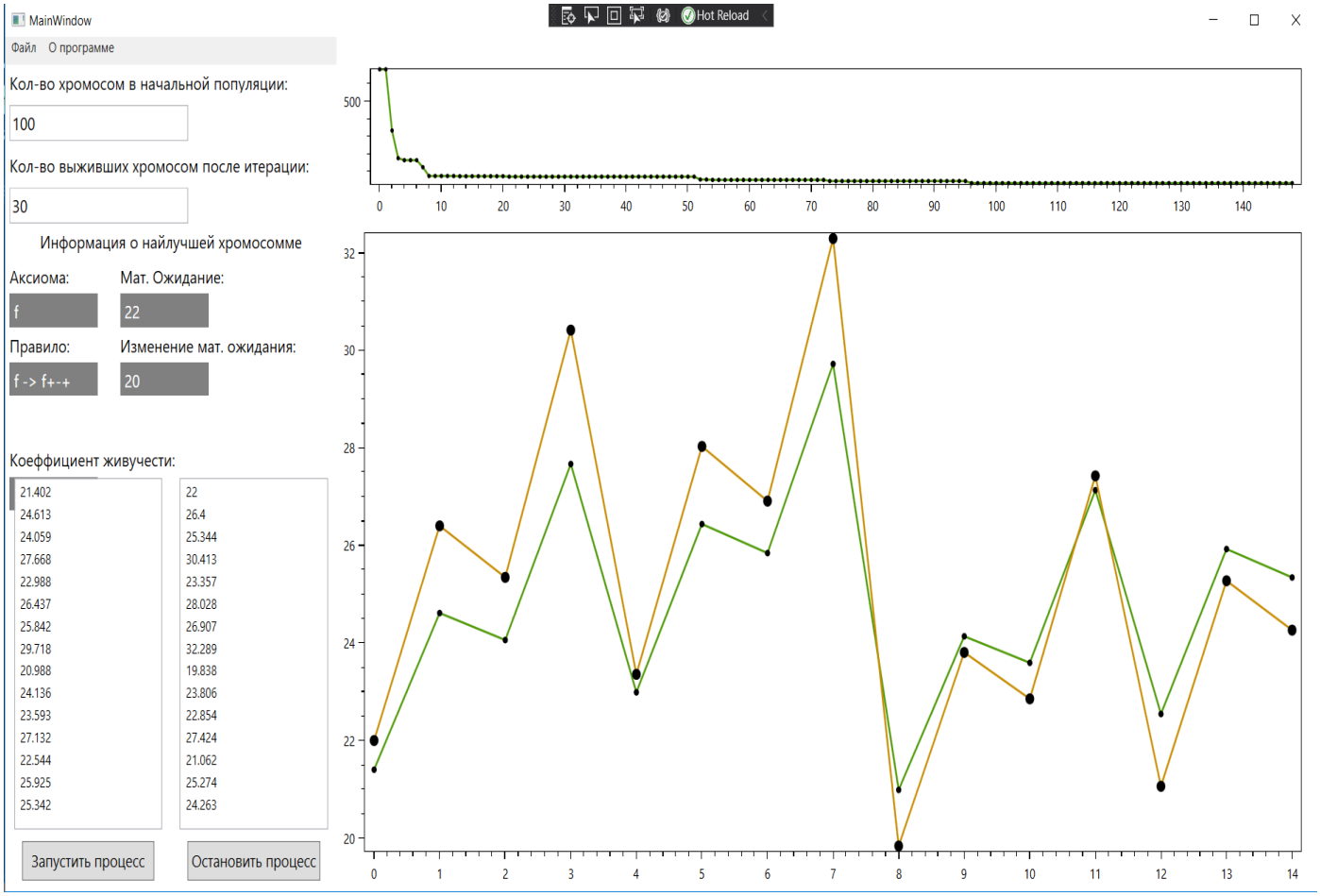


Рисунок 1.4 – Моделювання часового ряду за допомогою додатку LGenetic

Даний програмний додаток єдиний в своєму роді який використовує адаптацію методу конструктивно-продукційного моделювання для моделювання конструктивних моделей вхідних часових рядів.

З сильних сторін додатку можна виділити наступні:

– детальна візуалізація характеристик обчислень в процесі роботи;
– зручний ненавантажений інтерфейс;
– можливість манінуплювання налаштуваннями генетичного алгоритму.

1.4 Аналіз існуючих математичних підходів для конструювання та прогнозу часових рядів.

На даний момент існує маса методів які використовуються для прогнозування часових рядів. Всі методи поділяються на дві групи: алгоритмічні та аналітичні методи.

Відомими алгоритмічними методами є:

- авторегресивна ковзаюча середня [7];
- медіанний фільтр [8];
- нейромережевий аналіз з роздільної здатністю [9];
- нейромережеві багаторівневі мережі уваги [10];
- просторово-часові мережі;

Відомими аналітичними моделями є:

- стохастичні диференціальні рівняння [11];
- динамічний Баєсівський прогностичний синтез [12].

До переваг алгоритмічних методів слід віднести їх високу часову ефективність у використанні в задачах прогнозування, але такі моделі неможливо використовувати для вирішення проблем управління та структурного аналізу. Аналітичні моделі не здатні якісно працювати з осцилюючими рядами.

1.5 Аналіз використання мультиагентного підходу в проектуванні систем

Базове поняття, яке лежить в основі теорії мультиагентних систем є поняття агенту [13] як основної одиниці в мультиагентній системі.

В мультиагентній системі агент являє собою об’єкт котрий спроможний сприймати та діяти в контексті системи. Виходячи з цього у кожного агента є набір сенсорів за допомогою яких він сприймає повідомлення від інших агентів та зміни навколишнього середовища та набір актуаторів, за допомогою яких агент впливає на інших агентів в системі та навколишнє середовище. В контексті вирішення задачі формування фрактальної моделі часового ряду методами конструктивно-продукційного моделювання мультиагентний підхід надає можливість практично безмежного масштабування обчислювальних ресурсів за рахунок використання

множини робочих машин. Це дає змогу значно підвищити часову ефективність роботи системи.

1.5.1 Аналіз мультиагентної архітектури програмних систем

Як і в кожному підході, мультиагентний підхід має свої особливості. В даному підході закладено принцип складної поведінки системи при відносно простій поведінці кожного агента. Для забезпечення консистентності системи агент повинен володіти наступними властивостями:

- активність – агент здатний до організації і реалізації дій (відповідно до внутрішнього алгоритму функціонування);
- автономність – відносна незалежність агенту від навколишнього середовища;
- цілеспрямованість – наявність власних джерел мотивації, що означає, що у кожного агенту є деяка ціль, для досягнення якої він функціонує.

Як було зазначено вище у кожного агенту є деяка ціль, групу агентів які мають однакові цілі об’єднують в клас агентів.

Виходячи з того що структура мультиагентних систем дуже близька до структур в реальному світі, область застосування МАС дуже обширна:

- робототехніка;
- логістика;
- промисловість;
- розподілені обчислення.

Вище наведені лиш деякі області застосування, потенціал даного підходу неймовірно великий і на даний момент не реалізований в повну силу.

1.5.2 Огляд способів комунікації агентних сутностей в рамках системи

Виходячи з концепції мультиагентних систем можна зробити висновок що для функціонування системи необхідно розробити ефективний спосіб взаємодії агентів з навколишнім середовищем та між собою.

За способом локалізації агентів комунікація між агентами може здійснюватися:

– засобами операційної системи (міжпроцесна взаємодія);

– засобами мережевої взаємодії (мережа Інтернет).

В свою чергу міжпроцесна взаємодія в рамках однієї машини може реалізовуватися за допомогою:

– файлів;

– сигналів;

– сокетів;

– іменованих та неіменованих каналів;

– семафорів;

– розділяємої пам’яті;

– віртуальних файлів.

Засоби взаємодії через мережу інтернет поділяються на дві групи в залежності від розташування робочої машини відносно механізму перетворення IP-адрес транзитних пакетів (NAT) [14]:

– попереду NAT;

– за NAT.

Попереду NAT розташовуються машини які мають загальнодоступну адресу в мережі, тому для взаємодії з ними можна використовувати наступні протоколи:

– HTTP;

– TCP/IP (WebSocket);

– JSON-RPC;

– асинхронна черга повідомлень на базі протоколу AMQP [15].

Для взаємодії агентів які розташовані за NAT необхідно використовувати деякий проміжний вузол (Proxy) , який буде доступним для всіх агентів системи і тим чи іншим способом буде перенаправляти кожне повідомлення від одного агента до другого, в свою чергу даний вузол не має ніякої когнітивної ролі системи, його єдина роль маршрутизувати повідомлення між агентами. З таких способів можна виділити:

– асинхронна черга повідомлень на базі проколу AMQP;

– протокол HTTP в зв’язці з WebSocket;

– протокол HTTP в зв’язці з технологією тунелювання.

Вище було приведено багато різноманітних способів комунікації між агентами. Кожен метод взаємодії має як слабкі так і сильні сторони. В контексті мультиагентної системи конструктивно-продукційного моделювання фрактальної моделі часового ряду найбільш прийнятним способом взаємодії є асинхронна черга повідомлень на базі протоколу AMQP.

Інші підходи для комунікації так або інакше потребують розробки додаткових інструментів, або процесу хостингу додатка. В першому випадку зростає складність і час процесу розробки системи, а в другому зростає ціна експлуатації.

1.6 Огляд літератури

1.6.1 Конструктори та композитні конструктори

Поняття конструктора є базовим для конструктивно-продукційного моделювання, конструктор може бути описаний наступним чином:

$$C = \langle M, \Sigma, \Lambda \rangle \quad (1.1)$$

де M — оновлюваний гетерогенний носій, Σ — відносини та операції зв’язані з ними, Λ — набір тверджень з інформацією для підтримки процесу конструювання:

– умови початку та завершення конструювання;правила граматики та обмеження,
– онтологія.

Характеристиками конструктивно-продукційного моделювання є:

– атрибути елементів;
– проміжні конструкції;
– відношення та операції;
– виконавча модель;
– зв'язок операцій з алгоритмами реалізації;
– оновлюваний гетерогенний носій.

Отримані конструкції позначають як $\Omega(C)$.

1.6.2 Особливості середовища розробки Visual Studio Community Edition

Visual Studio – інтегроване середовище розробки від компанії Майкрософт. Продукт включає обширний об’єм інструментів та мов програмування в залежності від потреб розробника. Основною мовою для розробки являється C# на платформі .NET, яка активно пропагандується як інший продукт компанії.

Середовище розробки має великий набір інструментів для розробки, тестування, профілювання навантаження і багато чого іншого.

На даний момент активно ведеться підтримка і подальше покращення .NET Core який являється кросплатформенною заміною .NET Framework.

Розробка даної роботи велась з використанням Visual Studio 2019 Community Edition. Причиною даного вибору були:

– вибір C# на платформі .NET в якості основної мови та платформи для розробки системи;
– дане середовище являється безкоштовним;
– попередній досвід використання.

1.6.3 Особливості мови C# та обгрунтування її вибору для розробки системи

На даний момент мова програмування C# є однією з найпотужніших, швидко розвиваюча і затребуючих мов в IT-секторі. На даній мові розробляються різні додатки: від невеликих настільних програм до крупних розподілених систем и сервісів, які обслуговують кожен день мільйони користувачів.

C# є об’єктно-орієнтованою мовою і в цьому плані багато перейняв у Java і C++. Наприклад, C# підтримує поліморфізм, наслідування, перевантаження операторів, статичну типізацію. Об’єктно-орієнтований підхід дозволяє вирішувати задачі для конструювання великих і в той час гнучких, масштабованих и розширюваних додатків та систем. Також C# активно розвивається і з кожною новою версією з’являється все більше і більше цікавих функціональностей платформи.

Можна виділити наступні особливості C# на платформі .NET:

– підтримка декількох мов програмування. Основою платформи являється

загальне середовище виконання Common Language Runtime (CLR), завдяки цьому .NET підтримує декілька мов програмування, наряду з C# це також VB.NET, C++, F#. При компіляції код на будь-якій з цих мов компілюється в сбірку на загальній мові CIL (Common Intermediate Language) – свого роду ассемблер платформи .NET;

– кросплатформеність. .NET являється платформою яка переноситься. Наприклад .NET 5 підтримується на більшості сучасних ОС Windows, MacOS, Linux;
– потужна бібліотека класів. Загальномовне середовище виконання CLR і базова бібліотека класів є основною для цілого стека технологій, які розробники можуть задіяти при конструюванні тих чи інших додатків;
– продуктивність. Згідно ряду тестів веб-додатки на .NET 5 в ряді категорій сильно випереджають веб-додатки, побудовані за допомогою інших технологій. Додатки на .NET в принципі відрізняються високою продуктивністю роботи.

1.6.4 Еволюційні обчислення в контексті генетичного алгоритму

Генетичний алгоритм – це евристичний алгоритм пошуку оптимального вирішення деякої задача, який використовується для вирішення задач моделювання та оптимізації шляхом послідовного підбору, комбінування і варіації параметрів з використанням механізмів, які нагадують біологічний еволюційний процес. Алгоритм являється різновидом еволюційних обчислень. Особливістю генетичного алгоритму являється акцент на використання оператора схрещування та мутації, які проводять операції зміни генів вирішень-кандидатів, роль яких аналогічна ролі мутацій та схрещування в живій природі.

Сильної стороною даного методу обчислень являється його проста адаптація до вирішення задачі в умовах коли задачу неможливо вирішити аналітично, або метод для вирішення ще не був знайдений. Так виходить через те, що сам алгоритм являється формалізованим методом і не накладає ніяких обмежень на фактичну його реалізацію.

Слабка сторона даного підходу криється в необхідності значних обчислювальних ресурсів та часу для отримання оптимального результату, ця проблема являється загальної проблемою через яку евристичні методи програють за часовою ефективністю аналітичним.

1.6.5 Взаємодія агентів в середовищі мультиагентної системи

Для взаємодії агентів в контексті мультиагентної системи яка методом конструктивно-продукційного моделювання відтворює фрактальну конструктивну модель вхідного часового ряду було обрано асинхронну чергу повідомлень на базі протоколу AMQP. Програмний інструмент RabbitMQ являє собою вузол який розташовується як загальнодоступний сервіс для обміну повідомлень між розподіленими компонентами системи, в даному випадку між агентами системи.

Відкритий код RabbitMQ-брокера написаний на мові програмування Erlang. Erlang був розроблений компанією Ericsson в середині 1980-х як розподілена, відмовостійка, система реального часу для додатків, які потребують безвідмовної роботи 99.999%.

RabbitMQ – це брокер повідомлень з відкритим кодом. Він маршрутизує повідомлення за всіма базовими принципами протокола AMQP [15] які описані в специфікації протоколу [16].

1.7 Формування вимог до системи

1.7.1 Вимоги отримані від зацікавлених осіб

Для формування вимог до системи було проведено опитування спеціалістів.

Результати опитування представляються у вигляді наступних тверджень і вимог, які необхідно буде досягти:

- система повинна бути гнучкою для інтегрування в неї нових типів агентів;
- система повинна раціонально працювати з обчислювальними ресурсами кожної робочої машини, робота сторонніх додатків на машині повинно бути можливе під час роботи системи;
- інтерфейс користувача повинен бути простим.

1.7.2 Результати та висновки огляду літератури

В результаті огляду літератури було сформовано вимоги до системи:

- система повинна розроблятися з точки зору оптимального використання інструментів обраної платформи та мови програмування які було образні для процесу розробки;
- система повинна бути розроблена на мові програмування C# [17] на платформі .NET [18];
- система повинна використовувати метод конструктивно-продукційного моделювання на базі розподіленого генетичного алгоритму;
- комунікація між агентами системи повинна здійснюватися за допомогою асинхронної черги повідомлень на базі протоколу AMQP RabbitMQ;
- всі агенти системи повинні мати змогу легко розгортатись на робочих машин без надання машині публічного IP [19] адреси в мережі Інтернет.

Висновки до першого розділу

В даному розділі було розглянуто формалізацію мети завдання, доцільність досліджень в даній області та актуальність в науковій діяльності, практичне значення майбутньої системи. Також було розглянуто програмні засоби які мають схожу область використання.

Були досліджені основні вимоги до програмного забезпечення. Опираючись на основні вимоги були досліджено архітектурні особливості проектування та розробки мультиагентних систем та способи взаємодії агентів в контексті системи.

Потребують подальшого вивчення питання проектування та розробки мультиагентних систем та структури розподілених еволюційних обчислень на базі генетичного алгоритму, а також способів підвищення часової ефективності методу конструктивно-продукційного моделювання та розширення області його застосування для стохастичних та недетермінованих рядів.

Для подальших досліджень необхідно розробити інструментарій мультиагентної системи з базовим функціоналом для роботи з конструктивізмом в контексті часових рядів та механізми протоколювання стану роботи системи для аналізу результатів.

2 ОБГРУНТУВАННЯ НАПРЯМКУ ДОСЛІДЖЕННЯ МУЛЬТИАГЕНТНОГО ПІДХОДУ ДЛЯ КОНСТРУКТИВНО-ПРОДУКЦІЙНОГО МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ СКЛАДОВИХ ЧАСОВИХ ПОСЛІДОВНОСТЕЙ

2.1 Формування основної мети та задач дослідження

Мета даного дослідження полягає в підвищенні часової ефективності для методу конструктивно-продукційного моделювання для формування фрактальної моделі часового ряду за рахунок масштабування обчислювальних ресурсів за допомогою мультиагентного підходу, а також можливість використання даного методу для роботи зі стохастичними рядами.

Даний метод працює на базі того, що фрактальні ряди мають властивості самоподібності, тому структуру ряду можна описати за допомогою фрактальної моделі, яка потім буде використовуватися для реконструювання ряду, в тому числі з продовженням, що дозволяє ефективно проводити прогноз майбутніх значень і показників.

Була поставлена наступна задача: детермінований або стохастичний фрактальний часовий ряд генерується за допомогою L-систем [21] з деяким компонентом випадковості, що вносить в структуру ряду стохастичну природу. Також в якості експерименту було прийняте рішення взяти деякий реальний часовий ряд наприклад графік сердцебиття людини. Необхідно відтворити фрактальну модель, яка складається з:

Конструктор на основі L-систем з урахуванням математичного очікування значення ряду, відхилення математичного очікування та дисперсії математичного очікування.

2.2 Обгрунтування методу конструктивно-продукційного моделювання як методу вирішення поставленої задачі

Конструктивно-продукційний підхід успішно використовується як метод моделювання конструкцій на основі L-систем, в тому числі за допомогою цього методу можна конструювати фрактальні часові ряди. На рис. 2.1 – 2.5 наведено

приклади згенерованих конструкцій (часових рядів) які були побудовані за допомогою L-систем які описують вхідний ряд.

Використовуючи даний підхід можна вважати, що можливо провести процес обернений до даного, хоча метод для виведення L-систем з конструкції ще не був винайдений але неможливість даного процесу не була доведена теоретично.

Основним компонентом для методу конструктивно-продукційного моделювання являється конструктор [22]. Було винайдено особливий тип конструкторів – породжуючі конструктори. Породжуючі конструктори використовуються для створення наборів конструкцій або процесів на основі початкових даних та умов, умов закінчення конструювання та правил заміщення.

Нижче наведено приклади побудови детермінованих часових рядів на основі конструктора L-систем з наступними позначеннями:

- [S] – аксіома, початковий символ конструювання;
- [f],[g],[h] – нетермінальні символи які представляють собою значення ряду;
- [+],[-] – термінальні символи які представляють собою операції зміни збільшення та зменшення математичного очікування значення ряду;
- [M] – математичне очікування;
- [dM] – відхилення математичного очікування;

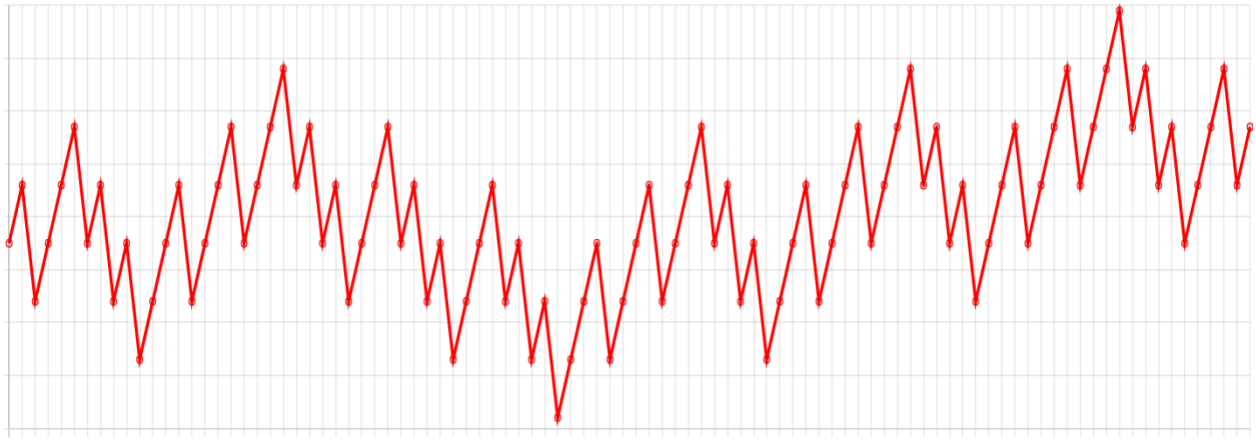


Рисунок 2.1 – Графік часового ряду для конструктивної моделі:

$$S \to f,\ f \to f - f + + f - f,\ M = 13,\ dM = -2.2$$

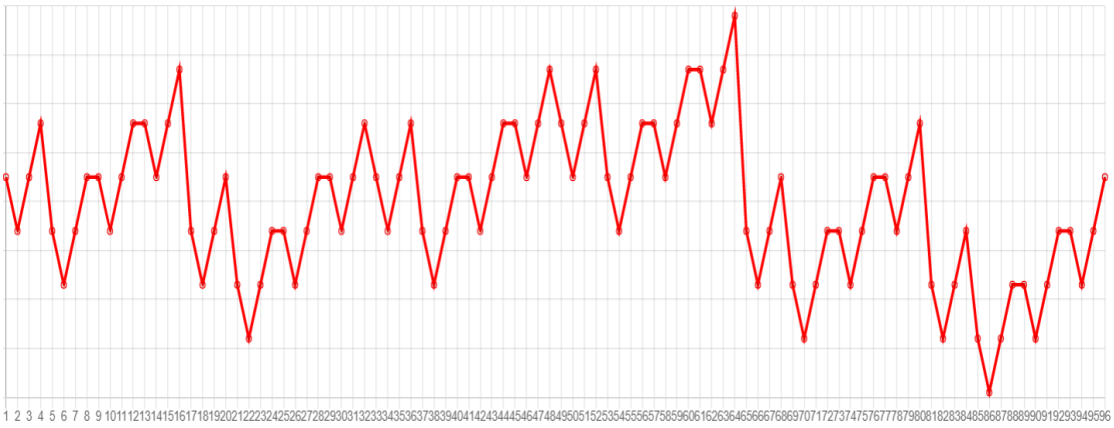


Рисунок 2.2 – Гафік часового ряду для конструктивної моделі:

$$S \to f, f \to f + f - f - f +, M = 13, dM = -2.2$$

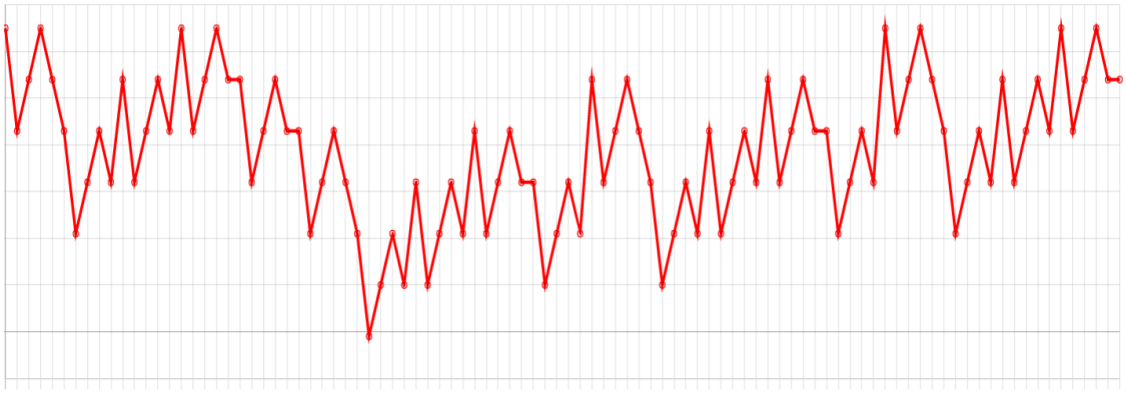


Рисунок 2.3 – Графік часового ряду для конструктивної моделі:

$$S \to f, f \to f + + f - f - f + f -, M = 13, dM = -2.2$$

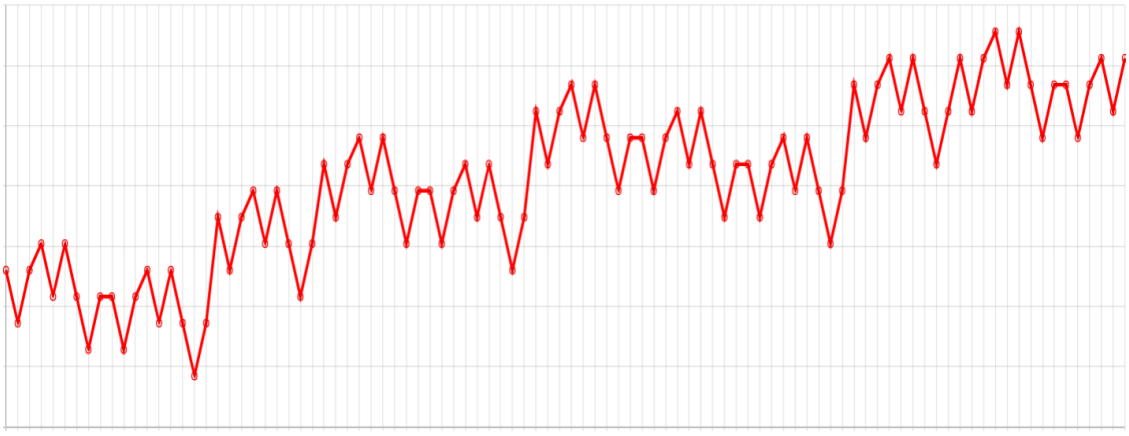


Рисунок 2.4 – Графік часового ряду для конструктивної моделі:

$$S \to f, f \to g + + g - - g, g \to f - f + + f - -, M = 12, dM = -2.2$$

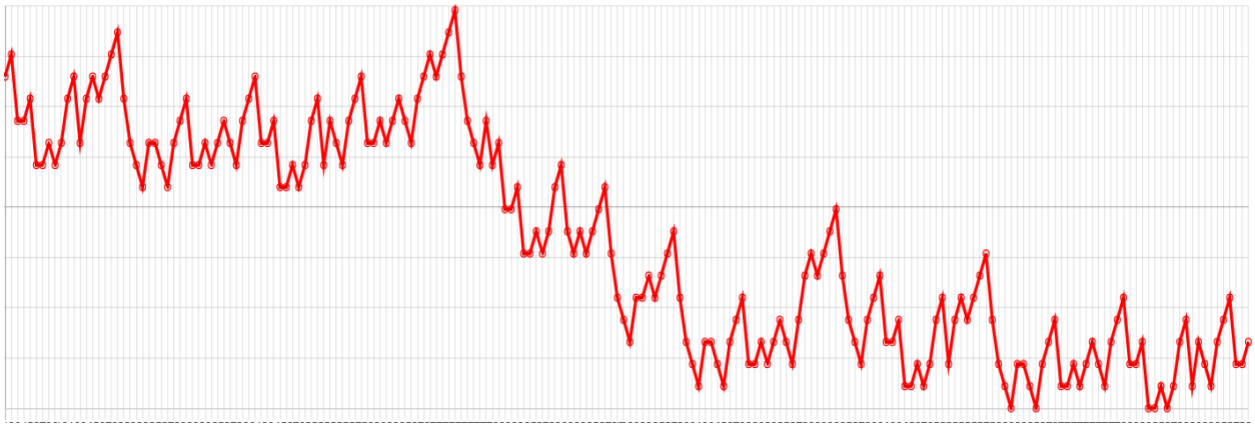


Рисунок 2.5 – Графік часового ряду для конструктивної моделі:

$S \to f, f \to g+g+h--g, g \to h-g+++f-, h \to f-g+h-f, M=13, dM=-2.2$

Як видно з наведених прикладів часових рядів які були зконструйовані за допомогою різних L-систем складність структури часового ряду прямо пропорційна складності вхідної L-системи. Чим складніша L-система, тим більш складний процес може бути породжений на основі неї. При достатньо складних L-системах (рис. 2.5), природа часового ряду все більш схожа на стохастичну, що дозволяє припустити що метод конструктивно-продукційного моделювання можна успішно використовувати для формування фрактальної моделі не тільки для детермінованих рядів, але і для стохастичних послідовностей.

2.3 Обгрунтування використання еволюційних обчислень в контексті генетичного алгоритму для формування фрактальної моделі вхідного часового ряду.

Як вже було зазначено вище, до цього часу аналітичних методів для відновлення L-системи за допомогою конструкції породженої нею, що являється оберненим процесом до процесу конструктивно-продукційного моделювання, не було винайдено, хоча неможливість існування такого методу не було теоретично доведено. Таким чином, для вирішення поставленої задачі слідує сфокусувати увагу на еврістичних методах [23], одним з яких є генетичний алгоритм [24].

Генетичний алгоритм як евристичний метод вже доволі довго показує свою ефективність роботи для вирішення задач оптимізації. Популярність даного методу обумовлюється його простотою для адаптації під конкретну задачу та мінімальною кількістю обмежень, що спрощує процес розробки та дозволяє розробляти деякі

нововведення та покращення алгоритму які позитивно впливають на часову ефективність.

2.4 Обгрунтування використання мультиагентного підходу для розробки системи

В результаті попередніх досліджень використання методів конструктивно-продукційного моделювання для відтворення фрактальної моделі вхідного часового ряду на базі генетичного алгоритму та проведення відповідних експериментів було з’ясовано що даний метод потребує значних обчислювальних ресурсів для забезпечення потрібної якості формування моделі в адекватних часових рамках.

Обмеження попередніх досліджень полягали в відтворенні фрактальної моделі тільки для детермінованих часових рядів, L-система яких обмежувалася невеликою кількістю правил заміщення та малою довжиною цих правил. Як було зазначено вище, складність вхідного ряду та його стохастична компонента різко впливають на складність фрактальної моделі. В свою чергу при відтворенні складної фрактальної моделі порівняно з простою можна спостерігати експоненціальний ріст в потребі часу на процес обчислень. Виходячи з цього необхідно розробити ефективний спосіб для збільшення часової ефективності даного методу.

Збільшити часову ефективність обчислень можна за рахунок:

- оптимізації методу обчислень;
- використання більш продуктивних в плані часової ефективності мов програмування та платформ для розробки системи;
- використання апаратного обладнання з більш потужними характеристиками для проведення обчислень;
- масштабування обчислювальних ресурсів в межах однієї робочої машини;
- масштабування обчислювальних ресурсів за рахунок розподілення системи на багато обчислювальних агентів.

Виходячи з вимог сформованих до системи, можна сказати що пункт 4 являється непридатним для використання так як при максимальному споживанні

ресурсів, робота сторонніх програм та додатків в процесі обчислень на робочій машині буде неможливою.

Варто звернути увагу на масштабування ресурсів за рахунок розподілення процесу обчислень між багатьма агентами та розробити мультиагентний підхід для проектування системи, так як це дозволяє горизонтально [25] масштабувати ресурси, що в свою чергу позитивно впливає на часову ефективність.

2.5 Визначення процесу формування фрактальної моделі часового ряду

2.5.1 Визначення архітектури конструкторів

Функціонал відтворення конструктивної моделі ряду закладений в обчислювальному агенті, який використовує наступні типи конструкторів:

- Fractal Chain Constructor – конструктор який приймає як аргументи L-систему, необхідну довжину і будує фрактальний мультисимвольний ланцюжок:
- Time Series Constructor – конструктор який приймає як аргументи фрактальний мультисимвольний ланцюжок, математичне очкування, відхилення математичного очікування, дисперсію та будує фрактальний часовий ряд;
- LSystem Constructor – конструктор який приймає задану конфігурацію, обмеження при побудові випадкової L-системи та будує випадкову L-систему на базі цих аргументів. Цей конструктор використовується для генерації нових хромосом в ході роботи генетичного алгоритму.

2.5.2 Визначення структури генетичного алгоритму

Було визначено функціонал генетичного алгоритму для формування конструктивної моделі.

На початку роботи генетичний алгоритм отримує вхідний часовий ряд, конфігурацію генетичного алгоритму та умови завершення процесу обчислень.

Генетичний алгоритм запускає свою роботу генеруючи першу популяцію з хромосом за допомогою конструктору L-систем використовуючи задану конфігурацію.

Після генерації хромосом в потрібній кількості генетичний алгоритм проводить фазу мутації [26], при цьому слід відмітити, що початкова хромосома залишається в популяції в первинному вигляді, а в популяцію додається її змутований клон.

Після фази мутації проводиться фаза кроссоверу [27], за допомогою якої відбувається рекомбінація генів в популяції. Слідує відмітити, що фази кроссоверу та мутації залежать від конфігурації генетичного алгоритму, тому при розподілені процесу обчислень між агентами системи кожен агент проводить дані фази по різному, що позитивно впливає на сумарну часову ефективність процесу обчислень за рахунок розширення глобальної області пошуку оптимального рішення і забезпечення достатньої різноманітності генофондів та роботи над ними.

Наступна фаза називається фазою селекції [28]. Метою фази селекції є відібрати найбільш пристосовані хромосоми з популяції опираючись на їх фітнес функцію, яка обчислюється конструюванням часового ряду на базі моделі яка знаходиться в хромосомі і порівнянням отриманого часового ряду з вхідним методом середньо квадратичного відхилення.

Остання фаза генетичного алгоритму являється нововведенням і є опціональною. В процесі проведення експериментів було виявлено що в процесі роботи генетичного алгоритму популяція деградує за рахунок засмітнення більшої частини популяції схожими генами, відсутність різноманітності генофонду провокує сильне зменшення динамічної оптимізації результатів від популяції до популяції, тому був запроваджений метод «глобального катаклізму» над популяцією. Суть методу полягає в тому щоб при досягненні певного рівня виродження необхідно знищити майже всю частину генофонду, при цьому залишити дуже малий відсоток кращих і заповнити популяцію хромосомами створеними випадковим чином. Цей метод сильно прискорює процес обчислень і досягнення потрібного результату.

2.5.3 Визначення схеми розподіленої роботи генетичного алгоритму

Так як система проектується за мультиагентним підходом, актуальним питанням є організація роботи генетичного алгоритму при розподіленні процесу обчислень на множину робочих машин [29].

Основними аспектами роботи генетичного алгоритму є:

– схема роботи алгоритму;
– міграція хромосом.

Схема роботи розподіленого генетичного алгоритму означає спосіб розподілення функцій алгоритму між машинами. Основними схемами роботи є:

– схема «майстер-робітник». Як правило застосовується коли структура хромосоми є доволі складної і залежить від великої кількості параметрів, що сильно ускладнює процес обчислення фітнес функції. Існують деякі приклади задач, коли обчислення фітнес функції займає дуже велику кількість часу, тому для оптимізації роботи генетичного алгоритму і масштабування ресурсів використовується схема «майстер-робітник»;
– схема «островів». Відрізняється від попередньої тим, що при використанні даного підходу кожна машина яка приймає участь в процесі обчислень має свою незалежну копію генетичного алгоритму, при цьому необхідно запровадити процес міграції між «островами» для використання переваги розподілених обчислень.

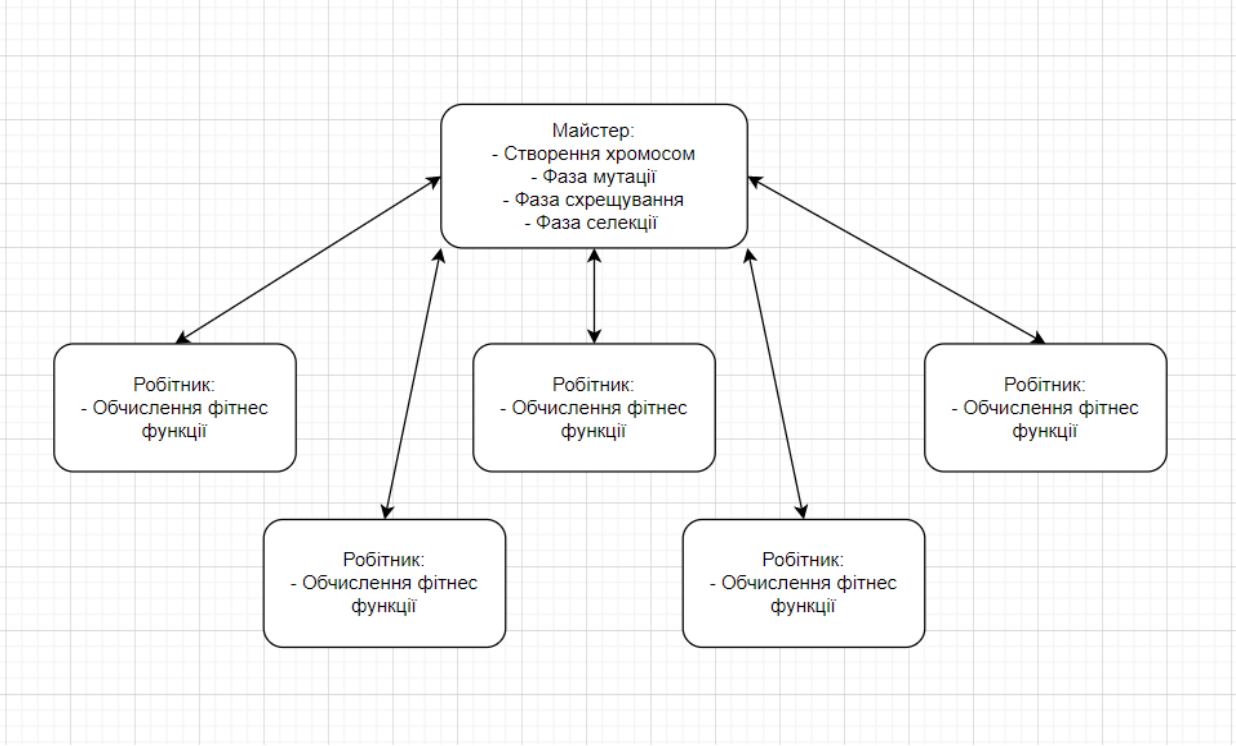


Рисунок 2.6 – Схема роботи генетичного алгоритму «майстер-робітник»

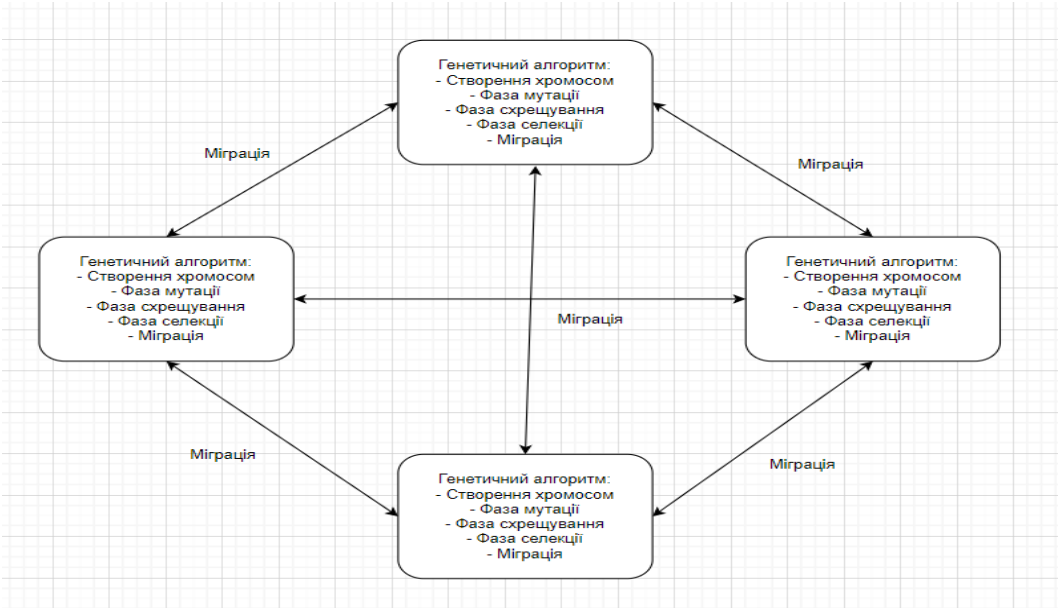


Рис. 2.7 – Схема роботи генетичного алгоритму «схема островів»

В контексті даної задачі більш прийнятним для використання являється схема «островів» так як фітнес функція в даному випадку не займає велику кількість часу, з цього можна зробити висновок що делегування обчислень яке передбачається проводити через мережу не має сенсу, так як займає більше часу ніж безпосередньо обчислення цієї функції.Виходячи з вибору схеми «островів» для роботи генетичного алгоритму необхідно розробити схему міграції хромосом між генофондами. З відомих способів міграцій можна виділити наступні:

– однорангова міграція;
– міграція за допомогою допоміжного банку хромосом.

Однорангова міграція хромосом між генофондами являє собою обмін хромосомами за чітко заданою схемою або довільним чином безпосередньо між агентами, такий підхід відрізняється простотою в реалізації і малим обмеженням в процесі обміну хромосомами, що дозволяє запроваджувати і розробляти нескінченну кількість способів обміну.

Але в процесі експериментів та огляду літератури на дану тематику було виявлено що процес обміну в такий спосіб прискорює динаміку виродження всіх генофондів, що дуже негативно впливає на різноманітність генофондів, що в свою чергу зменшує часову ефективність генетичного алгоритму.

Міграція за допомогою допоміжного банку хромосом є видозміненим способом обміну хромосомами між генофондами, в основі якого лежить принцип делегування процесу міграції окремому компоненту системи. При використанні даного способу процес контролю та управління міграцією може включати різноманітні інтелектуальні інструменти, такі як нечітка логіка або нейронні мережі, за рахунок використання яких можна максимально оптимізувати процес міграції хромосом, що в свою чергу забезпечує більшу ефективність роботи розподіленого генетичного алгоритму. Виходячи з цього було обрано саме спосіб міграції через допоміжний банк хромосом, що в контексті системи буде називатися міграційним агентом.

2.5.4 Визначення структури одиниці генофонду генетичного алгоритму

На даному етапі була сформована структура хромосоми генетичного алгоритму як атомарна сутність в контексті алгоритму. Виходячи з канонічного опису генетичного алгоритму як евристичного методу обчислень слідує відмітити, що кожна хромосома входить до множини рішень деякої задачі і робота алгоритму полягає в тому щоб знайти найбільш оптимальне рішення в межах відведених ресурсів на його роботу. В контексті даної роботи і поставленої задачі, рішенням є конструктор на основі L-системи з урахуванням математичного очікування, відхилення математичного очікування та дисперсії.

Як відомо, генетичний алгоритм являється евристичним методом який описується формально і не має чітких обмежень для процесу обчислень. Це означає, що процес роботи генетичного алгоритму залежить від множини параметрів його роботи, з таких параметрів можна виділити наступні:

- кількість хромосом в популяції;
- вірогідність мутації;
- вірогідність кроссоверу;
- кількість хромосом для селекції.

Маніпулювання даними параметрами може позитивно або негативно впливати на часову ефективність обчислень за допомогою генетичного алгоритму.

Тим не менш, слідує відмітити, що в залежності від різних вхідних даних різний набір даних характеристик може показувати різні показники ефективності. Виходячи з цього необхідно розробити процес конфігурації генетичного алгоритму, а також процес конфігурації обмежень для конструювання L-системи.

2.6 Визначення структури системи в контексті мультиагентної архітектури

2.6.1 Визначення ролей та обов’язків агентів в системі

Мультиагентний підхід як спосіб проектування і розробки систем які працюють на множині робочих машин передбачає розподілення системи на множину агентів, які проводять певну роботу спільно для досягнення певної мети. При цьому множина всіх агентів ділиться на класи агентів [30]. Кожний клас агентів має свою певну мету, цілі та мотивацію, також кожний агент має певний ступінь автономності і ізольованості від інших агентів. Множина агентів які належать одному класу агентів мають однакову мету, цілі, мотивацію, знання та ступінь ерудованості в контексті системи.

У табл. 1 наведені обов’язки та знань кожного класу агентів.

Таблиця 1 – Обов’язки та знання агентів

Назва агенту	Обов’язки	Знання
Кластерний агент	Взаємодія з агентом користувача, облік доступних агентів в системі, ініціювання команд процесу обчислень, конфігурація обчислювальних агентів.	Інформація про агентів, конфігурації обчислювальних агентів, специфікація задачі
Агент міграцій	Взаємодія з обчислювальними агентами в контексті міграцій, вивід схеми міграцій в залежності від поточного стану обчислень.	Буфер генофонду, поточний стан обчислень кожного агенту.
Обчислю-вальний агент	Взаємодія з кластерним та міграційним агентами, процес обчислень за допомогою генетичного алгоритму з заданою конфігурацією.	Конфігурація, специфікація задачі.
Агент взаємодії з користувачем	Валідація користувацьких команд з подальшим делегуванням кластерному агенту	Інформація про користувача, інформація про кластерного агента.

2.6.2 Визначення складу системи

Для коректної роботи системи необхідно визначити конфігурацію агентів. Виходячи з вищезазначених визначених класів агентів та на основі їх ролей та знань можна визначити наступну конфігурацію мультиагентної системи:

- агент взаємодії з користувачем - для взаємодії користувача з системою було виділено окремий клас агентів, обов'язками яких має бути делегування користувацьких команд системі та сповіщення користувача про зміну стану системи. Мінімальна конфігурація передбачає обов'язкову наявність та активність що найменш одного агенту з даного класу.
- кластерний агент – процес обчислень в контексті одного користувацького запиту управляється кластерним агентом, в обов'язки якого це входить. Конфігурація передбачає наявність що найменш одного кластерного агенту в системі.
- міграційний агент - в обов'язки агентів даного класу входить управління процесом міграції хромосом між генофондами в межах кластеру.
- обчислювальний агент - обов'язки даного класу агентів – безпосередньо проводити процес обчислень за допомогою генетичного алгоритму, дані агенту незалежно один від одного проводять обчислення, та прослуховують кластерних та міграційних агентів щодо вхідних команд.

2.6.3 Визначення способу комунікації агентів в середовищі системи

Процес обміну повідомленнями між агентами в межах системи являється одним з головних та необхідних процесів для функціонування мультиагентної системи. З головних вимог до способу взаємодії були видвинуті наступні:

- висока пропускна спроможність [31];
- висока досяжність брокера повідомлень в мережі Інтернет;
- мінімізація затримок в доставці повідомлень;
- необхідність пересилання повідомлень за протоколом який гарантує надійну та консистентну передачу повідомлень.

Для задовільнення даних вимог було обрано спосіб передачі повідомлень за допомогою асинхронної черги повідомлень, використовуючи при цьому конкретний продукт під назвою RabbitMQ. RabbitMQ – інструмент з відкритим кодом, який являє собою брокер повідомлень, який працює на базі протоколу AMQP.

2.6.4 Опис варіативності поведінки агентів в залежності від конфігурації

Варіативність роботи агентів в контексті даної роботи відноситься до класу міграційних агентів та обчислювальних агентів. Під варіативністю роботи слідує розуміти різну конфігурацію для таких складових процесу обчислень як:

- робота генетичного алгоритму;
- міграція хромосом між генофондами;
- конструювання L-систем та спосіб підбору математичних характеристик хромосоми.

Маніпулювання та кастомізація даних процесів позитивно впливає на ефективність обчислень в цілому, а також додає гнучкості процесу досліджень, що дозволяє перевіряти гіпотези.

Висновки до другого розділу

В цьому розділі було обгрунтовано напрям даного дослідження. Були сформовані завдання та мета дослідження. Були визначені інструменти, які застосовувалися для вирішення різних задач, формалізовано роботу генетичного алгоритму, процес відтворення конструктивної моделі вхідного часового ряду методом конструктивно-продукційного моделювання. Також було визначено структуру мультиагентної системи, знання та обов’язки кожного класу агентів, конфігурацію системи.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ФОРМУВАННЯ ФРАКТАЛЬНОЇ МОДЕЛІ ЧАСОВОГО РЯДУ ТА ВІДНОВЛЕННЯ ВХІДНОГО РЯДУ

3.1 Зовнішнє проектування системи

3.1.1 Формат вхідних даних

В контексті даної системи вхідні дані поділяються на наступні типи:

– вхідні дані для обчислень;
– конфігурація генетичного алгоритму;
– конфігурація конструювання L-систем;
– конфігурація математичних параметрів конструктивної моделі;
– конфігурація мережевої взаємодії;
– конфігурація логування.

Вхідними даними для обчислень є:

– часовий ряд;
– точність обчислень.

Конфігурація генетичного алгоритму складається з таких параметрів:

– початковий розмір популяції;
– розмір селекції;
– вірогідність кроссоверу;
– тип кроссоверу;
– вірогідність мутації;
– тип мутації;
– максимальний вік хромосоми;
– максимальна концентрація хромосом з максимальним віком в популяції.

Конфігурація конструювання L-систем має такі параметри:

– максимальна довжина аксіоми;
– мінімальна довжина аксіоми;
– максимальна кількість правил заміщення;
– мінімальна кількість правил заміщення;

- максимальна довжина правила заміщення;
- мінімальна довжина правила заміщення.
- Конфігурація математичних параметрів конструктивної моделі складається з таких параметрів:
- максимально допустиме математичне очікування;
- мінімально допустиме математичне очікування;
- максимальне відхилення математичного очікування;
- мінімальне відхилення математичного очікування;
- максимальна дисперсія;
- мінімальна дисперсія.
- Конфігурація мережевої взаємодії агентів в системі складається з таких складових:
- адреса брокера повідомлень;
- ідентифікатор обміну повідомлень.
- Конфігурація логування складається з наступних параметрів:
- тип логування;
- шлях до файлу для запису логів.

3.1.2 Формат вихідних даних

Результатом роботи системи є:

- повідомлення (оповіщення користувача);
- фрактальна конструктивна модель часового ряду;
- лог-файли з записами про роботу програми.

3.2 Попередній етап проектування структури та архітектури системи

На початку розробки програмної системи основною задачею є проектування компонентної та архітектурної моделі [32]. Дана модель повинна описувати основні компоненти та логічні сутності системи та повинна розроблятися у відповідності загальним практикам проектування та розробки ПЗ.

Первинне завдання – виділення агентів [33] як логічних сутностей. Необхідно представити базову структури системи. На рис. 3.1 представлено структуру системи.

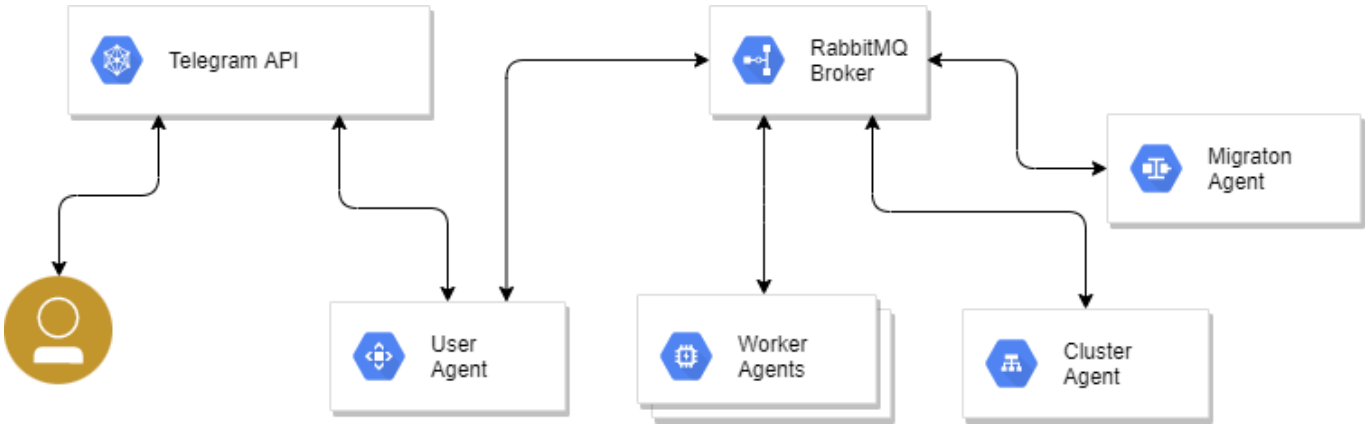


Рисунок 3.1 – Структура мультиагентної системи

На рис. 3.1 зображено структуру та схеми взаємодії окремих компонентів системи. Як видно, вся комунікація між агентами проводиться через брокер повідомлень, тому процес розгортання кожного агенту не потребує щоб робоча машина мала загальнодоступну адресу в мережі Інтернет. Подальше проектування передбачає визначення обов'язків кожного агента.

Агент взаємодії з користувачем являє собою агента системи в обов'язки якого входить підтримувати взаємодію з користувачем, а саме:

– аутентифікацію користувача;
– валідація команд користувача;
– запит даних у користувача;
– делегування команд користувача;
– сповіщення користувача повідомлення від системи;
– надання результату обчислень користувачеві.

Даний агент спілкується з користувачем за допомогою Telegram API [34]. Таким чином користувач маючи додаток Telegram на будь-якому з пристроїв може керувати системою, відправивши команду та необхідні дані.

Кластерний агент є агентом системи в обов’язки якого входить:

– ведення обліку агентів в системі;
– взаємодія з користувацьким, міграційним та обчислювальними агентами;
– управління конфігурацією обчислювальних та міграційного агентів;
– ініціалізація та контроль процесу обчислень;
– агрегування результатів обчислень;
– генерування користувацьких сповіщень від імені системи.

Кластерний агент є головною керуючою ланкою в системі. Взаємодія з іншими агентами проводиться за допомогою асинхронної черги повідомлень RabbitMQ [35].

Обчислювальний агент. Даний клас агентів безпосередньо проводить процес обчислень. В обов’язки даного класу агентів входить:

– за допомогою генетичного алгоритму проводити процес обчислень;
– реагувати на команди зі сторони кластерного та міграційного агентів та обробляти їх запити;
– використовувати задану конфігурацію при проведенні процесу обчислень;
– відправляти запит на закінчення обчислень кластерному агенту при досягненні необхідної точності рішення задачі.

Міграційний агент є керуючою ланкою в контексті міграцій хромосом між обчислювальними агентами (їх генофондами). В обов’язки даного класу агентів входить:

– керувати процесом міграції;
– аналізувати конфігурацію та поточну ефективність обчислень кожного агенту;
– відправка команд та обробка запитів від обчислювальних агентів.

3.3 Виділення структурних та логічних сутностей та зв’язків між ними

3.3.1 Визначення базової архітектури агентів

За основу при розробці архітектури агентів взято гексагональну архітектуру. Гексагональна архітектура [36] або як її ще називають архітектура портів та

адаптерів є однією з архітектур для розробки програмного забезпечення. Дана архітектура націлена на розробку слабозв'язних програмних компонентів системи які підключаються для спільної роботи за допомогою принципів портів та адаптерів. Це робить кожен компонент системи легкозамінним на будь-якому рівні системи та полегшує процес тестування. Однією з причин вибору даної архітектури була її сильна концептуальна схожість з поняттям агенту в мультиагентній теорії побудови програмних систем.

Агент як логічна сутність взаємодіє зі своїм середовищем за допомогою сенсорів, якими він прослуховує зміни в середовищі, та актуаторів, якими він впливає на середовище своїми діями. В результаті даної аналогії чітко видно обгрунтованість використання даного підходу.

На рис. 3.2. зображена візуалізація гексагональної архітектури

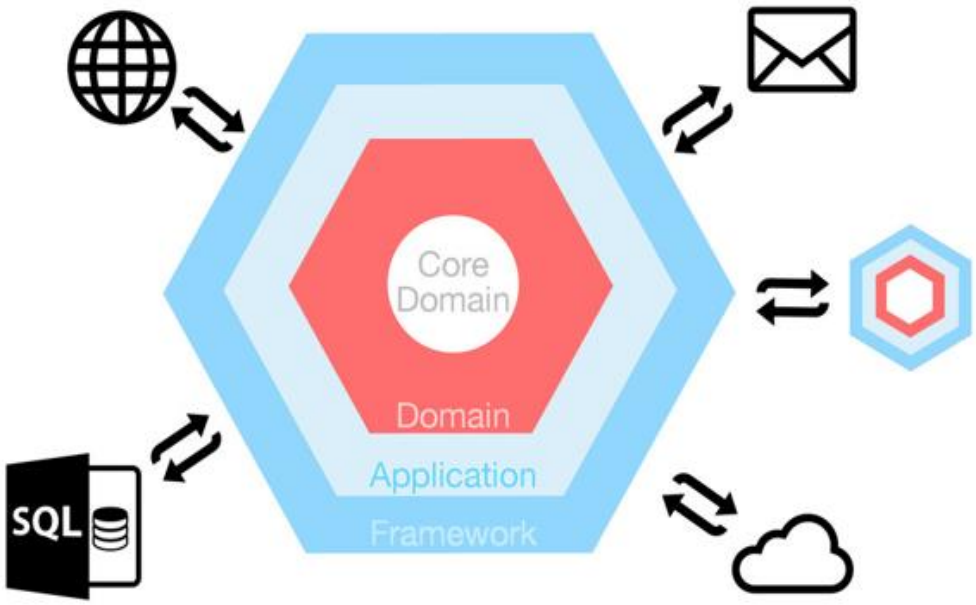


Рисунок 3.2 – Зображення гексагональної архітектури

Основною особливістю даної архітектури є незалежність логіки роботи додатку від конкретних технічних інструментів які використовуються. Наприклад:

– бізнес-логіка [37] для запису та отримання даних не залежить від конкретної реалізації бази даних.

– якщо додаток відправляє повідомлення користувачеві, то бізнес-логіка не знає яким чином це повідомлення буде доставлено, за допомогою пошти, чату в месенджері або безпосередньо через користувацький інтерфейс.

Популяризуючи базові принципи гексагональної архітектури та мультиагентного підходу в проектуванні систем можна представити абстрактну архітектуру агента представивши його основні компоненти у вигляді діаграми.

3.3.2 Визначення зв’язків та комунікації між агентами

На рис. 3.1. була представлена схема системи. В даній мультиагентній системі представлено 2 основних способу взаємодії між агентами. Способами взаємодії є:

– взаємодія «агент-агент»;
– взаємодія «агент-користувач».

Взаємодія «агент-агент» завжди проводиться через брокер асинхронної черги повідомлень. Цей тип взаємодії дозволяє агентам в системі обмінюватися повідомленнями в процесі своєї роботи. Нижче описані контракти взаємодії між кожною парою класів агентів:

– користувацький агент – кластерний агент;
– кластерний агент – обчислювальний агент;
– кластерний агент – міграційний агент.

Взаємодія «агент взаємодії з користувачем – кластерний агент». Комунікація між цими агентами абстрактно являє собою взаємодію користувача з системою. Користувацький агент після проведення аутентифікації користувача та валідації команди перенаправляє команду до кластерного агенту, який реагує на команди від імені системи в цілому.

Взаємодія «кластерний агент – обчислювальний агент». Взаємодія між даними агентами схожа за своєю суттю на взаємодію начальника з робітником. Після успішного ініціювання процесу обчислень кластерним агентом, він відсилає команди для початку обчислень всім обчислювальним агентам в кластері, при цьому передавши їм вхідні дані для обчислень (часовий ряд, необхідна точність обчислень)

та конфігурацію для кожного з них. В процесі проведення обчислень, обчислювальний агент котрий знайшов рішення з необхідною точністю передає результат обчислень кластерному агенту, який в свою чергу при отриманні позитивного результату ініціює завершення процесу обчислень для всіх агентів в кластері.

Взаємодія «кластерний агент – міграційний агент». Міграційний агент як і обчислювальний агент приймає участь в процесі обчислень тим, що керує процесом міграції хромосом між генофондами. Ініціювання процесу обчислень та завершення процесу обчислень також стосується даного класу агентів. Кластерний агент транслює команди початку та завершення обчислень для міграційного агенту кластеру.

Взаємодія «агент-користувач» проводиться між користувацьким агентом та користувачем за допомогою Telegram API. Як вже було зазначено, користувацький агент в мультиагентній системі є проміжковою ланкою для взаємодії користувача з системою. Метою даної взаємодії є аутентифікація користувачів які мають право на використання системи та інтерпретація користувацьких команд на зрозумілу для мультиагентної системи мову.

3.3.3 Визначення атомарної одиниці популяції (хромосоми)

Хромосома в генетичному алгоритмі є прототипом біологічної особи деякого виду. В методі обчислень за допомогою генетичного алгоритму хромосома являє собою модель деякого рішення поставленої задачі яку вирішує генетичний алгоритм. Операції над хромосомами:

- схрещування хромосом;
- мутація хромосом.

В процесі роботи генетичного алгоритму постійною операцією над хромосомами є операція схрещування. Дана операція проводиться між двома хромосомами по аналогії з ссавцями в живій природі, результатом операції схрещування є рівно дві хромосоми потомків, які певним чином наслідують гени обох батьків.

За аналогією з живою природою присутня також операція мутації над хромосомами. Дана операція передбачає випадкову мутацію деяких генів хромосоми і проводиться рівно над однією хромосомою. Необхідність даної операції пояснюється тим, що в процесі роботи алгоритму необхідно запровадити механізм який забезпечує постійну різноманітність генофонду, інакше всі хромосоми унаслідують всі гени і популяція виродиться.

3.3.4 Визначення популяції

Популяція як один з термінів генетичного алгоритму являє собою кінцевий набір хромосом з різними генами. В процесі проведення операцій над хромосомами таких як схрещування та мутація популяція поповнюється новими хромосомами, які наслідують певні гени та з деякою вірогідністю видозмінюється. В кінці епохи над популяцією проводиться операція селекції, метою якої є залишити в популяції тільки кращі хромосоми, які краще всього пристосовані для вирішення задачі закладної в генетичний алгоритм. За показник пристосованості хромосоми відповідає значення її фітнес функції, які розраховується для кожної хромосоми окремо.

3.3.5 Визначення роботи генетичного алгоритму

Генетичний алгоритм – це евристичний алгоритм, який використовується для вирішення задач оптимізації та моделювання шляхом випадкового перебору, комбінування та варіації параметрів на основі механізмів які є аналогічними природньому відбору в живій природі. Основними механізмами природнього відбору є схрещування, мутація, селекція та наслідування.

Як одне з нововведень було додано механізм катаклізму популяції. Даний механізм проводить майже повне винищення популяції при досягненні популяцією певного ступеня виродженості [38]. Суть роботи механізму аналогічна катаклізмам в живій природі. Як приклад з історії планети Земля можна взяти мел-палеогенове вимирання внаслідок якого вимерли майже всі динозаври. Як результат вимирання можна відмітити колосальну рекомбінацію генів флори та фауни на всій планеті та глобальне змінення вектору розвитку генофонду планети.

Робота генетичного алгоритму полягає в тому, щоб ітеративно проводити операції зміни генів над представниками популяції (хромосомами) відбираючи при цьому найпристосованіших представників на кожній ітерації, даний процес дозволяє знайти найбільш оптимальне рішення задачі в залежності від затрачених ресурсів. Процес обчислень в генетичному алгоритмі являється аналогією еволюції в живій природі.

3.3.6 UML-діаграма послідовності роботи системи

На рис. 3.3 зображено діаграму послідовності роботи системи.

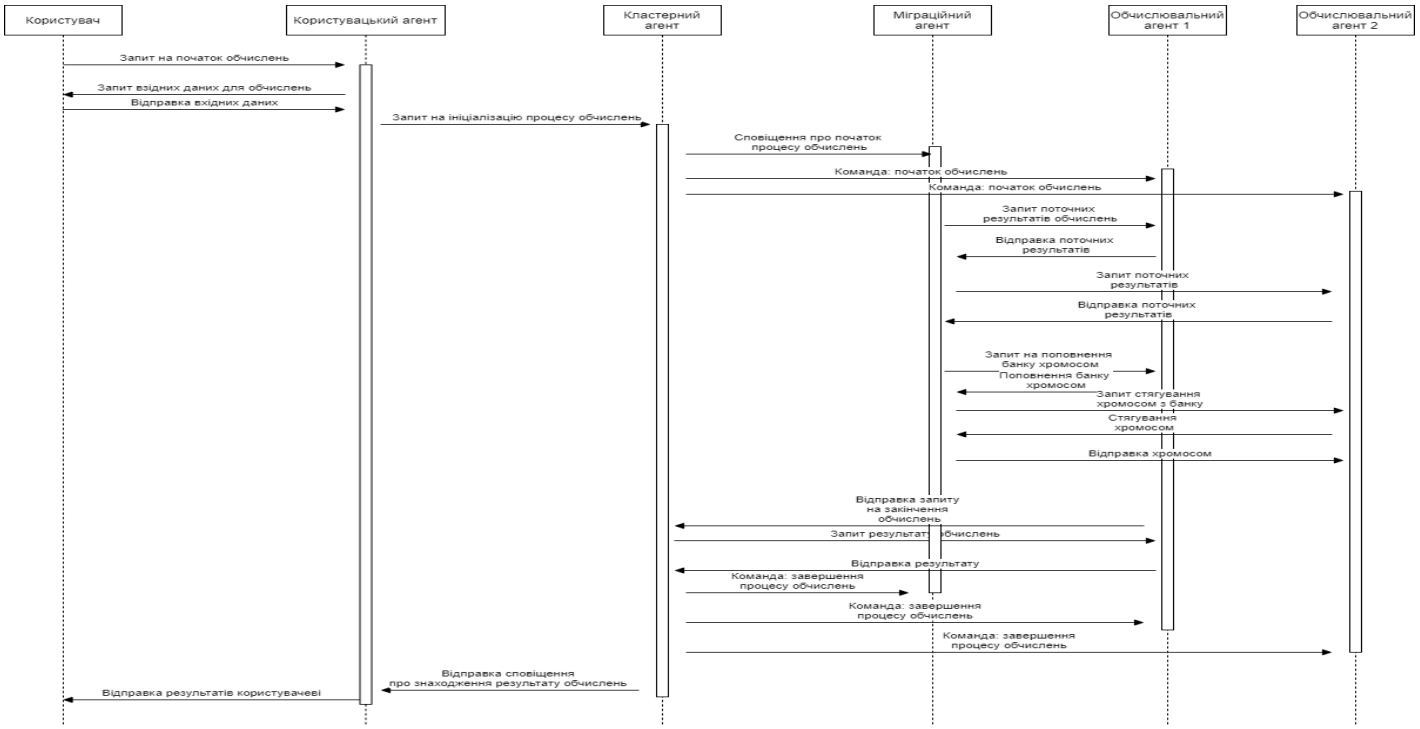


Рисунок 3.3 – Діаграма послідовності роботи мультиагентної системи

3.4 Реалізація базового функціоналу

3.4.1 Виявлення та підтримка контакту між агентами в процесі роботи

Концепція мультиагентного підходу при розробці систем передбачає взаємодію програмних агентів в межах середовища системи. Це означає що для взаємодії агентів між собою необхідно запровадити механізм спостереження за середовищем на предмет підключення або відключення з контексту системного нового агента в незалежності від його ролей в системі. Також необхідно розробити механізм підтримки контролю досяжності агентів між собою.

Ідея механізму полягає у відправленні повідомлень агентами між собою для виявлення стану активності агентів. Сценарій роботи механізму:

- при запуску агента в системі, він відправляє широкомовне повідомлення в середовище яке означає його присутність в системі;
- інші агенти які отримують це повідомлення відправляють повідомлення про свою активність в системі, при цьому вказуючи одержувача повідомлення нового агенту;
- після встановлення зв’язку між агентами кожен агент періодично відправляє повідомлення до кожного з агентів для підтримки досяжності агенту в системі;
- при самостійному відключенні з мультиагентної системи агент повинен відправити широкомовне повідомлення про свою відсутність, отримавши яке кожен агент видаляє даного агенту зі списку доступних агентів.

Ідея реалізації даного механізму не нова, концепція підтримки контакту між робочими агентами була запроваджена в більшість систем агентного типу. Один з таких інструментів – Kubernetes [39]. Kubernetes – це відкрите програмне забезпечення для оркестрування контейнеризованих додатків, а також автоматизації їх розгортування [40], масштабування, координації в межах кластеру. Дане програмне забезпечення вирішує більшість основних проблем при розробці складних систем які працюють на багатьох робочих машинах. Реальність така, що чим складніша система і чим більше робочих машин приймають участь у її роботі тим більша вірогідність відказу деяких вузлів цієї системи. Основні причини відказу робочої машини в процесі експлуатації:

- перебої з електроживленням;
- перебої з з’єднання з мережею Інтернет;
- критичні помилки операційної системи.

Вищеописаними проблемами не можна нехтувати якщо необхідно розробити ПЗ з доступністю 99.9999% и вище, адже такий відсоток безвідмовної роботи в рік це 52 хвилини даунтайму в рік.

3.4.2 Реалізація гексагональної архітектури моделі агентів

Основна ідея архітектури агенту як програмної сутності є перевикористання загальної функціональності в усіх типах агентів та слабка залежність комунікації від конкретних інструментів які забезпечують дану взаємодію. На рис. 3.4 представлено базову діаграму класів основних модулів агентів. Слід відмітити що обробка зовнішніх запитів до агентів було реалізовано за принципом подійно-орієнтованого програмування [41].

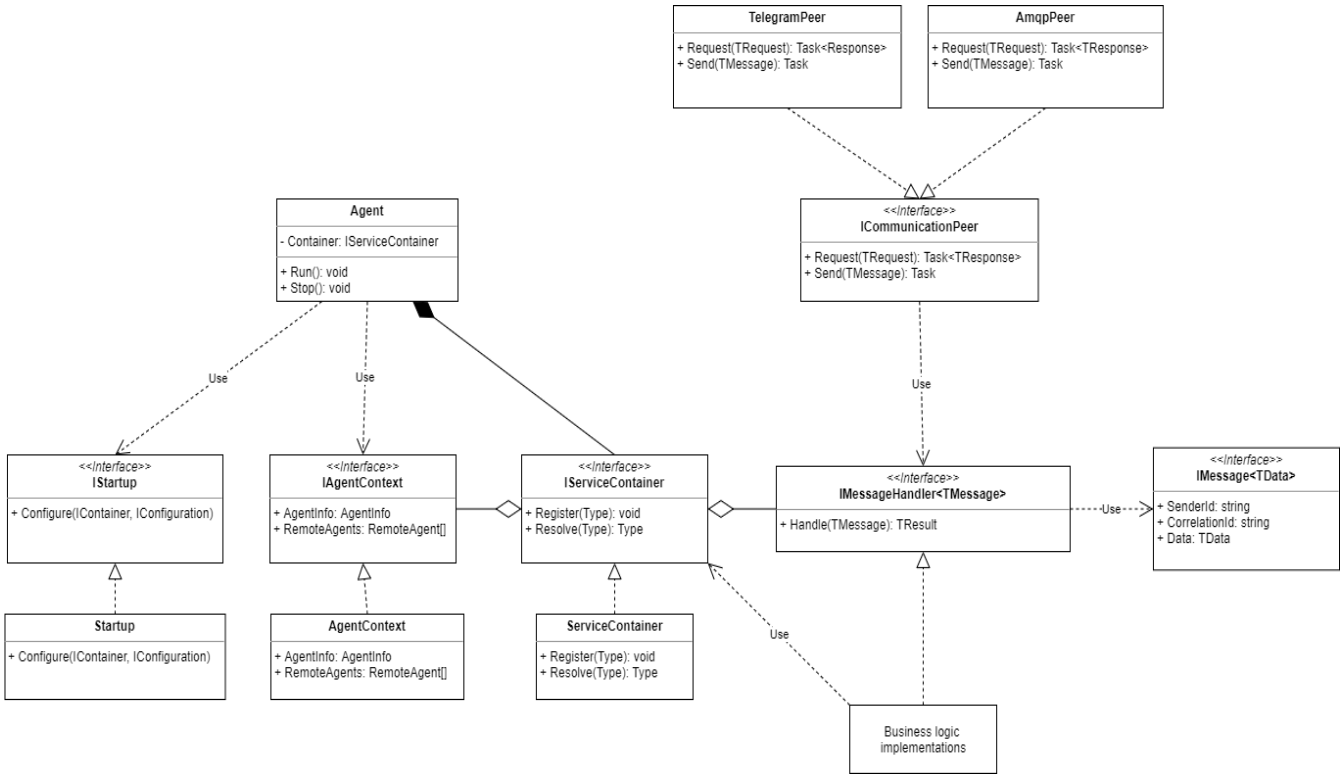


Рисунок 3.4 – Діаграма класів базової архітектури агентів

3.4.3 Використання принципу Dependency Injection при розробці агентів

Dependency Injection [42] – це процес представлення залежності в програмному компоненті. Являється специфічною формою «інверсії управління» [43], коли вона застосовується до керування залежностями. В повній відповідності принципу єдиної відповідальності [44] об’єкт віддає турботу про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього механізму.

Використання даного принципу в процесі розробки програмного забезпечення сильно спрощує процес розробки за рахунок впровадження слабкого

зчіплення [45] між модулями та сильної зв’язності [46] всередині них. Також використання всіх класів через інтерфейси додає гнучкості в архітектурі додатку.

3.4.4 Інтеграція користувацького агенту з платформою Telegram

Платформа Telegram дозволяє інтегрувати стороннє програмне забезпечення для виконання деяких дій спеціальними аккаунтами ботами [47]. Бот – спеціальне програмне забезпечення яке використовує Telegram API для своєї роботи. За допомогою ботів які розробляються як стороннє ПЗ користувачі мають змогу виконувати різні задачі, це може бути:

- пошук в інтернеті;
- обробка відео- та аудіо- файлів;
- планування задач;
- взаємодія з базами даних.

В контексті даної мультиагентної системи бот буде являти собою клієнта для взаємодії з користувачем. Керування ботом інкапсульоване в користувацького агента, який проводить аутентифікацію користувача, реагує на користувацькі команди, та сповіщує користувача про зміну стану системи, тощо.

Проектування взаємодії користувацького агента з користувачем проводилось з урахуванням таких умов:

- повідомлення від системи повинно бути в легкій для читання людиною формі;
- користувацькі команди для системи не повинні містити в собі складні для сприйняття людини дані;
- користувацькі сценарії повинні бути реалізовані в формі діалогів;
- необхідно передбачати затримку у відповіді користувача до 30 секунд.

Виходячи з вищеописаних вимог в структурі користувацького агенту було розроблено механізм діалогів. Діалог в даному контексті являє собою деякий сценарій запитів від користувача та до користувача який схематично можна представити у вигляді кінцевого автомата [48]. Результатом діалогу є деяка дія.

На рис. 3.5. зображено кінцевий автомат діалогу запуску процесу обчислень в системі.

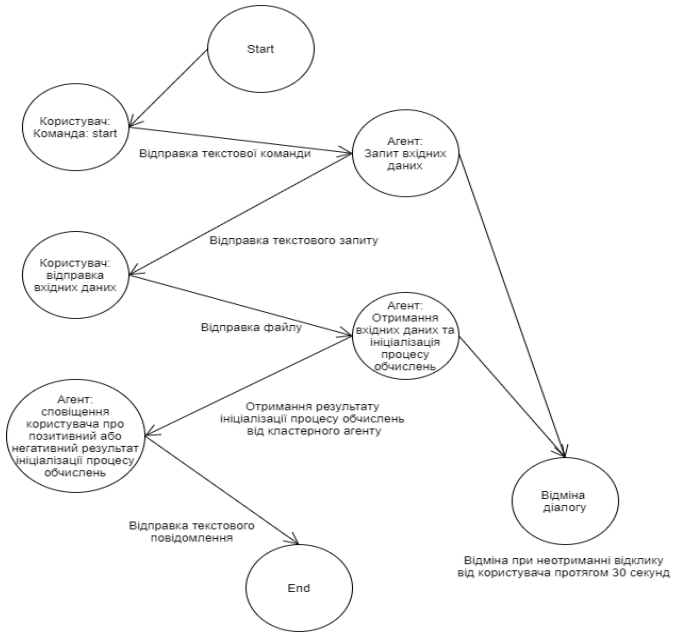


Рисунок 3.5 – Кінцевий автомат діалогу користувацького агенту з користувачем

Висновки до третього розділу

В даному розділі був описаний процес проектування та розробки інструментального ПЗ. Було описано процес проектування який складався з чотирьох етапів:

– визначення ролей агентів в мультиагентній системі;
– визначення архітектури агенту;
– визначення схеми роботи генетичного алгоритму;
– визначення головних сутностей з їх функціоналом та відповідальністю.

Базовою концепцію для побудови архітектури агенту було обрано гексагональну архітектуру, за допомогою якої можна ефективно знижувати зв’язність між модулями. Також було визначено схему роботи системи, ролі кожного з класів агентів, їх функціональність та відповідальність. Було визначено та формалізовано способи комунікації агентів в середовищі системи та способи комунікації користувача з системою.

4 ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ МЕТОДУ КОНСТРУКТИВНО-ПРОДУКЦІЙНОГО МОДЕЛЮВАННЯ

4.1 Етапи випробувань

Після завершення початкового етапу розробки в результаті якого було розроблено стабільну версію системи було розпочато етап тестування [49] програмної системи шляхом запуску тестування на заздалегідь підготовлених часових рядах. На початкових етапах тестування було протестовано ефективність роботи методу на рядах які мають детерміновану структуру та були породжені одним правилом довжиною до 8 символів. При досягненні стабільних результатів поступово тестування проводилось на складніших для відтворення часових рядах. Ускладнення експериментів та рядів проводилось за рахунок збільшення:

- кількості символів у правилах;
- кількості правил;

Загалом етапи випробувань можна поділити на наступні категорії:

- експерименти на часових рядах, які породжені за допомогою фрактальної моделі з одним правилом заміщення;
- експерименти на часових рядах, які породжені за допомогою фрактальної моделі з декількома правилами;
- експерименти на часових рядах які мають слабку стохастичну природу. Мова йде про часові ряди які описують відносно стабільні процеси, але факт їх замірювання являється стохастичним;
- експерименти на часових рядах які мають сильну стохастичну природу. Мова йде про часові ряди які описують реальні процеси в природі.

Основною метою даної роботи є збільшення часової ефективності методу використання конструктивно-продукційного моделювання для відтворення фрактальної моделі вхідного часового ряду для підвищення часової ефективності роботи даного методу за рахунок використання мультиагентного підходу.

4.2 Характеристики обладнання яке використовувалось для проведення експериментів

Перед проведенням експериментів безпосередньо необхідно визначити характеристики обладнання яке використовувалося для проведення випробувань. Ці дані необхідні для визначення так як відносні часові параметри роботи системи будуть залежати від конфігурації обладнання на якому випробовувалась система. В контексті цієї роботи та використаних практик цікавлять характеристики наступних складових:

– характеристики ЦП [50];
– характеристики з’єднання з мережею Інтернет.

4.2.1 Характеристики ЦП

Для визначення конфігурації центрального процесора було використано програмне забезпечення CPU-Z [51]. Використаний процесор для випробувань – Intel Core i5 8th Gen [52]. У табл. 4.1 було наведено список характеристик.

Таблиця 4.1 – Характеристики ЦП

Параметр ЦП	Значення
Базова частота	3.4 Ггц
Стічна частота процесора	1.3 Ггц
Кількість ядер	4
Кількість потоків	8
CPUID	6.8E.U0
Швидкість шини	~100 Мгц

У табл. 4.2. було наведено характеристики кешу.

Таблиця 4.2 – Характеристики кешу

Рівень кеш-пам’яті	Значення
L1 Data	4 x 32 Kbytes
L1 Inst.	4 x 32 Kbytes
L2	4 x 256 Kbytes
L3	3 Mbytes

4.2.2 Характеристики з’єднання з мережею Інтернет

Для визначення характеристик з’єднання з мережею інтернет було використано ресурс speedtest.net [53] . Результати було наведено в таблиці 4.3.

Таблиця 4.3 – Характеристики з’єднання з мережею Інтернет

Характеристика	Значення
Download speed	55.3 Mbps
Upload speed	55.6 Mbps
Ping	3 ms

Також слід зазначити характеристики оперативної пам’яті встановленої на робочій машині. На ЕОМ було встановлено 12 Гб оперативної пам’яті DDR4 [54].

4.3 Випробування на детермінованих та стохастичних часових рядах

В даному розділі випробування діляться на дві окремі групи. Це випробування на часових рядах детермінованої природи та випробування на часових рядах зі стохастичним [56] компонентом. Розділення випробувань на ці дві групи обумовлюється деякими змінами в підходу до відтворення фрактальної конструктивної моделі ряду стохастичної природи відносно до відтворення детермінованих рядів.

Детерміновані ряди як математична модель є повністю однозначними і не включають жодної стохастичної компоненти в своїй природі, такі ряди однозначно задаються конструктивною моделлю яка включає:

– L-систему;
– математичне очікування;
– відхилення математичного очікування.

При побудові детермінованих рядів, дисперсія математичного очікування не враховується і рівна нулю.

Порівняно з детермінованими часовими рядами стохастичні часові ряди мають дисперсію математичного очікування в своїй природі і не можуть бути однозначно реконструйовані опираючись тільки на математичне очікування та

відхилення математичного очікування. Для реконструювання стохастичних рядів структура хромосоми розширюється внаслідок додавання ще одного гена в її структуру – дисперсія математичного очікування.

4.3.1 Випробування на детермінованих рядах породжених одним та множиною правил

Випробування на даному етапі характеризуються дослідженням ефективності відновлення фрактальної моделі за заданим детермінованим рядом. Детерміновані ряди для цього етапу випробувань були сконструйовані за допомогою конструктора на основі L-систем.

Слідує відмітити що стохастичний компонент в детермінованих рядах відсутній, тому дисперсія для побудови цих рядів не враховується.

Так як система проектувалася та розроблялася для досягнення ефективного масштабування обчислювальних ресурсів, що в свою чергу збільшує часову ефективність методу конструктивно-продукційного моделювання, випробування проводились з різної кількістю активних обчислювальних агентів для дослідження динаміки зміни характеристик часової ефективності роботи системи.

Після проведення деякої кількості випробувань було визначено наступні проблеми в роботі генетичного алгоритму, побудованого за схемою попередніх досліджень в даному напрямку:

- висока домінація певних генотипів в популяції;
- застрягання в локальних мінімумах;
- обмеженість процесу рекомбінації генів.

Швидкість виродження популяції дуже висока через замалий коефіцієнт мутації в процесі роботи алгоритму. Слідує відмітити, що висока чисельність популяції, що складає 1500 особин, не тільки не вирішує дану проблему, але й посилює її.

В процесі роботи генетичного алгоритму виявилась дана проблема яка є типовою для сімейства алгоритмів еволюційних обчислень. Знайдення більш оптимального рішення при застряганні в локальному мінімумі являється дуже

складною ресурсномісткою задачею, що негативно впливає на часову ефективність роботи методу.

Дана проблема являє собою обмеженість способів рекомбінації генів в фазах схрещування та мутації. Обмеженість рекомбінації звужує область пошуку оптимального рішення задачі, що ускладнює процес обчислень та негативно впливає на часову ефективність.

Вищеописані проблеми всі в тій чи іншій мірі є типовими проблемами в еволюційних алгоритмах, але при цьому однозначного рішення для них немає виходячи з особливостей постановки певної задачі яку вирішує алгоритм.

В контексті даної роботи було розроблено наступні покращення та оптимізаційні механізми для роботи генетичного алгоритму:

– зменшення кількості хромосом в популяції;
– механізм катаклізму популяції;
– розширення способів рекомбінації генів в фазах мутації та схрещування;
– видалення клонів з популяції;
– перевага фази мутації над фазою схрещування.

Виходячи з задачі основним геном який складно підбирається в хромосомі являється L-система. Навіть зміна одного символу в одному з правил може кардинально змінити структуру конструктивного ряду побудованого за даною L-системою. Домінація певних L-систем в популяції являється дуже акцентованою виходячи з близькості сконструйованого за ними часового ряду до вхідного. Цей фактор дозволяє визначити що використання великих за кількістю хромосом популяцій негативно впливає на час обчислень і не привносить в процес обчислень значущих характеристик ефективності;

Механізм є нововведенням в роботі генетичного алгоритму за допомогою якого досягається високий ступінь різноманітності генів в контексті популяції. Робота механізму полягає в винищенні левової частки популяції при досягненні популяцією певного рівня виродження та заповнення популяції новими хромосомами які були створені випадковим чином. Рівень виродження в даному випадку розуміється як відсоток схожості генів в популяції та середній вік

хромосом. Експериментально доведено, що використання даного методу в генетичному алгоритмі знижує вірогідність попадання алгоритму в локальний мінімум та полегшує знаходження більш оптимального рішення при попаданні в цей локальний мінімум;

За рахунок використання розподіленого генетичного алгоритму який проектувався та розроблявся в контексті мультиагентного підходу, доцільно використовувати різні стратегії в фазах схрещування та мутації на кожному з обчислювальних агентів. Різна обробка хромосом в цих фазах розширює область пошуку оптимального рішення. Крім того було визначено експериментально, що комбінування деяких стратегій при одночасному їх використанні показують високу ефективність обчислень. Детальний опис даних механізмів буде описано нижче;

В процесі проведення випробувань нерідко вникає ситуація коли при схрещуванні чи випадкової мутації деяких хромосом з’являються клони. Використання хромосом-клонів в контексті однієї популяції є недоцільним так, як це тільки зменшує різноманітність генофонду, що в свою чергу негативно впливає на часову ефективність роботи системи. Було прийняте рішення видаляти хромосом-клонів з популяції;

Особливість задачі відновлення конструктивної моделі вхідного часового ряду полягає в тому, що сама L-система виступає базовим основним компонентом котрий займає ключову роль в реконструюванні часового ряду. Мутації забезпечують популяцію більшим ступенем різноманітності генів, яка являється ключовим показником для знаходження потенціального рішення. В свою чергу схрещування в контексті даної задачі реконструювання не грає таку важливу роль як мутації через надмірність в реплікації генів, які як правило несумісні між собою. Тому для забезпечення більшої ефективності обчислень прийнято рішення віддати перевагу процесу мутації перед схрещуванням.

Слідує відмітити, що в даній розробці генетичного алгоритму схрещування присутнє, як таке, але вірогідність проведення даної операції над хромосомами знижено. Зниження вірогідності схрещування хромосом позитивно вплинуло на часову ефективність процесу обчислень.

Виходячи зі змін в роботі алгоритму, хотілося б детальніше зупинитися на варіативності проведення фаз мутації та схрещування хромосом. Доцільність застосування варіативності при мутаціях та схрещуванні пояснюється використанням принципу розподіленого генетичного алгоритму на базі мультиагентного підходу. Розробленими механізмами схрещування є:

– комбінаторне розподілення генів.;
– рандомізація рекомбінації генів;
– випадкова рекомбінація генів.

Суть комбінаторного розподілення генів полягає в тому що як і в усіх інших стратегіях, в процесі схрещування приймають участь дві хромосоми, при цьому ця пара хромосом виробляє 2^n нових хромосом, де n –величина набору генів кожної хромосоми;

Рандомізація рекомбінації генів полягає в наслідуванні випадково вибраного гену від однієї хромосоми та всього набору крім даного гену від другої. При цьому виробляється інверсійна хромосома до даної. В результаті схрещування виходить рівно дві нових хромосоми;

Випадкова рекомбінація генів полягає в випадковому виборі типів генів та джерел цих генів і створення хромосом на основі даної конфігурації. При цьому створюється хромосома за прямою та інваріантна версією конфігурації.

Розробленими механізмами мутації є:

– мутація на основі наслідування властивостей L-системи хромосоми;
– випадкова мутація L-системи;
– зміна порядку слідування символів в правилах заміщення L-системи.

Мутація на основі наслідування властивостей L-системи хромосоми являє собою мутацію L-системи зі збереженням її властивостей. Характеристиками за якими оцінюється L-система в даному контексті:

– кількість правил;
– середня кількість символів в кожному правилі;
– дисперсія кількості символів у правилах;
– довжина аксіоми.

Вищезазначені характеристики наслідуються і на основі них генерується нова L-система. Даний метод дозволяє оптимізувати напрям пошуку рішення задачі в контексті об’єму та складності L-системи.

Випадкова мутація L-системи. В результаті роботи за стратегією випадкової мутації L-системи створюється нова L-система згідно з обмеженнями які були задані обчислювальному агенту. Особливість даного механізму полягає в забезпеченні великого ступеню варіативності пошуку оптимальних рішень.

Зміна порядку слідування символів в правилах заміщення L-системи полягає в зміні порядку слідування символів в правих частинах правил заміщення. Не складно визначити, що при такому підході структура L-системи не змінюється, при цьому змінюється конструктивна модель за якої будується часовий ряд. Це дозволяє більш точно мутувати кожну з L-систем в граничних випадках коли для досягнення потрібного результату необхідно провести незначні зміни в формі L-системи. Даний метод позитивно впливає на часову ефективність системи в цілому.

На рис. 4.1 – 4.4 наведено приклади реконструйованих детермінованих часових рядів різної складності. Всі ряди з цього етапу випробувань є детермінованими, що виключає компоненту стохастичності в їх природі. Кожний з таких рядів має нульову дисперсію своїх значень.

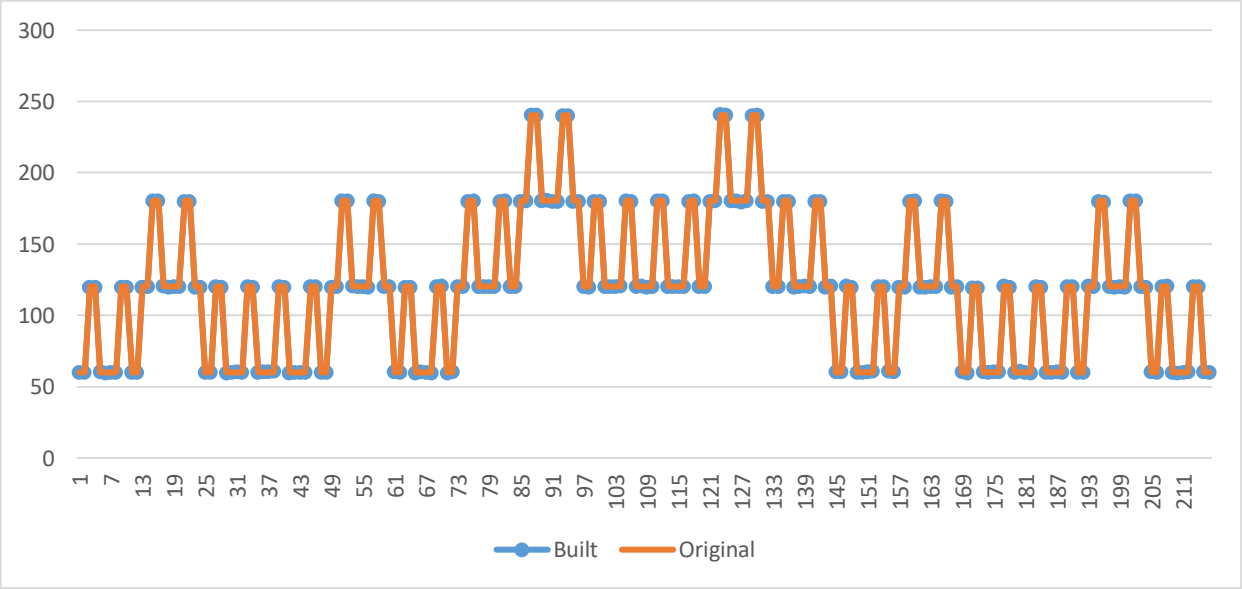


Рисунок 4.1 – Порівняння вхідного та реконструйованого часових рядів для конструктивної моделі: $f \to f + f - f + f - f$, M=10, dM=1

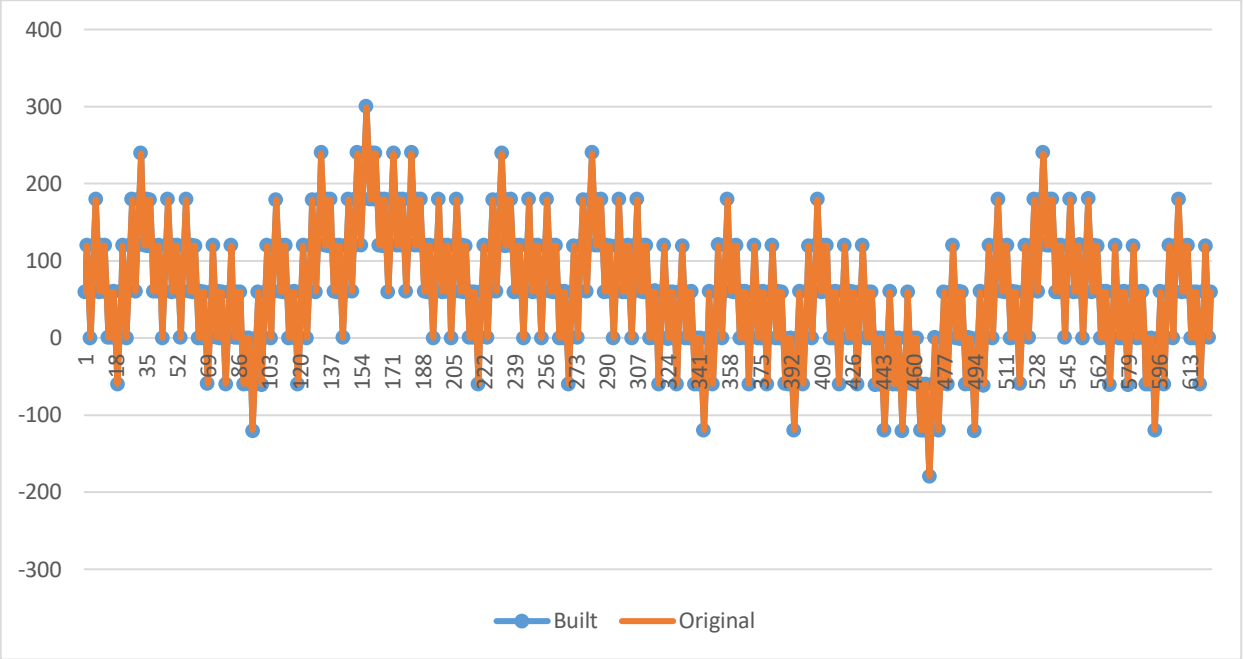


Рисунок 4.2 – Порівняння вхідного та реконструйованого часових рядів для конструктивної моделі: $f \to f - f + + f - f$, M=10, dM=5

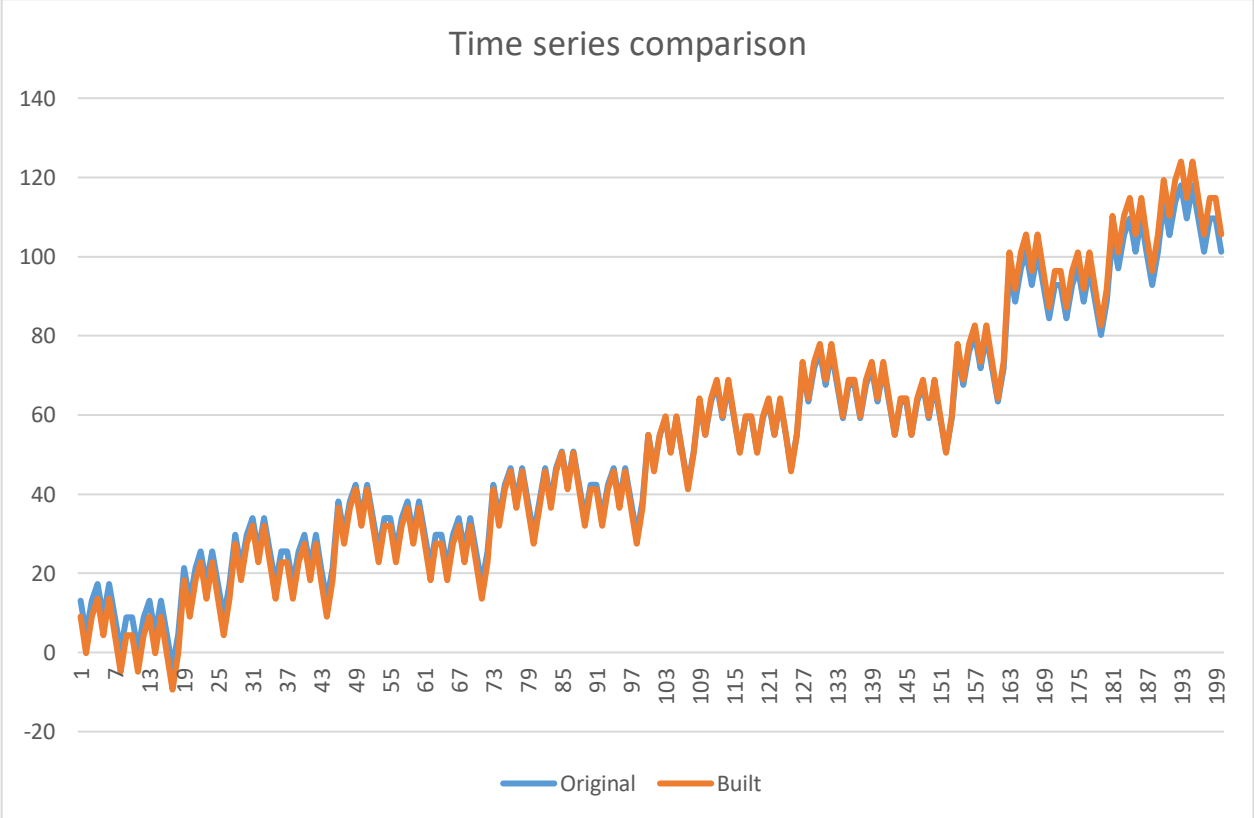


Рисунок 4.3 – Порівняння вхідного та реконструйованого часових рядів для конструктивної моделі: $f \to g + + g - - g, g \to f - f + + f - -$, M=9, dM=-4,6

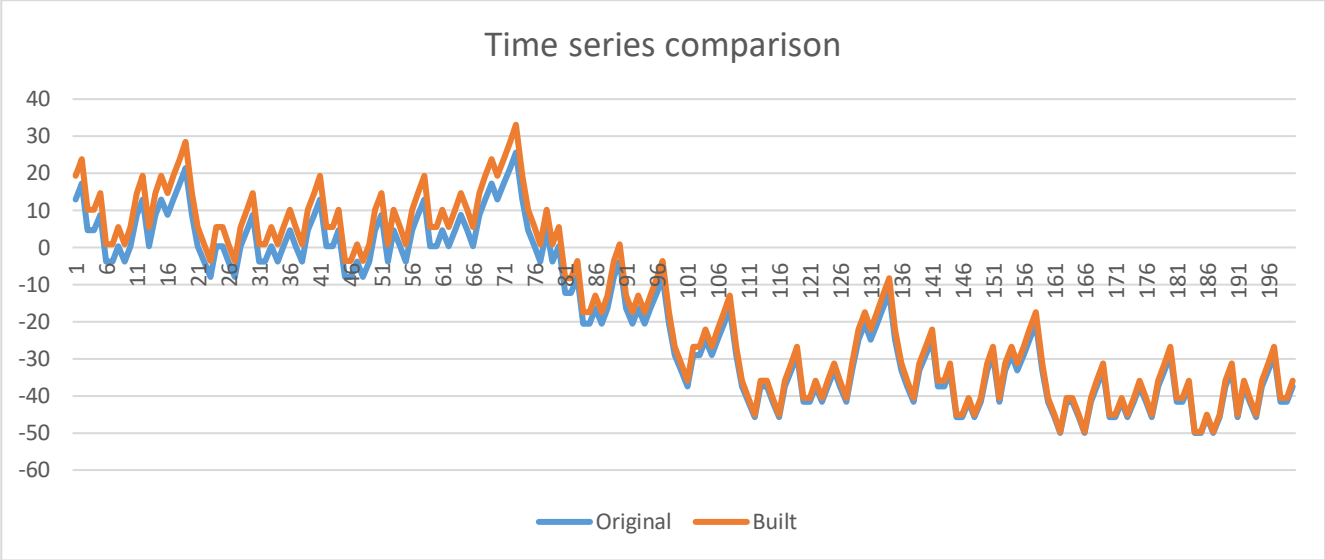


Рисунок 4.4 – Порівняння вхідного та реконструйованого часових рядів для конструктивної моделі: $f \to g+g+h--g, g \to h-g+++f-, h \to f-g+h-f$, M=19.3, dM=-3,93.

В результаті проведення даного випробування було реконструйовано детерміновані ряди різного рівня складності. Рівнем складності в даному випадку виступає об’єм фрактальної конструктивної моделі ряду, а саме такі її показники як:

– середня кількість символів в правій частині правила заміщення;
– кількість правил заміщення.

При проведенні даних випробувань були використані конструктивні моделі, L-система яких включає від одного до трьох правил заміщення. Першим на що варто звернути увагу це залежність швидкості проведення ітерацій від кількості значень вхідного часового ряду. В результаті проведення випробувань було виявлено пряму залежність від цих показників. Також, було виявлено, що при формуванні складних L-систем, які включають велику кількість правил та символів необхідно, щоб нова L-система відповідала критерію репродуктивності, тобто повинна бути можливість побудови фрактального мультисимвольного ланцюжка з необмеженим розміром. Виходячи з цього в процесі генерації фрактальної структури необхідно валідувати L-систему на предмет побудови необмежених за розміром конструкцій. Для цих потреб було розроблено валідатор для верифікації сформованої L-системи.

Результати випробувань на детермінованих часових рядах рядах наведені в табл. 4.4.

Таблиця 4.4 – Результати випробувань на детермінованих рядах

Кількість правил в L-системі	Середня кількість символів у правилах	Кількість значень ряду	Середня кількість ітерацій генетичного алгоритму	Середній час затрачений на випробування (с)	Швидкість ітерацій (ітерацій/с)
1	8	625	290	96.6	3
1	8	211	312	33.9	9.2
2	7.5	624	1301	481	2.7
2	7.5	212	1287	146.2	8.8
3	7	620	6923	2884.5	2.4
3	7	210	6842	823.3	8.3

Слід відмітити, що при збільшенні кількості правил в конструктивній моделі процес конструювання фрактальної моделі також ускладнюється в межах 5%. Кількість обчислювальних агентів які приймали участь у процесі обчислень рівна трьом.

Наступним етапом аналізу отриманих результатів є залежність часової ефективності методу відтворення конструктивної моделі вхідного часового ряду за допомогою конструктивно-продукційного моделювання від кількості активних обчислювальних агентів в системі які проводять обчислення спільно. Проведення випробувань в таких умовах дозволить ефективно проаналізувати підвищення часової ефективності роботи методу конструктивно-продукційного моделювання для часових рядів за рахунок горизонтального масштабування обчислювальних ресурсів, а саме підключення до процесу обчислень більшї кількості активних агентів, які будуть приймати участь в обчисленнях паралельно. Для проведення випробувань було використано від одного до трьох активних обчислювальних агентів. В табл. 4.5 наведені результати випробувань відтворення конструктивної моделі з різною кількістю активних обчислювальних агентів.

Таблиця 4.5 – Результати випробувань для різної кількості активних агентів

Кількість активних обчислювальних агентів	Кількість правил в L-системі	Середня кількість символів у правилах	Середня кількість ітерацій	Середній час затрачений на випробування (с)	Середня швидкість ітерацій (ітерацій/сек)
1	1	8	762	83.7	9.1
1	2	7.5	3954	449.3	8.8
1	3	7	31831	3789.4	8.4
2	1	8	538	57.8	9.3
2	2	7.5	2373	275	8.6
2	3	7	12632	1540.4	8.2
3	1	8	312	33.9	9.2
3	2	7.5	1287	146.2	8.8
3	3	7	6842	823.3	8.3

Як видно з результатів, масштабування обчислювальних ресурсів за рахунок використання розподіленого генетичного алгоритму в контексті мультиагентної системи сильно збільшує ефективність методу відновлення конструктивної моделі вхідного часового ряду. Такий результат пояснюється не тільки за рахунок проведення паралельних обчислень множиною агентів, але й за рахунок проведення міграції генофондів між окремими генетичними алгоритмами.

Генетичний алгоритм є еволюційним методом обчислення. Дане сімейство методів обчислення характеризується своєю стохастичною природою при проведенні процесу обчислень. Це означає, що заздалегідь не можна знати точну кількість ітерацій та час витрачений на процес обчислень. Для ефективної оцінки результатів обчислень отриманих з використанням еволюційних обчислень слід проводити достатньо велику кількість випробувань. Прицьому чим більше випробувань було проведено, тим точніше оцінку можна отримати. Нижче наведені (рис. 4.5 – 4.7) порівняння середньої кількості ітерацій до максимальної кількості ітерацій в залежності від складності конструктивної моделі детермінованого ряду та кількості активних обчислювальних агентів.

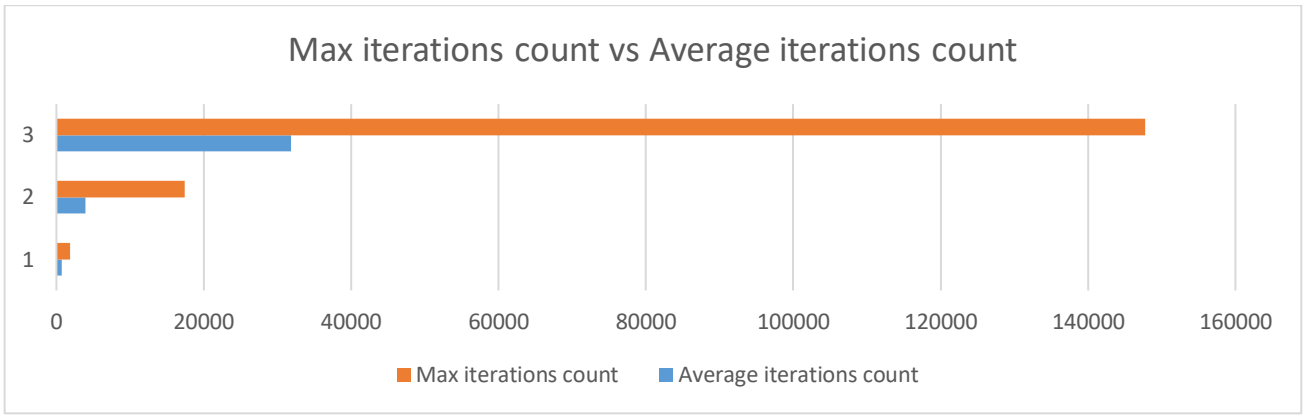


Рисунок 4.5 – Порівняння середньої кількості ітерацій з максимальною кількістю ітерацій для обчислень з одним активним обчислювальним агентом

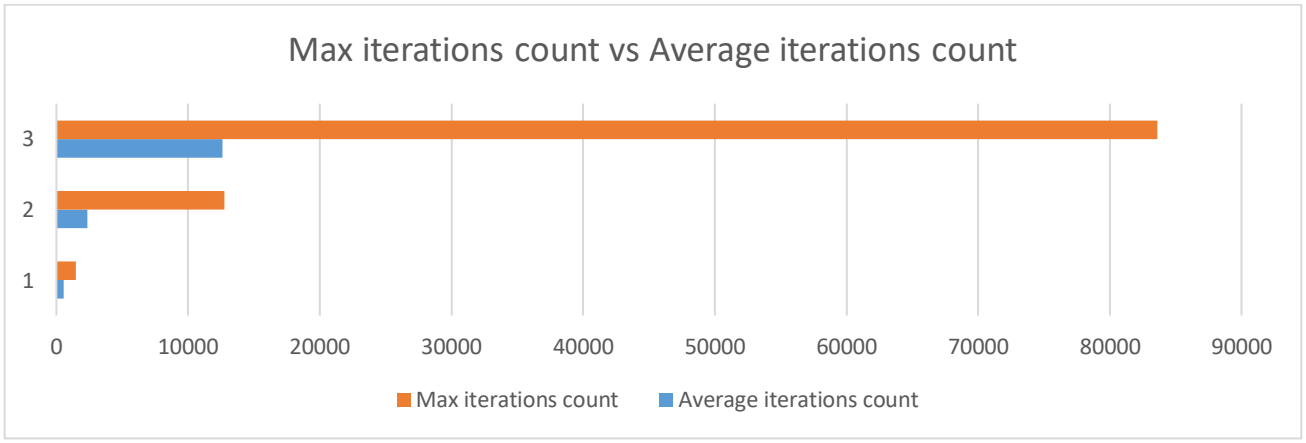


Рисунок 4.6 – Порівняння середньої кількості ітерацій з максимальною кількістю ітерацій для обчислень з двома активними обчислювальними агентами

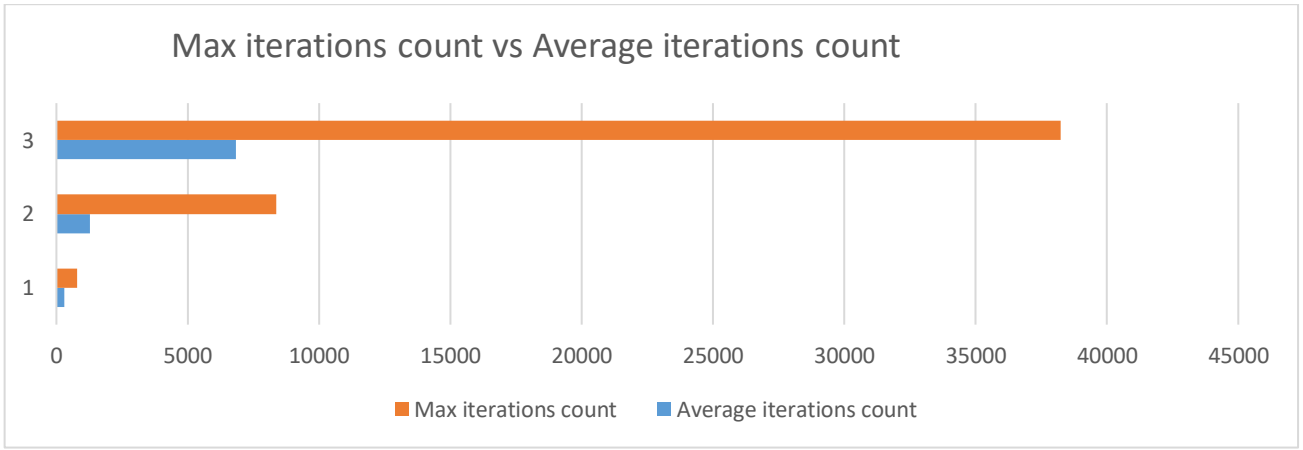


Рисунок 4.7 – Порівняння середньої кількості ітерацій з максимальною кількістю ітерацій для обчислень з двома активними обчислювальними агентами

Як видно, при збільшенні кількості обчислювальних агентів зменшується середня та максимальна кількість ітерацій генетичного алгоритму за рахунок його розподілення. Але при цьому експоненціальна тенденція збільшення складності відтворення конструктивної моделі відносно об’єму L-системи залишається сталою.

Наступним аспектом досліджень даного етапу є дослідження ефективності роботи нового розробленого механізму катаклізму популяції для генетичного алгоритму. Даний механізм був розроблений для мінімізації вірогідності попадання та тривалого перебування оптимального рішення в околицях локального мінімуму [55]. Робота даного механізму нагадує наслідки природніх глобальних катаклізмів над популяціями живих організмів в природі. Прикладом глобального катаклізму з історії планети є вимирання динозаврів. Дана подія призвела до повної рекомбінації в структурі та формі генів більшості представників популяції. В результаті цього домінуючі види змінились що спричинило зміну вектору популяції. Даний процес дозволяє ефективно вибиратися з локальної області рішення для знаходження більш оптимального. На рис. 4.8 зображено типовий графік середнього значення фітнес-функції в процесі проведення обчислень.

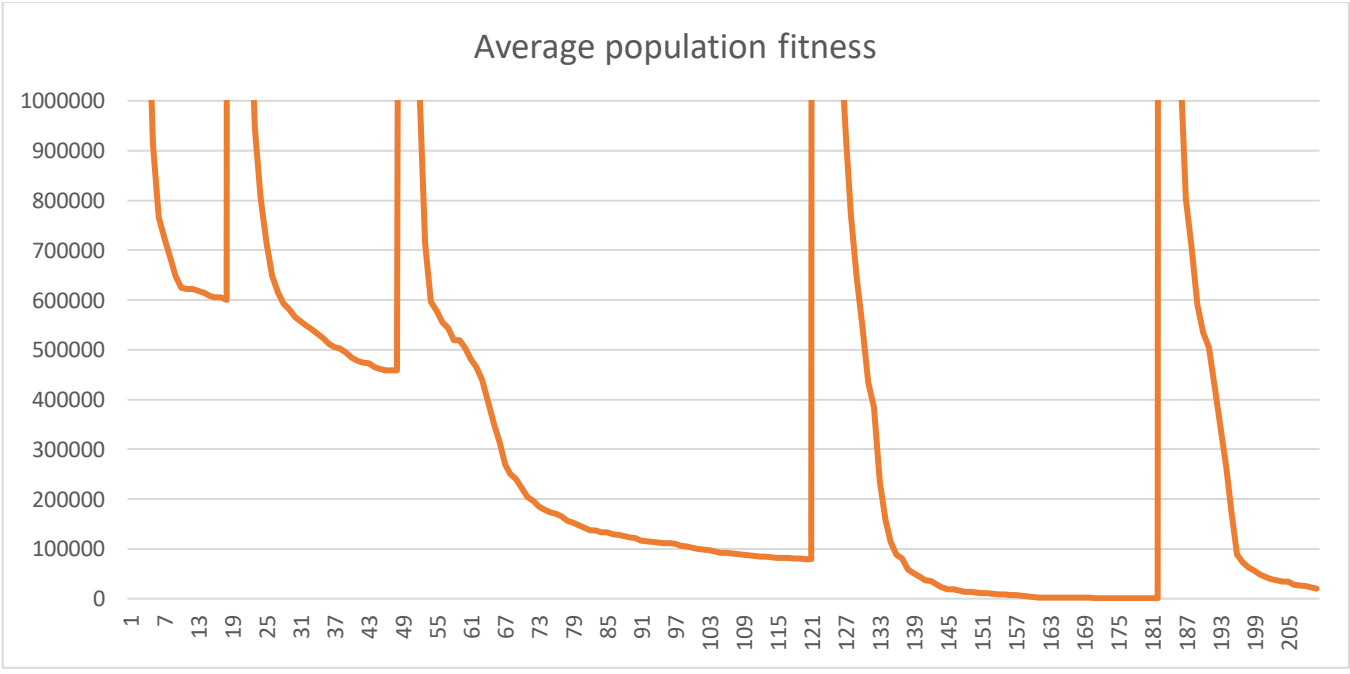


Рисунок 4.8 – Середнє значення фітнес-функції в процесі роботи генетичного алгоритму

На діаграмі зображеній вище видно, що при проведенні катаклізму популяції (різке зростання значень на графіку) після відновлення стану популяції до нормального спостерігається різке зниження середніх значень фітнес-функції в популяції що означає знаходження більш оптимального рішення. Таким чином можна зробити висновок що дане нововведення в роботі генетичного алгоритму є успішним інструментом для оптимізації методу відтворення конструктивної моделі вхідного часового ряду.

4.3.2 Випробування на стохастичних рядах

Випробування на даному етапі характеризуються дослідженням ефективності відновлення наближеної конструктивної моделі для стохастичних рядів та їх прогнозування. Відновлення конструктивної моделі для стохастичних часових рядів характеризується ступенем приближення реального вхідного часового ряду до сконструйованого за фрактальною конструктивною моделлю. Варто розуміти, що реальні стохастичні процеси не можуть бути реконструйовані з дуже високою точністю через їх випадкову природу.

Ступінь стохастичності будь-якого природнього процесу включає масу факторів випадковості в процесі спостереження за ним. Будь-який пристрій, який реєструє деякі показники процесу завжди має свою власну похибку вимірювань, кожен показник, в свою чергу, може залежати від деяких невідомих подій або умов. Всі ці фактори посилюють стохастичність спостережуваного процесу, що ускладнює його формалізацію, аналіз та прогнозування.

Перш за все, необхідно розібратись з поняттям стохастичного ряду, в контексті даної роботи. В процесі випробувань були застосовані детерміновані часові ряди з ненульовою дисперсією значень ряду та реальні стохастичні часові ряди. Детермінований часовий ряд з ненульовою дисперсією значень являє собою часовий ряд сконструйований за конкретною фрактальною конструктивною моделлю, але при цьому математичне очікування значень ряду має певну дисперсію, яка спотворює структуру ряду прямо пропорційно до величини свого модулю. На

рис. 4.9 зображено канонічний детермінований ряд сконструйований за допомогою фрактальної моделі який має нульову дисперсію.

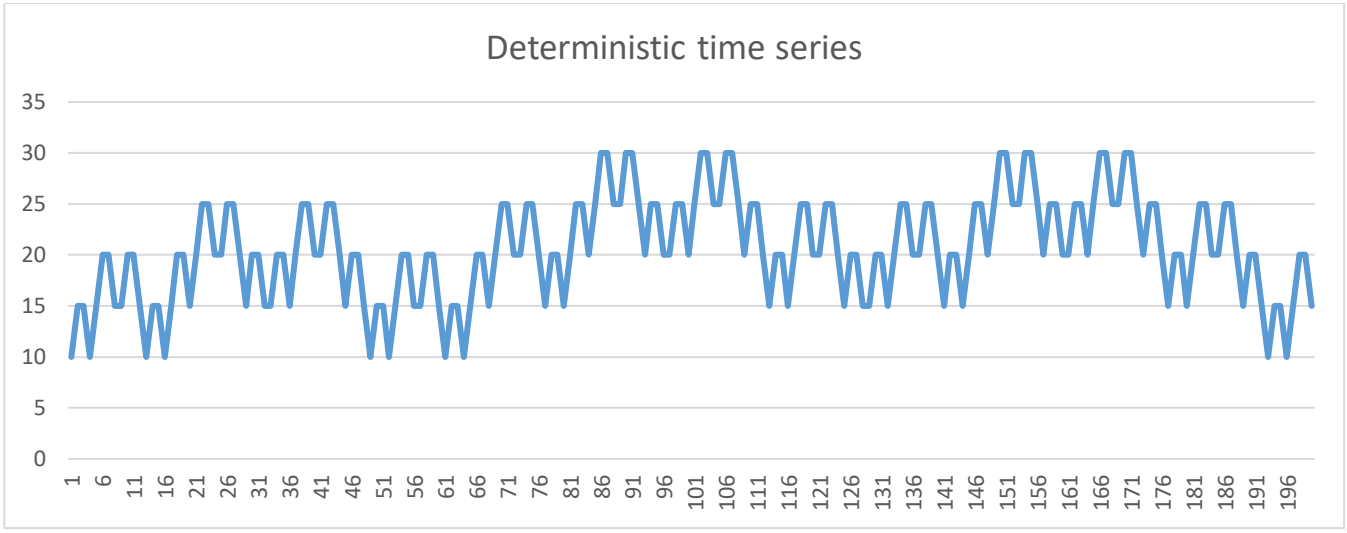


Рисунок 4.9 – Детермінований часовий ряд для конструктивної моделі: $f \to f + ff - f$, $M = 10, dM = 5, \sigma = 0$

Як видно на графіку структура ряду має повністю детермінований характер, це означає, що кожне наступне значення ряду може бути чітко спрогнозоване маючи конструктивну модель цього ряду. На рис. (4.10 – 4.12) зображено часові ряди побудовані за цією самою конструктивною моделлю, але мають різну величину дисперсії.

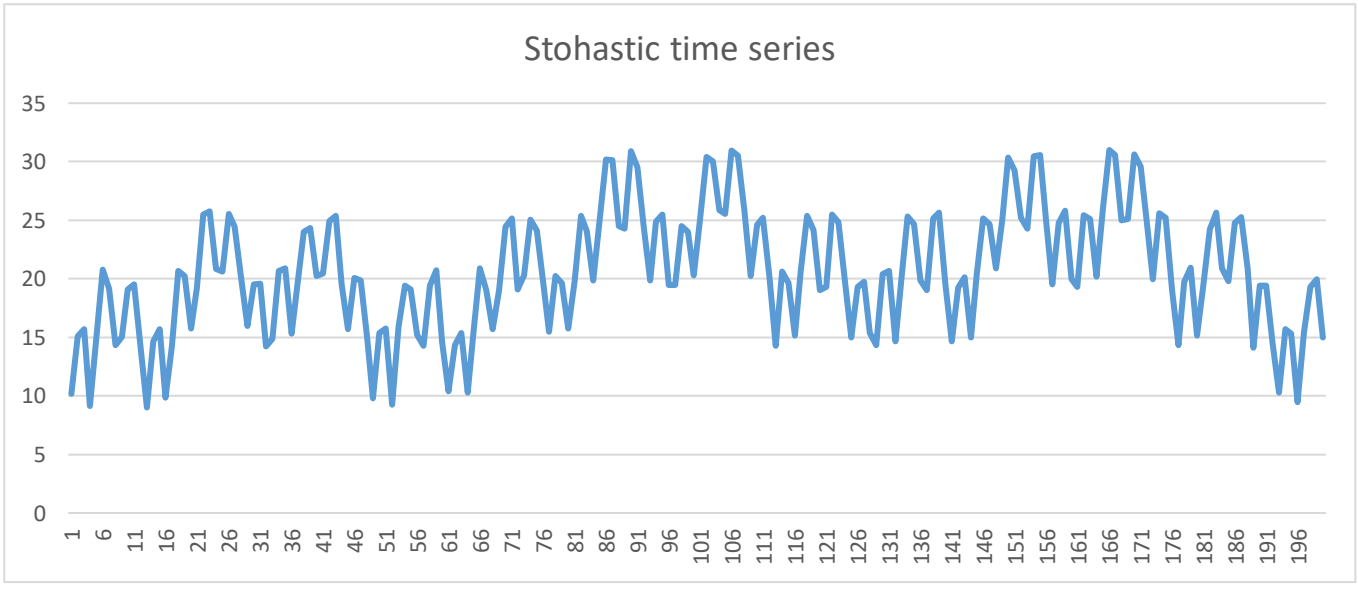


Рисунок 4.10 – Недетермінований часовий ряд для конструктивної моделі:

$f \to f + ff - f$, $M = 10, dM = 5, \sigma = 1$

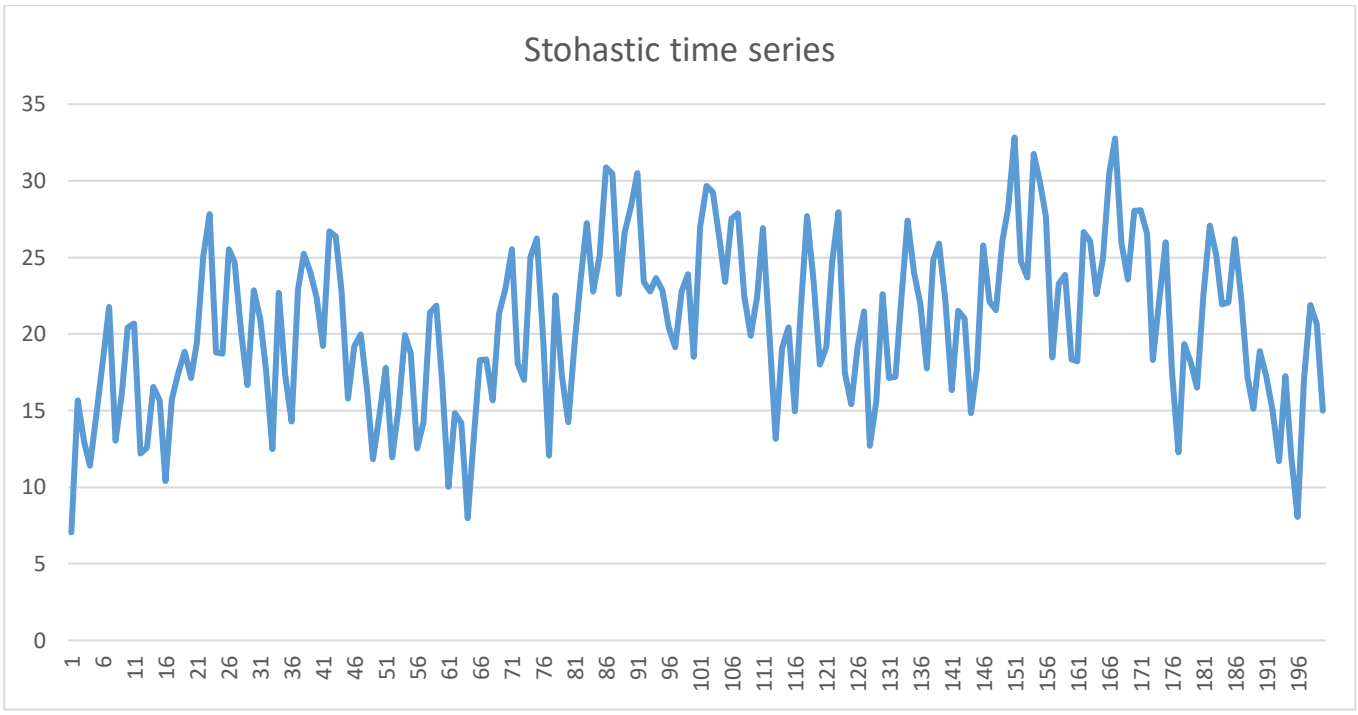


Рисунок 4.11 – Недетермінований часовий ряд для конструктивної моделі:

$f \to f + ff - f$, $M = 10, dM = 5, \sigma = 3$

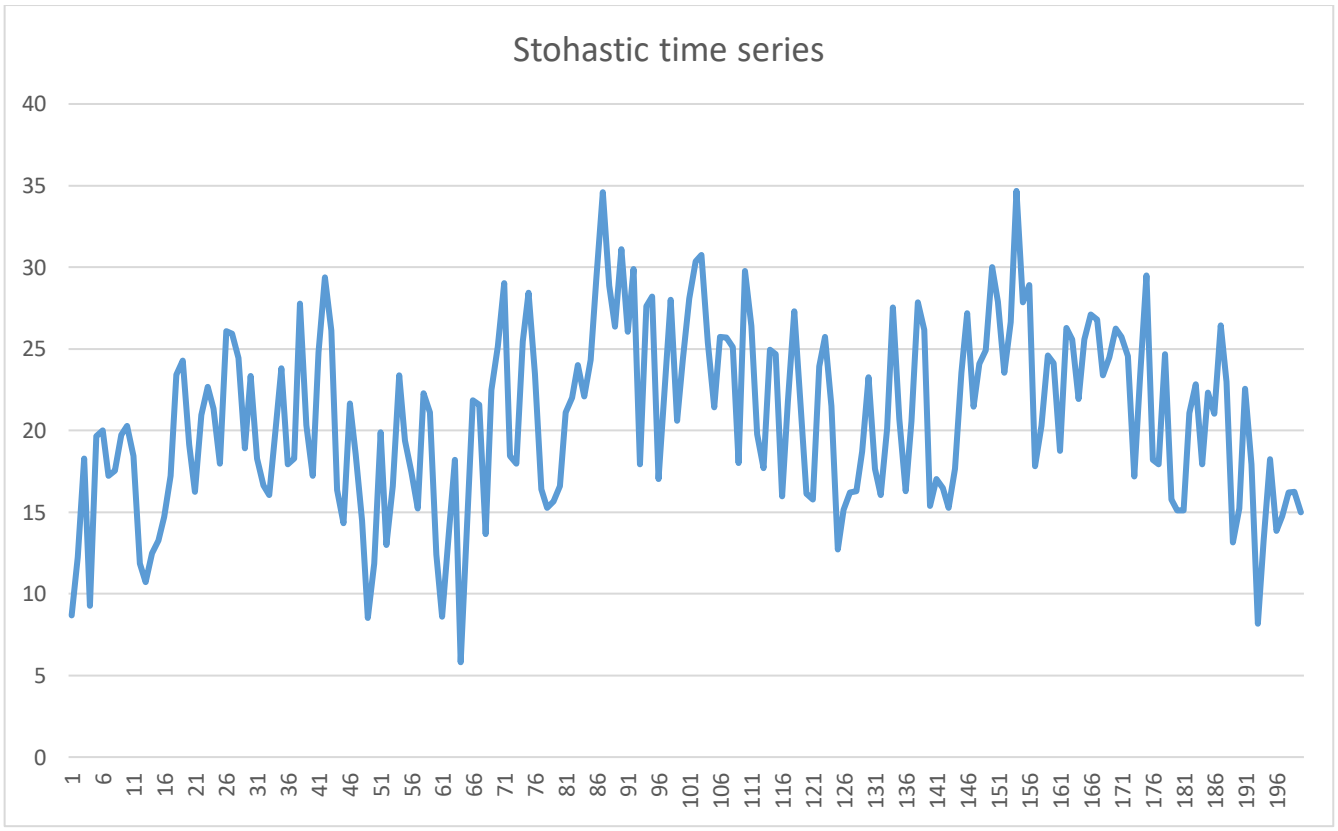


Рисунок 4.12 – Недетермінований часовий ряд для конструктивної моделі:

$f \to f + ff - f$, $M = 10, dM = 5, \sigma = 5$

Як видно з вищенаведених графіків при зростанні значення дисперсії ряду зростає стохастична компонента часового ряду, що в свою чергу спотворює його детерміновану природу.

При цьому основні трендові та циклічні показники ряду зберігаються. При накладанні детермінованого та стохастичного рядів видно їх схожу структуру. На рис. 4.13 зображено накладання детермінованого та стохастичного ряду.

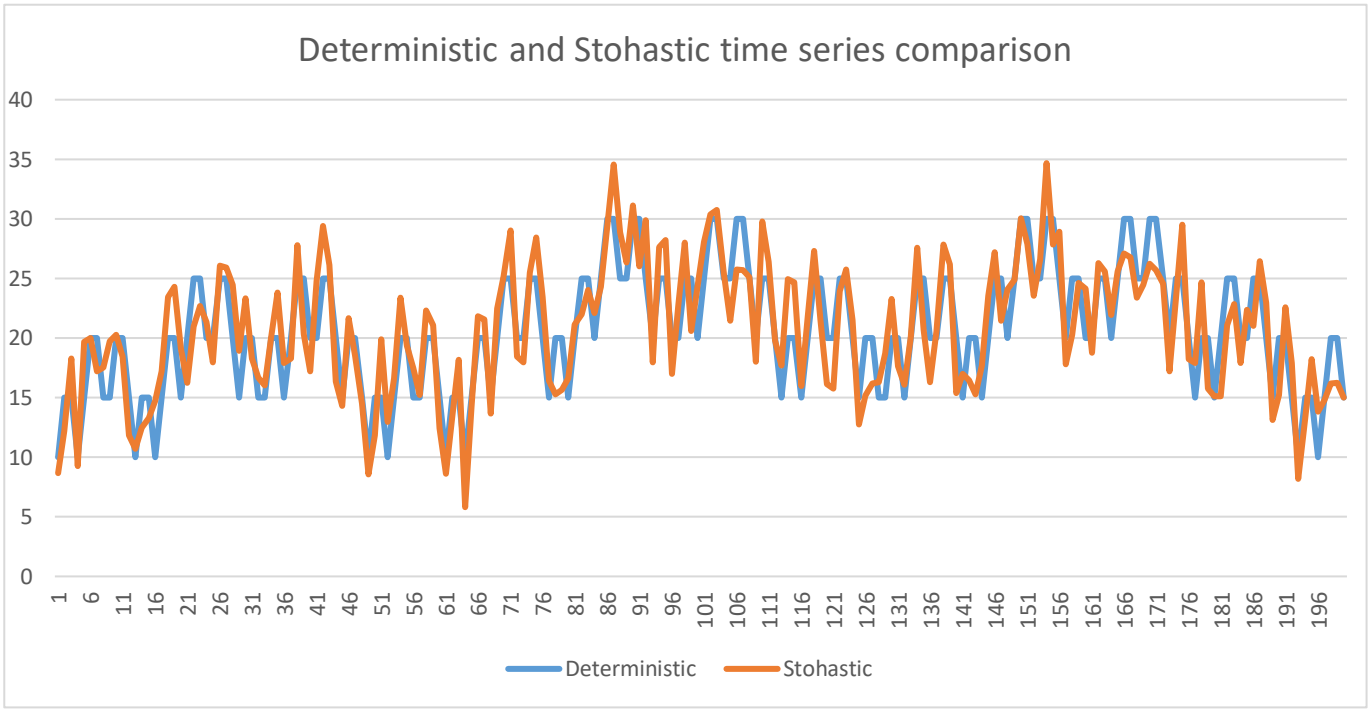


Рисунок 4.13 – Накладання детермінованого та стохастичного рядів

В процесі тестування програмної системи для реконструювання стохастичних рядів було виявлено різке зростання складності обчислень. Складність відтворення стохастичних рядів пояснюється необхідністю врахування дисперсії математичного очікування як одну зі складових конструктивної моделі. У зв'язку з цим було проведено деякі зміни в програмній системі, а саме в модулі який відповідає за функціонал генетичного алгоритму. Зміни в системі полягають в розширенні структури хромосоми та конструктивної моделі як такої за рахунок впровадження додаткового гену який являє собою дисперсію ряду. Також було змінено принцип обчислення фітнес-функції генетичного алгоритму.

Фітнес-функція [57] яка обчислюється для хромосоми яка має ненульову дисперсію наступним чином:

- на основі конструктивної моделі яка складається з генів хромосоми за допомогою конструктора будується мультисимвольний фрактальний ланцюжок за довжиною рівний довжині вхідного ряду;
- на основі мультисимвольного ланцюга який є неоднорідним носієм, математичного очікування, відхилення математичного очікування та дисперсії за допомогою конструктора фрактального мультисимвольного ряду конструюється ряд;
- обчислюється середнє значення середньоквадратичного відхилення значень вхідного та сконструйованого ряду та зберігається як результат;
- кроки 2 та 3 повторюється задану кількість разів згідно з конфігурацією;
- обчислюється значення фітнес функції як математичне очікування згідно зі значень середнього квадратичного відхилення для кожної ітерації реконструювання.

Ітеративний підхід при обчисленні фітнес-функції дає змогу локалізувати стохастичну компоненту вхідного стохастичного часового ряду, що дає можливість адекватно порівнювати значення фітнес-функцій хромосом між собою і таким чином визначати більш та менш пристосованих хромосом на основі їх фрактальної конструктивної моделі.

Слідує відмітити що певна неоднорідність та стохастичність яка була додана до процесу обчислення фітнес-функції загалом підвищує час для виконання обчислень, але при цьому фактично складність зросла не пропорційно до кількості спроб реконструювання в фітнес-функції. Менш стрімке зростання складності обчислень пояснюється за рахунок неоднорідності значень фітнес-функцій кожної хромосоми. Одна й та сама хромосома на кожній ітерації має різне значення своєї фітнес-функції в межах невеликого відхилення. Дана особливість позитивно впливає на принцип роботи генетичного алгоритму, так як при такому підходу зменшується статичність результатів конструювання на кожній ітерації, що каже про те що в популяції не буде чітко виражених лідерів, домінуючі гени яких швидко розповсюджуються в межах популяції. Стрімке розповсюдження домінуючих генів – це одна з фундаментальних проблем сімейства генетичних алгоритмів.

Розповсюдження генів хоч і являється одним з основних інструментів зміни генів який дозволяє ефективно забезпечувати процес комбінаторної рекомбінації генів, та даний підхід має і негативні сторони, з основних можна виділити наступні:

– виродження генофонду популяції внаслідок домінування генів;
– застрягання алгоритму в локальному мінімумі;
– складність підбирання більш оптимального рішення при перебуванні в локальному мінімумі.

Вищеописані фактори які негативно впливають на часову ефективність роботи системи після останніх нововведень зменшили ступінь свого впливу, що було виявлено екпериментально.

Для проведення випробувань в контексті даного етапу було використано детерміновані часові ряди з ненульовою дисперсією математичного очікування значення ряду. Ненульова дисперсія додає стохастичну компоненту до вхідного ряду. На рис. (4.14 – 4.15) зображено фрактальний стохастичний та реконструйовані часові ряди.

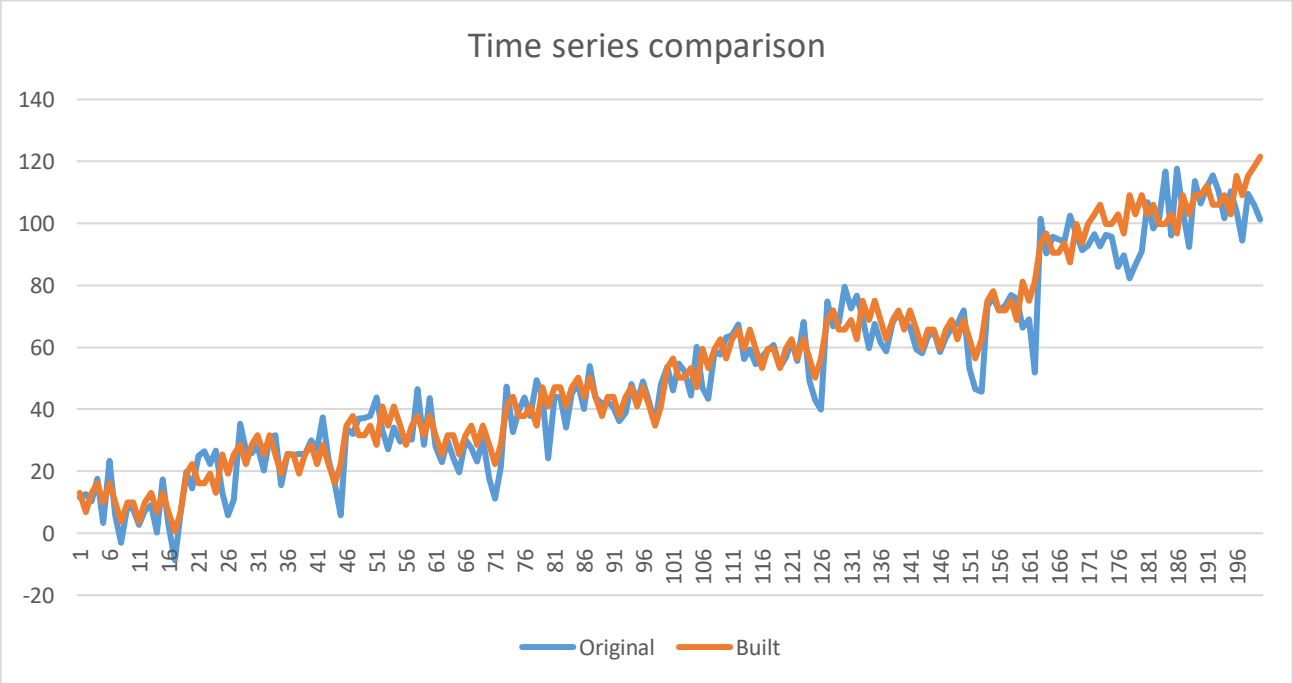


Рисунок 4.14 – Стохастичний часовий ряд та реконструйований часовий ряд для конструктивної моделі:

$$S \to f, f \to g{+}{+}g{-}{-}g, g \to f - f{+}{+}f{-}{-}, M = 13, dM = -4.2$$

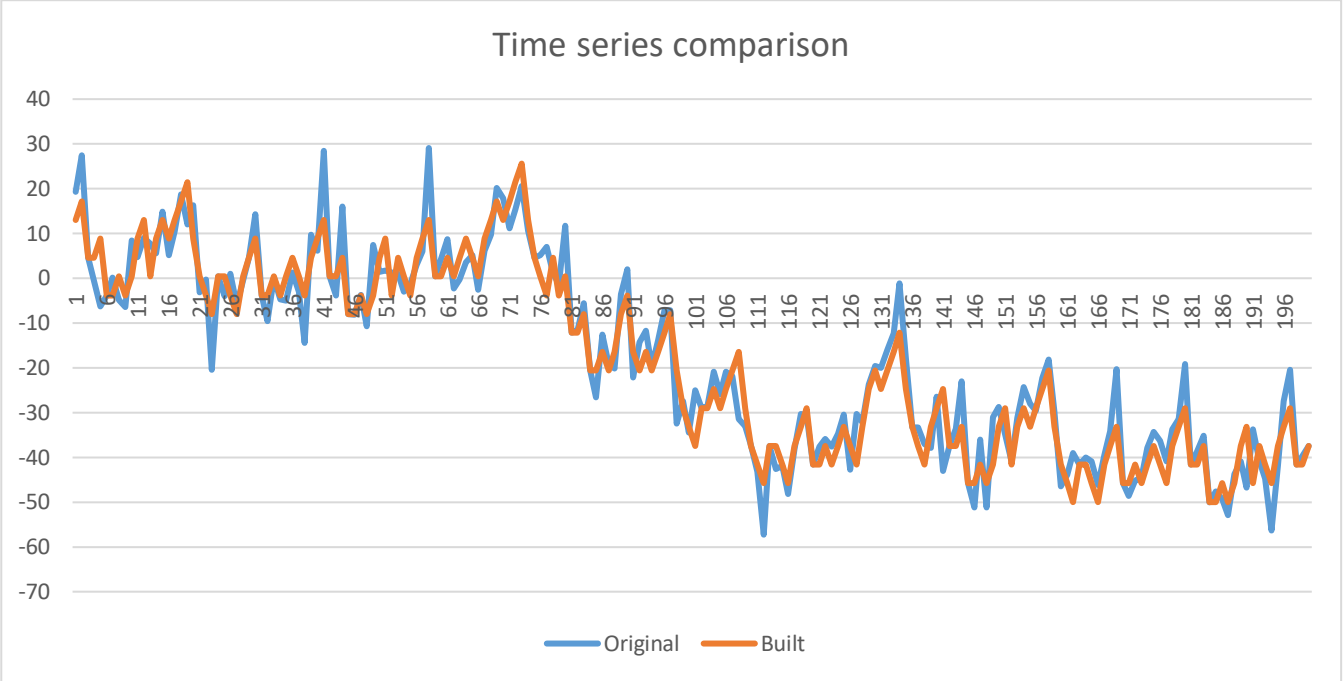


Рисунок 4.15 – Стохастичний часовий ряд та реконструйований часовий ряд для конструктивної моделі:

$$S \to f, f \to g+g+h--g, g \to h-g+++f-, h \to f-g+h-f, M=12.6, dM=-3.8$$

Нижче наведено аналіз результатів випробувань для реконструювання стохастичних часових рядів.

Таблиця 4.6 – Результати випробувань на стохастичних рядах

Кількість правил заміщення	Середня кількість символів у правилах	Кількість значень ряду	Середня кількість ітерацій	Максимальна кількість ітерацій	Середній час затрачений на випробування (с)	Максимальний час затрачений на випробування	Швидкість ітерацій (ітерацій /с)
1	8	200	2471	7183	574	1670	4.3
2	8	206	4363	12739	1212	3538	3.6
3	7.5	209	15342	84901	4794	26531	3.2

З результатів видно, що середній та максимальний час на проведення випробувань збільшився в результаті зміни процесу обчислення фітнес-функції генетичного алгоритму та розширення структури хромосоми за рахунок додаткового гену. При цьому слідує відмітити, що система стала більш ефективною для роботи зі стохастичними часовими рядами згідно з результатами тестування попередньої версії системи на стохастичних рядах.

Після проведення попереднього етапу випробувань маємо достатньо статистичних даних для проведення порівняння часових характеристик розробленої системи та додатку який є результатом досліджень в попередній роботі в області застосування методу конструктивно-продукційного моделювання для часових рядів [20]. Для проведення порівняння часової ефективності були взяті результати випробувань з обох робіт при рівних вхідних даних для забезпечення об’єктивності порівняння часових характеристик.

В таблиці 4.7 наведено результати порівняння часових характеристик програмних забезпечень.

Таблиця 4.7 – Порівняння часових характеристик

Кількість правил заміщення в конструктивній моделі	Середній час затрачений на обчислення для системи – «мультиагентна система моделювання фрактальних часових рядів»	Середній час затрачений на обчислення для додатку – «Система для конструктивно-продукційного моделювання фрактальних складових наявних часових рядів»
1	574	1217
2	1212	7324
3	4794	19380

На рис. 4.16 зображено порівняння часових характеристик програмних продуктів згідно з таблицею 4.7.

Рисунок 4.16 – Порівняння часових характеристик

З результатів видно, що в процесі досліджень та розробки мультиагентної системи вдалось досягти більш ефективних часових характеристик в порівнянні з попередніми розробками в даній області.

Наступним етапом випробувань програмної системи є випробування для задачі прогнозування часових рядів [58]. В той час коли прогнозування часових рядів є формою прогностичного моделювання, аналіз часових рядів являється формою моделювання. Ціллю аналізу та моделювання часових рядів є виявлення тенденцій та сезонних закономірностей та зв’язати їх з певними зовнішніми чинниками. Ціллю прогнозування часових рядів є прогнозування майбутніх значень ряду до конкретного моменту в часі використовуючи деяку модель.

На даний момент існує безліч математичних методів моделювання часових рядів з ціллю їх подальшого прогнозування [59]. Як було сказано вище, для прогнозування майбутніх значень ряду використовується деяка модель, за допомогою якої можна відтворити заданий ряд. Кожний математичний метод тим чи іншим чином розроблявся для моделювання та аналізу часових рядів певної природи.

З цього можна зробити висновок, що для аналізу та прогнозування рядів необхідний висококваліфікований спеціаліст з аналізу даних, який в межах своєї компетентності та професіоналізму повинен вибрати найбільш оптимальний метод для аналізу певного ряду. Особливість конструктивно-продукційного методу для аналізу та прогнозування часових рядів полягає в тому, що метод не потребує значних маніпуляцій зі сторони кваліфікованої персони для ефективного відтворення конструктивної моделі вхідного часового ряду. Робота методу базується на ідеї фрактальної природи всіх природніх процесів.

Для використання методу конструктивно-продукційного моделювання в контексті задачі прогнозування часового ряду необхідно провести деяку модернізацію програмного інтерфейсу системи. Для проведення прогнозування вхідного часового ряду необхідно розширити модель вхідних даних додавши кількість періодів для прогнозу.

В результаті даного підходу буде згенеровано конструктивну модель, за допомогою якої можна реконструювати часовий ряд з продовженням на задану кількість періодів. Значення реконструйованого часового ряду які мають більший індекс ніж останній елемент вхідного часового ряду і будуть прогнозом майбутніх значень ряду. На рис. 4.17 зображено вхідний часовий ряд та реконструйований часовий ряд з продовженням.

Рисунок 4.17 – Вхідний часовий ряд та реконструйований часовий ряд з продовженням

Як видно на рисунку вище, реконструйований часовий ряд повторює фрактальну структуру вхідного часового ряду базуючись на його фрактальних властивостях які закладені в конструктивну фрактальну модель.

Для проведення випробувань на предмет прогнозування та реконструювання стохастичних часових рядів було використано часовий ряд замірів серцебиття людини в процесі проведення силового тренування. Для замірів серцебиття було проаналізовано тренування тривалістю – 40 хвилин. Слід відмітити, що даний ряд є часовим рядом з сильним стохастичним компонентом, так як силове тренування на такому тривалому інтервалі допускає сильні перепади серцебиття, на відмінну від більш сталих процесів, як наприклад електрокардіограма роботи серця. Слід відмітити, що реконструювання часових рядів які мають сильну стохастичну природу потребують значної кількості обчислювальних ресурсів.

На рис 4.18 зображено стохастичний часовий ряд замірів серцебиття людини в процесі проведення силового тренування.

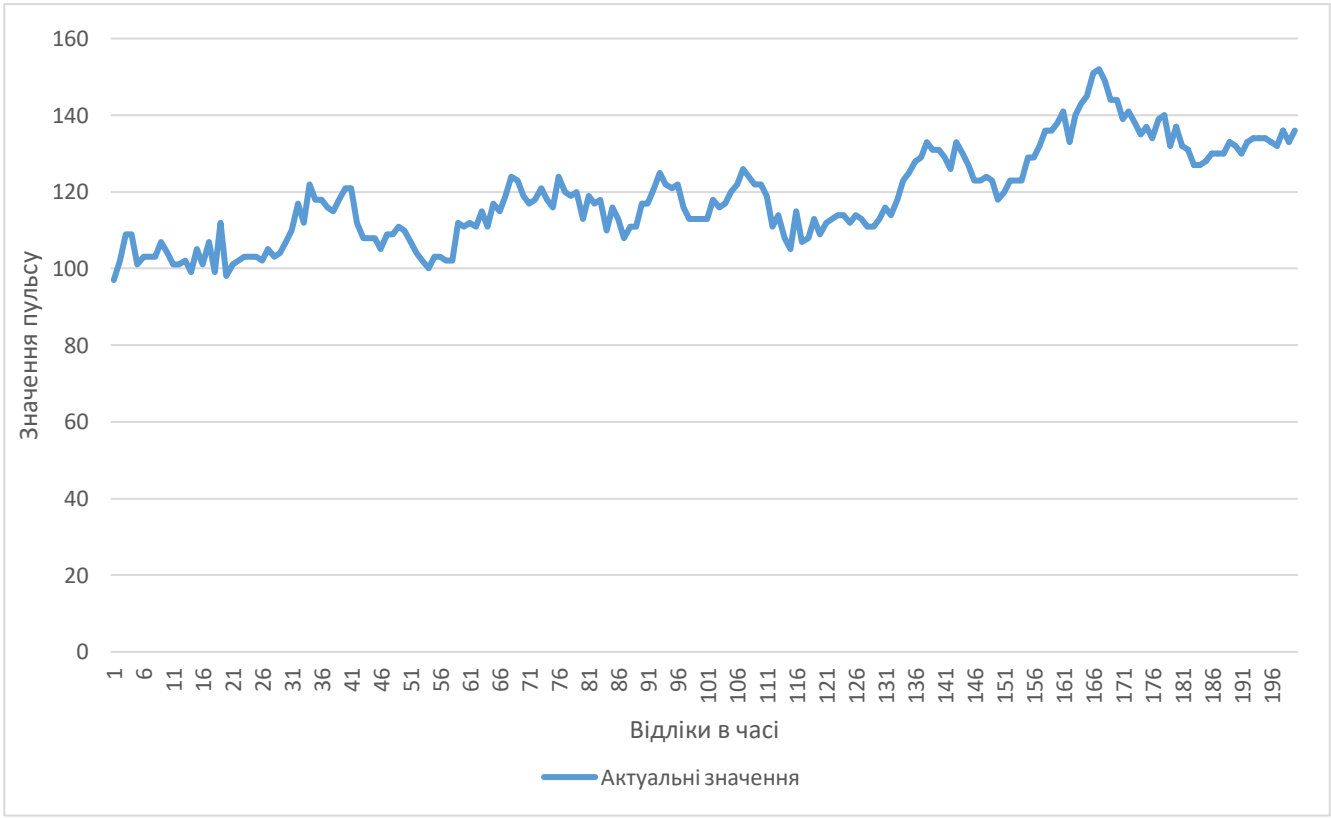


Рисунок 4.18 – Стохастичний часовий ряд серцебиття людини в процесі силового тренування

Для проведення випробування на предмет прогнозування було взято 90% значень ряду, тобто 180 значень з 200. В результаті мультиагентна система за допомогою методу конструктивно-продукційного моделювання та генетичного алгоритму повинна сформувати фрактальну конструктивну модель вхідного часового ряду та базуючись на ній побудувати часовий ряд з продовженням на 20 періодів.

Для подальшого аналізу результатів та для визначення ефективності методу конструктивно-продукційного моделювання для прогнозу фрактальних стохастичних часових рядів було прийняте рішення провести прогнозування ряду за допомогою інших математичних методів, таких як:

– поліноміальне прогнозування [60];
– експоненціальне прогнозування;
– лінійне прогнозування [61];
– метод експоненціального зглажування [62].

Результати випробування зображено на рис. 4.19.

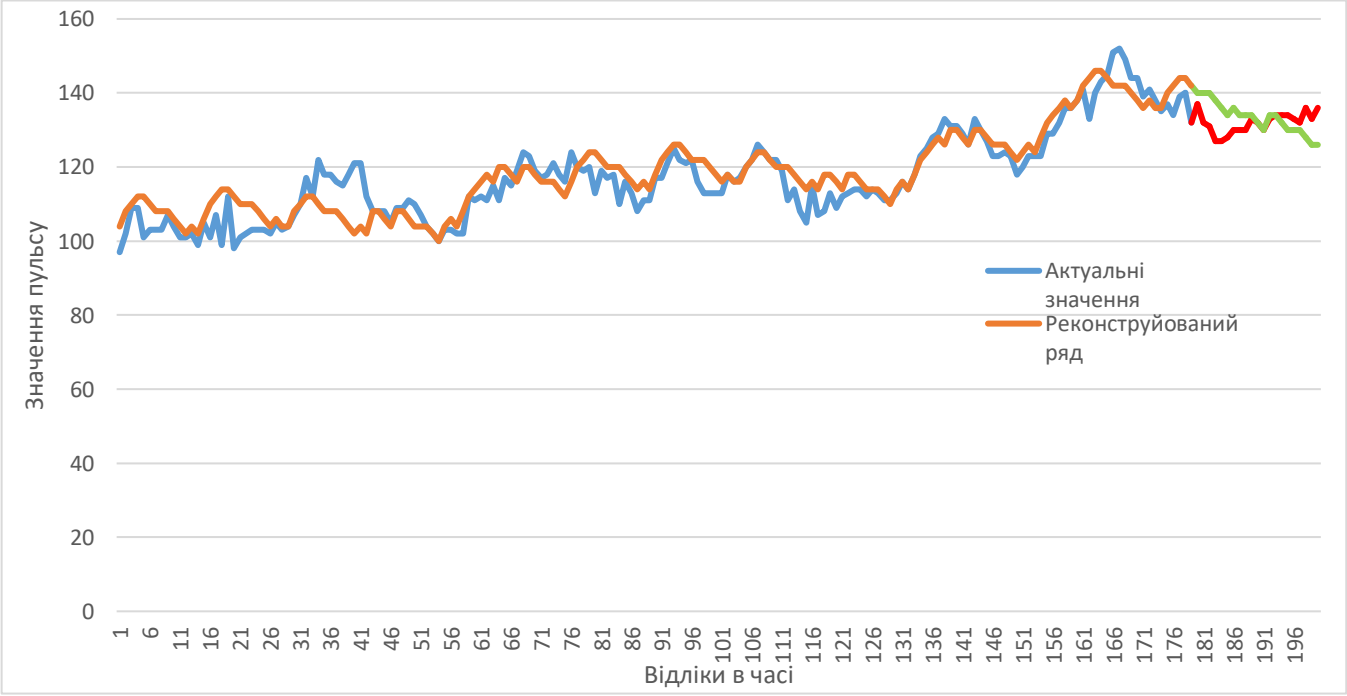


Рисунок 4.19 – Вхідний часовий ряд та реконструйований часовий ряд з продовженням для конструктивної моделі:

$$S \to f,\ f \to +gf - f - h;\ g \to h + + g + h;\ h \to f + f - f;\ M = 98.2;\ dM = 3.3;$$

На рис. 4.19 зображено результати реконструювання з подальшим прогнозуванням часового ряду. Синім зображено актуальні значення ряду які були надано для проведення обчислень, помаранчевим – реконструйований ряд за фрактальною конструктивною моделлю. Червоним кольором зображено актуальні значення ряду про які було невідомо мультиагентній системі, зеленим зображено прогнозовані значення ряду які були отримані в результаті конструювання ряду з продовженням базуючись на конструктивній моделі.

Як видно з результатів, можна спостерігати деяку дивергенцію між актуальними та зпрогнозованими значеннями, але на середині інтервалу можна спостерігати точний збіг в значеннях рядів. На рис. 4.20 зображено актуальні та зпрогнозовані значення рядів в більшому масштабі.

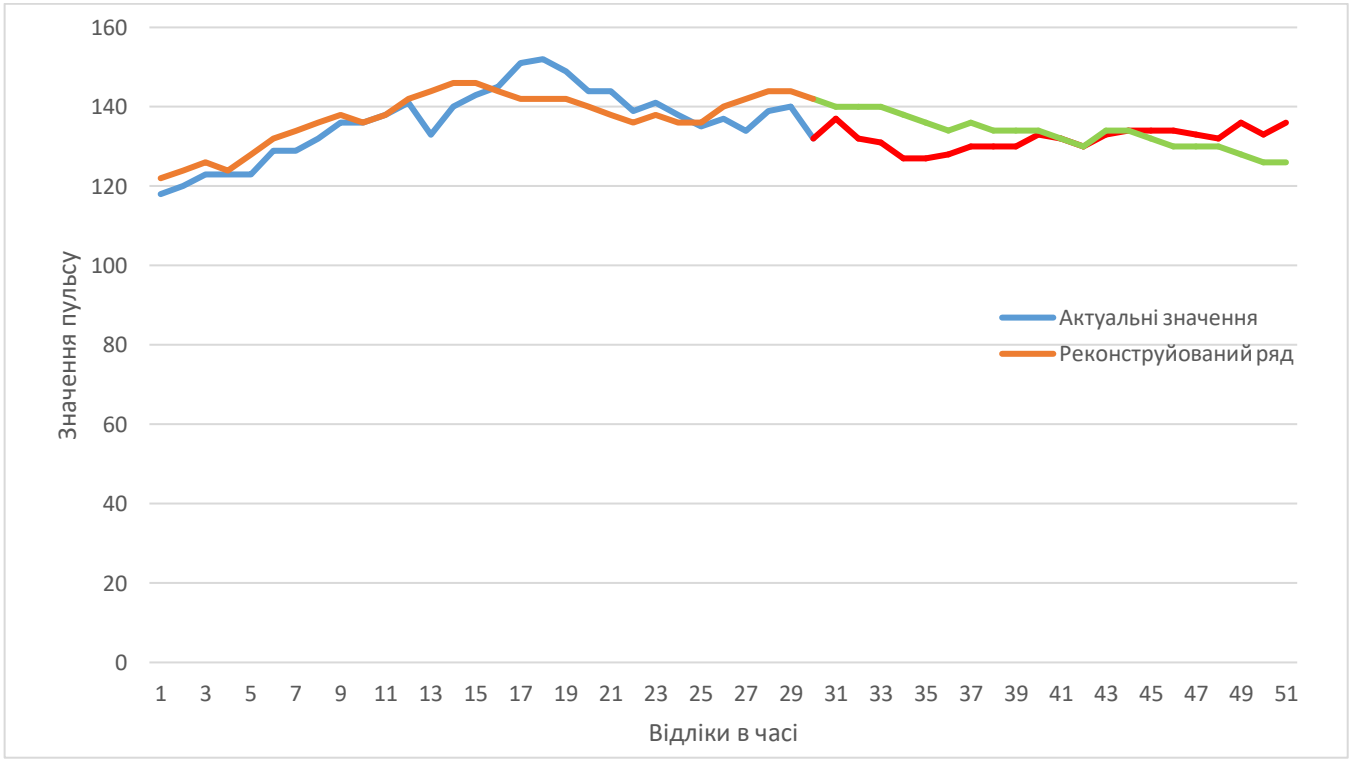


Рисунок 4.20 – Актуальні та зпрогнозовані значення рядів в більшому масштабі

Для проведення подальшого аналізу ефективності методу конструктивно-продукційного моделювання необхідно порівняти його ефективність з іншими методами прогнозування часових рядів. Для цього було проведено розрахунки з прогнозування часового ряду за допомогою методів поліноміального, експоненціального та лінійного прогнозування значень ряду.

Для ефективного порівняння якості прогнозу різних методів в порівняння з методом конструктивно-продукційного моделювання необхідно провести подальші розрахунки відхилення між актуальними та прогнозованими значеннями ряду. Це допоможе більш точно визначити актуальність та ефективність методу для використання його в контексті моделювання, аналізу та прогнозування значень.

Для проведення порівняння якості прогнозу були обрані наступні загальновідомі методи прогнозування часових рядів:

– метод поліноміального прогнозування;
– метод експоненціального прогнозування;
– метод лінійного прогнозування;
– метод експоненціального зглажування.

На рис. (4.21 – 4.26) зображено результати прогнозування математичних методів в порівнянні з результатом прогнозу методом конструктивно-продукційного моделювання.

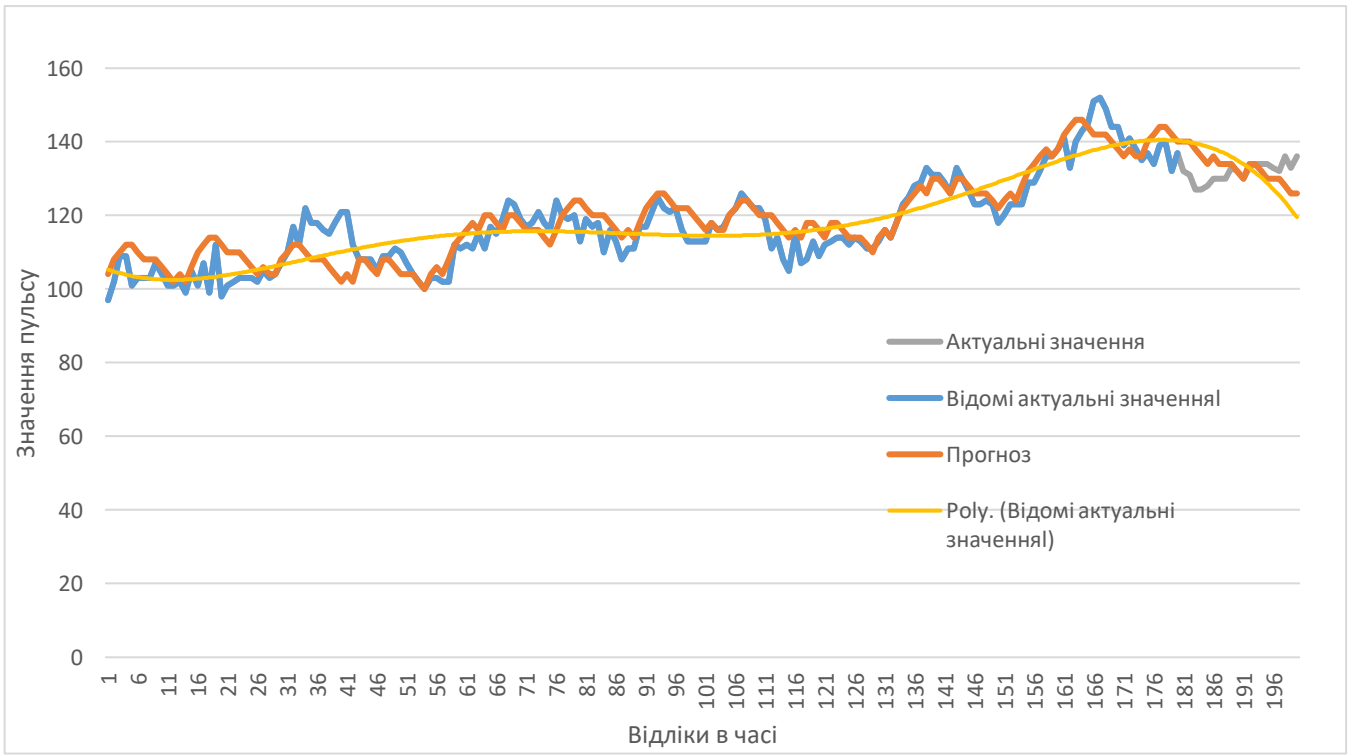


Рисунок 4.21 – Прогнозування значень ряду за допомогою полінома п’ятого ступеню в порівнянні з прогнозом методу конструктивно-продукційного моделювання

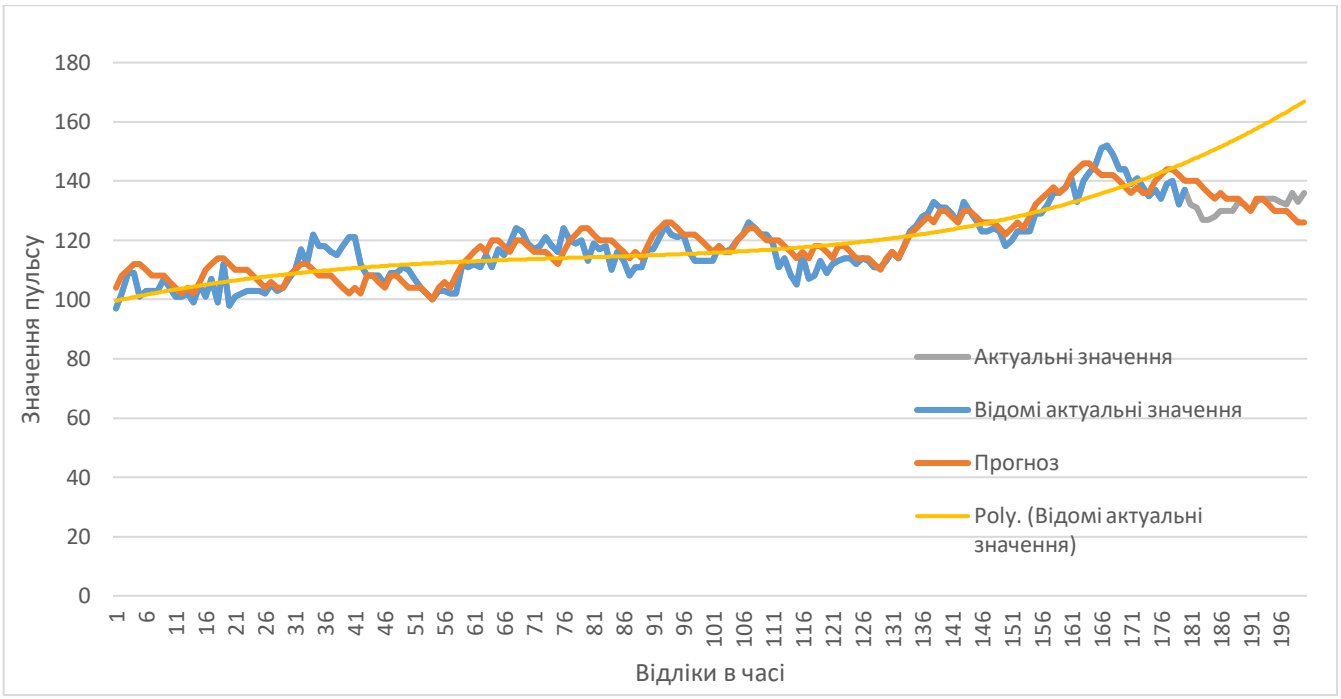


Рисунок 4.22 – Прогнозування значень ряду за допомогою полінома четвертого ступеню в порівнянні з прогнозом методу конструктивно-продукційного моделювання

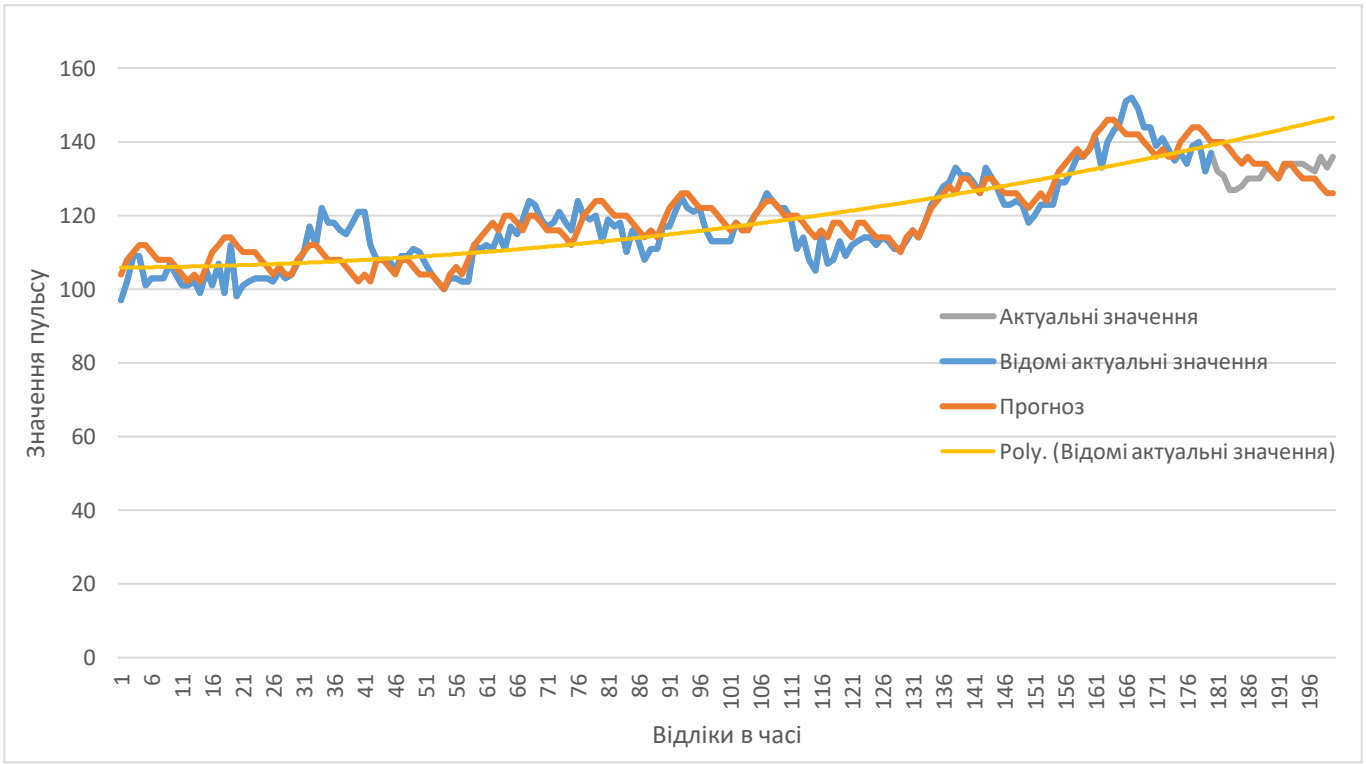


Рисунок 4.23 – Прогнозування значень ряду за допомогою полінома другого ступеню в порівнянні з прогнозом методу конструктивно-продукційного моделювання

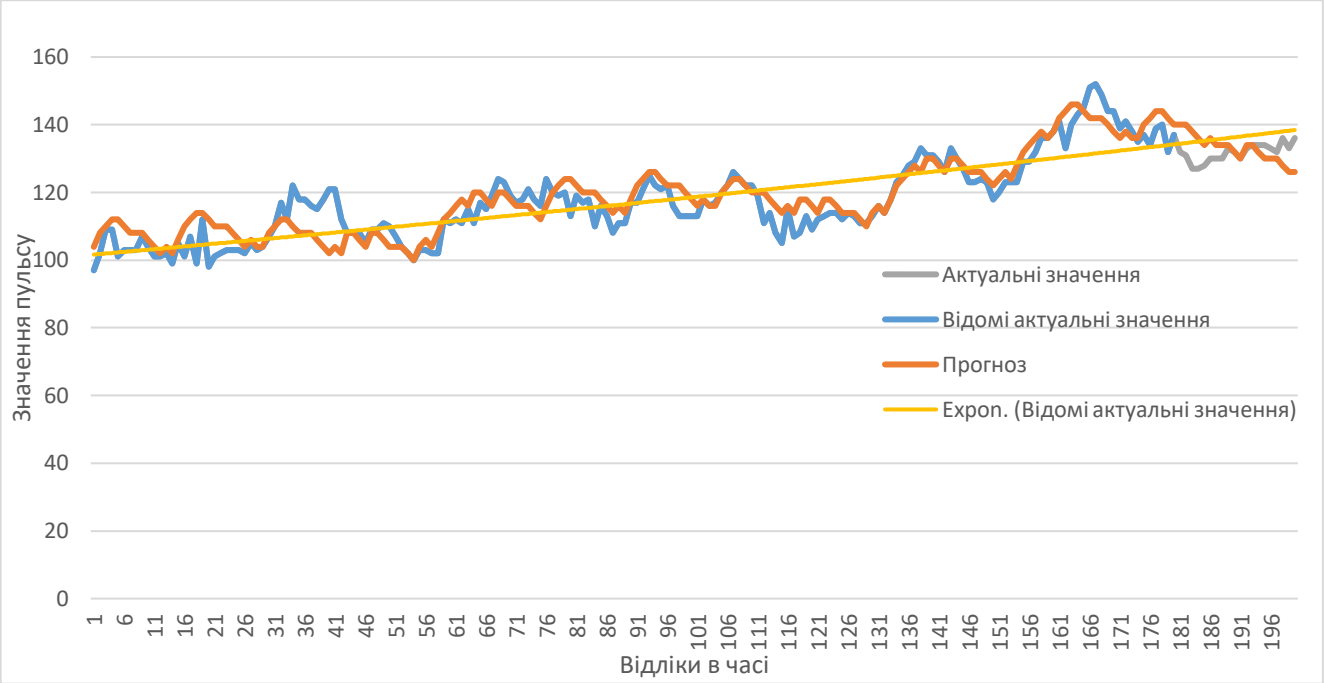


Рисунок 4.24 – Прогнозування значень ряду методом експоненціального прогнозування в порівнянні з прогнозом методу конструктивно-продукційного моделювання

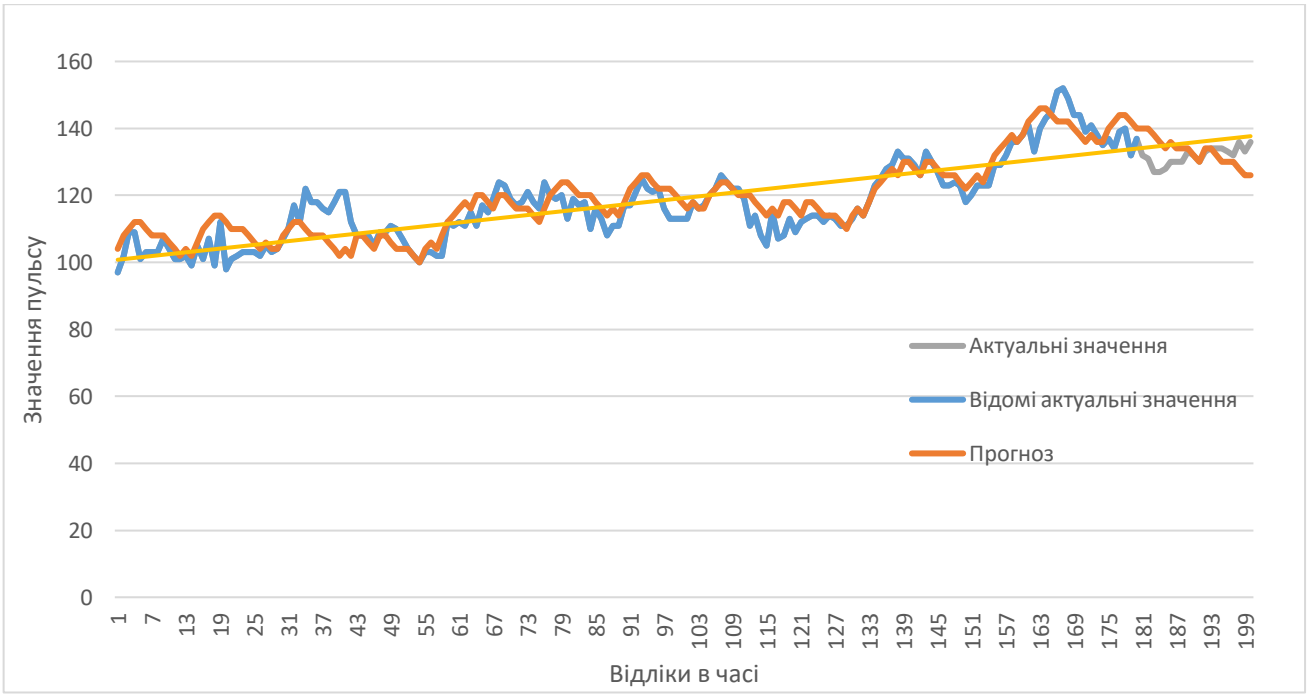


Рисунок 4.25 – Прогнозування значень ряду методом лінійного прогнозування в порівнянні з прогнозом методу конструктивно-продукційного моделювання

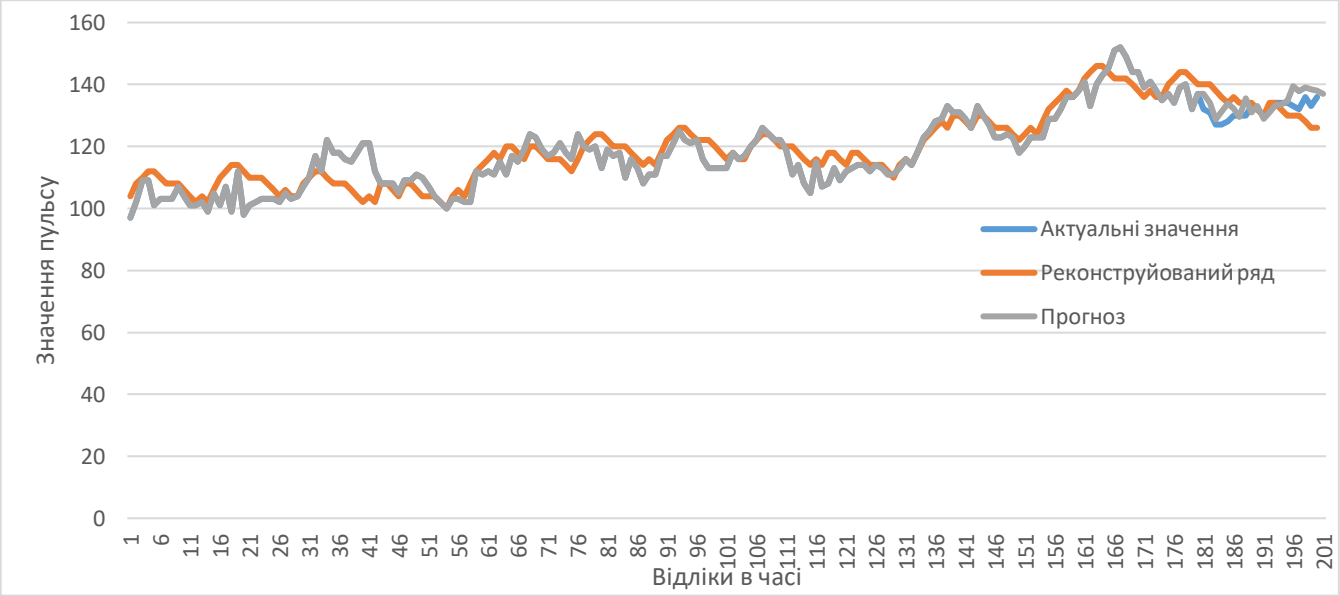


Рисунок 4.26 – Прогнозування значень ряду методом експоненціального зглажування в порівнянні з прогнозом методу конструктивно-продукційного моделювання

Далі проведено розрахунки методом середньоквадратичного відхилення для визначення ефективності методу конструктивно-продукційного моделювання для задач прогнозування фрактальних стохастичних часових рядів в порівнянні з іншими математичними методами прогнозування. В таблиці 4.4 наведено результати дослідження ефективності методу конструктивно-продукційного моделювання в порівнянні з іншими математичними методами прогнозування.

Таблиця 4.8 – Результати випробувань для прогнозування

Назва методу	Кількість періодів прогнозування	Середнє відхилення значень ряду
Реконструйований ряд	20	4.609
Поліном п’ятого ступеню	20	7.169
Поліном четвертого ступеню	20	21
Поліном другого ступеню	20	8.5
Експоненціальне прогнозування	20	3.7
Лінійне прогнозування	20	3.39
Експоненціальне зглажування	20	2.94

З результатів видно, що метод експоненціального зглажування виявився більш ефективним в порівнянні з іншими методами прогнозування. Метод конструктивно-продукційного моделювання виявився менш ефективним ніж метод експоненціального зглажування, більш ефективним в порівнянні з поліноміальним прогнозуванням та ненабагато поступається методам експоненціального та лінійного прогнозування через більше значення середнього відхилення.

На рис. 4.27 зображено порівняння методу конструктивно-продукційного моделювання з методом експоненціального зглажування для прогнозування часового ряду.

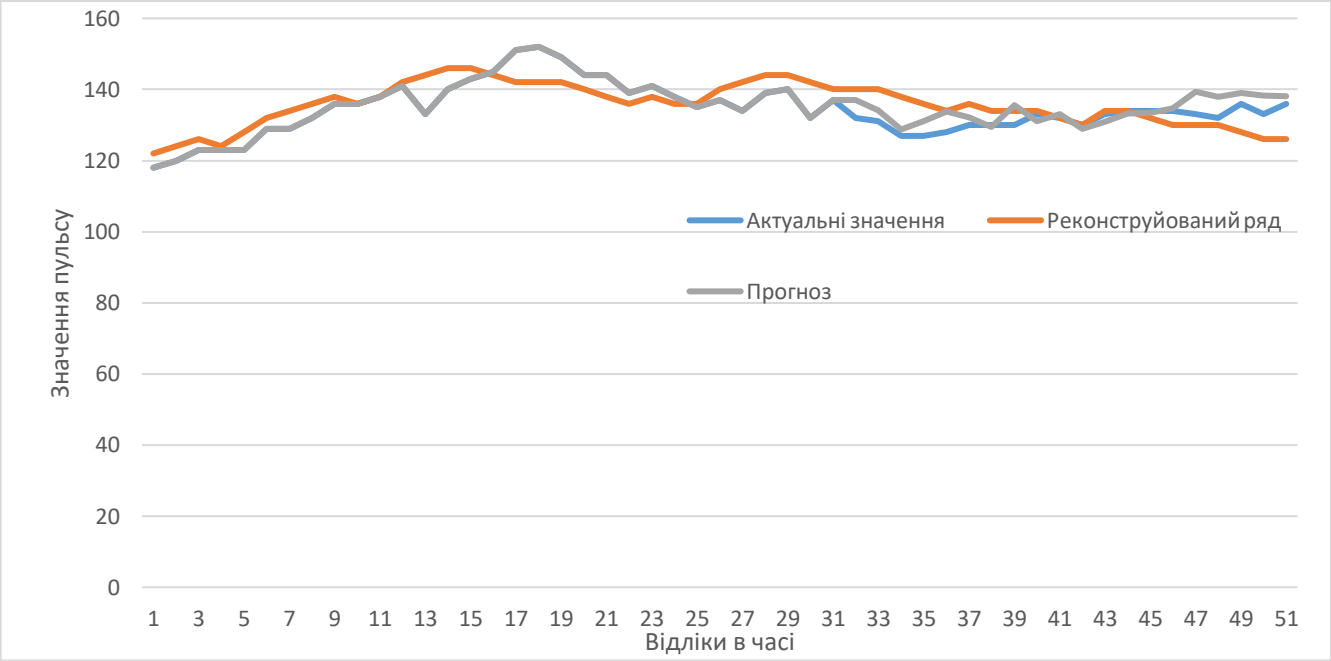


Рисунок 4.27 – Порівняння методу конструктивно-продукційного моделювання з методом експоненціального зглажування для прогнозування часового ряду

Особливість методу конструктивно-продукційного моделювання для аналізу та прогнозування часових рядів полягає в тому, що на відміну від інших математичних методів, прогноз часового ряду може бути значно ефективнішим через відсутність агрегації та усереднення значень ряду в процесі роботи методу. Це дозволяє прогнозувати різкі зміни значень графіку, якщо вони піддаються деяким закономірностям.

Висновки до четвертого розділу

В даному розділі було описано проведення випробувань та вдосконалення програмної системи. Основний акцент в ході випробувань було зроблено на використання методу для роботи зі стохастичними рядами. Було проведено наступні випробування:

- відтворення конструктивної моделі детермінованих рядів;
- відтворення конструктивної моделі детермінованих рядів різної складності з різним об’ємом обчислювальних ресурсів;
- відтворення конструктивної моделі недетермінованих часових рядів які були побудовані за допомогою конструктивної моделі.
- прогнозування стохастичних рядів за допомогою відтвореної конструктивної моделі.

Відповідно до кожного з етапів випробувань, крім останнього, було проведено 20 випробувань для визначення більш точних статистичних даних про результати роботи системи. Кожний етап випробувань завершився з позитивними результатами.

Під час випробувань на стохастичних рядах було змінено структуру роботи генетичного алгоритму в контексті обчислення фітнес-функції та зміни структури хромосоми. Також до змін роботи генетичного алгоритму відноситься імплементація механізмів схрещування, мутації та механізму катаклізму популяції.

Згідно з результатами проведених випробувань для детермінованих рядів та порівняння їх з результатами випробувань попередньої роботи в області використання методу конструктивно-продукційного моделювання часового ряду, можна зробити висновок про успішне підвищення часової ефективності даного методу, що і є метою даної роботи.

За результатами випробувань для прогнозування часових рядів та їх порівняння з результатами класичних методів прогнозування, було з’ясовано, що метод конструктивно-продукційного моделювання ненабагато поступається в ефективності методу експоненціального зглажування, але при цьому показує себе краще ніж поліноміальне прогнозування.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Для дотримання належного рівня працездатності працівників робочі місце та середовище повинні відповідати певним нормам безпеки. Також від самих працівників потребується дотримання визначених норм та правил роботи.

За регулювання вищезазначених елементів використовуються група нормативно-правових актів з охорони праці (скорочено НПАОП) та державні санітарні правила і норми роботи (скорочено ДСанПіН).

Виходячи з того, що робочий процес перш за пов‘язаний із взаємодією із комп‘ютерними пристроями, слід використовувати НПАОП та ДСанПіН, які відносяться до роботи з комп‘ютерною технікою та відповідним приміщенням.

5.1 Вимоги до безпеки при виконанні робіт на робочому місці

Вимоги стосовно умов праці трактуються законодавством України, а саме Законом України «Про охорону праці» від 2002 р. [63].

У свою чергу, основні положення, які трактують основні правила та вимоги, які стосуються професії та робочого процесу інженера-програміста можна поділити на дві умовні групи:

– загальні;
– відносно роботи з комп‘ютерною технікою.

До загальних можна віднести наступні положення про санітарні норми, правила та вимоги:

– державні санітарні норми виробничого шуму, ультразвуку та інфразвуку ДСН 2.3.6.037-99, затверджені постановою Головного державного санітарного лікаря України від 01.12.99 р. № 37 [64];
– державні санітарні норми виробничої загальної та локальної вібрації ДСН 3.3.6.039-99, затверджені постановою Головного державного санітарного лікаря України від 01.12.99 р. № 39 [65];
– державні санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджені постановою Головного державного санітарного лікаря України від 01.12.99 р. № 42 [66];

– загальні вимоги стосовно забезпечення роботодавцями охорони праці працівників, затверджені наказом МНС від 25.01.2012 р. № 67.

До нормативно-правових актів, пов’язаних з організацією експлуатації комп’ютерної техніки, належать:

– НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207 [67];
– правила безпеки під час навчання в кабінетах інформатики навчальних закладів системи загальної середньої освіти (НПАОП 80.0-1.12-04) [68];
– інструкція щодо надання послуг з ремонту побутової радіоелектронної апаратури [69];
– державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПіН 3.3.2-007-98) [70];

При розробці програмного додатку основою робочого процесу є взаємодія з ЕОМ. Відповідно до НПАОП 0.700-7.15-18 [71] можна визначити наступні групи вимог:

– вимоги безпеки до робочих місць працівників з екранними пристроями;
– мінімальні вимоги безпеки під час роботи з екранними пристроями;
– мінімальні вимоги безпеки до екранних пристроїв.

Вимоги безпеки до робочих місць працівників з екранними пристроями:

– робоче місце повинне буде спроектоване таким чином, щоб надавати працівнику необхідний простір для зміни робочого положення та рухатися;
– випромінення від екранних пристроїв (до даного терміну відносять такі фактори як шум, вібрації, забруднювачі, температура) повинне бути зведене до граничного допустимого рівня відповідно до потреб безпеки та охорони здоров‘я;

– організація та розташування робочого місця та його елементів повинні відповідати ергономічним, антропологічним, психофізіологічним вимогам та характеру робіт;
– освітлення робочого місця повинне надавати необхідний контраст між екраном та навколишнім середовищам та відповідати вимогам ДСанПІН 3.3.2.007-98 [70];
– мікроклімат повинен підтримуватися на постійному рівні та відповідно до вимог Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [66];
– робочій стіл або поверхня повинна мати достатній розмір та поверхню з низькою відбивною здатністю та також допускати гнучкість при розміщенні документів, пристроїв (екрану або клавіатури) та відповідного устаткування;
– робоче крісло повинне дозволяти працівнику приймати зручне положення та легко рухатися.

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

– на початку кожного дня необхідно провидити очистку екранного пристрою від пилу та інших забруднень;
– в кінці робочого дня та при виникненні аварійної ситуації необхідно відключити екранний пристрій від мережі;
– при виконанні робіт операторського типу у приміщеннях необхідно підтримувати оптимальні умови мікроклімату відповідно до ДСН 3.3.6.042-99 [66].

Не допускається:

– виконувати технічне обслуговування, ремонт або налагодження екранних пристроїв на робочому місці працівника та під час роботи;
– відключати захисні пристрої, самому змінювати конструкцію та складові пристроїв або їх технічне налагодження;
– працювати з екранними пристроями які мають прояви несправності (виникають нехарактерні сигнали, нестабільне зображення та інше).

5.2 Шкідливі виробничі фактори на підприємстві

Шкідливий виробничий фактор (чинник) — елемент робочого середовища, який може впливати на робітника підчас робочого процесу та при певних умовах може викликати захворювання або зменшення працездатності.

Виходячи з робочого процесу інженера-програміста можна визначити наступні шкідливі виробничі фактори робочого середовища:

- поганий мікроклімат приміщення (підвищена або знижена температура повітря, запиленість, загазованість повітря, підвищена або знижена вологість повітря);
- недостатня освітленість робочого місця;
- занадто високий рівень шуму;
- занадто високий рівень іонізуючого випромінювання;
- занадто високий рівень електромагнітних полів;
- занадто високий рівень статичної електрики;
- небезпека поразки електричним струмом;
- недостатня контрастність/яскравість екрана монітору;
- порушення ергономічних норм.

5.2.1 Мікроклімат приміщення

Мікроклімат робочого приміщення — клімат внутрішнього середовища визначеного приміщення, який визначається фізичними показниками, які у свою чергу мають вплив на організм людини.

До показників мікроклімату можна віднести:

- температуру повітря;
- відносну вологість;
- швидкість руху повітря.

Відповідно від пори року, характеру трудового процесу і характеру виробничого приміщення встановлені рекомендовані значення для кожного з показників. Відповідно до класифікації тяжкості робочого процесу та ДСН 3.3.6.042-99 [66] у таблиці 5.1 представлені значення параметрів мікроклімату та

показники робочого приміщення, де проводилася робота над розробкою програмного продукту.

Таблиця 5.1 – Нормативні та визначені параметри мікроклімату приміщення

Період року	Параметри мікроклімату	Нормативна величина	Величина виміряна у приміщенні
Холодний	Температура повітря в приміщенні	22-24°С	23,5°С
	Відносна вологість	40-60%	54,7%
	Швидкість руху повітря	До 0.1 м/с	0,084 м/с
Теплий	Температура повітря в приміщенні	23 - 25° С	24,32°С
	Відносна вологість	40-60%	50,2%
	Швидкість руху повітря	До 0.1 – 0.2 м/с	0,094 м/с

Для підтримання нормативних параметрів мікроклімату слід використовувати зволожувачі повітря. Також приміщення повинно бути обладнане кондиціонерами повітря, або мати систему вентиляції.

Також у контексті мікроклімату приміщення слід розглянути показники подачі свіжого повітря до робочого приміщення відповідно до пори року, об‘єму приміщення та кількості людей. У таблиці 5.2 представлені рекомендовані норми подачі свіжого повітря в приміщення з ПК.

Таблиця 5.2 – Норми подачі свіжого повітря в приміщення з ПК

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, м3 на одну людину в годину
Об'єм до 20м^3 на людину	не менше 30
20 - 40 м^3 на людину	не менше 20
Більше 40 м^3 на людину	може бути використана природна вентиляція

Виходячи з отриманих даних можна стверджувати, що робоче приміщення в якому проходив процес розробки програмного забезпечення повністю відповідає визначеним нормам та вимогам.

5.2.2 Освітлення робочого місця

Виходячи з того, що робочий процес розробки програмного додатку проходить у приміщенні, освітлення повинне наближатися до оптимальних умов зорового сонячного рівня. Загалом вимоги до освітлення приміщень із встановленими ПК наступні:

- загальна освітленість 300 лк та 750 лк комбінована — при виконанні зорових робіт високої точності;
- загальна освітленість 200 лк та 300 лк комбінована — при виконанні зорових робіт середньої точності;

Слід зазначити, що випромінювачі світла повинні бути встановлені таким чином, щоб не створювати полисків на поверхні екрана. Саме поле зору повинне бути освітлено рівномірно.

Процес розробки програмного додатку слід віднести до типу робіт середньої точності. У свою чергу, значення загальної освітленості робочого місця дорівнювало 215,62 лк, а загальна освітленість дорівнювало 284,75 лк. Для освітлення робочого приміщення було використано природній та штучний джерела освітлення, які були встановлені відповідно до вимог.

Також слід зазначити, що екран монітора також є джерелом світла. При довгій роботі з даним пристроєм з‘являється стомленість очей. Це може бути обумовлене високим рівнем яскравості, встановленим користувачем. Також миготіння та нечітка картинка можуть ускладнювати робочий процес. Для уникнення подібних ситуацій слід персонально встановлювати рівень яскравості зображення, а частота кадрів у свою чергу повинне бути не меншим за 75 Гц для ЕЛТ-моніторів та 60 Гц для ЖКИ-моніторів.

5.2.3. Шум та вібрації

Вібрації — це коливання твердих тіл, частин апаратів, машин, устаткування, споруд, що сприймаються організмом людини як струс. У свою чергу шум — це звуковий процес, який викликає дискомфорт та негативно впливає на організм людини.

Після тривалої дії шуму інтенсивності вище 80 дБ можуть виникнути патології слухового органу та негативно впливає на нервову систему. Під дією шуму працівник швидко втомлюється, що може привести до виробничих помилок під час робочого процесу.

Зниження рівня шуму та вібрації може бути досягнуто наступними засобами:

– стіни та стеля приміщення можуть бути облицьовані звуковбирний матеріалами;
– використанням м‘яких килимків, прокладок із гуми або повсті або спеціальних кожухів;
– використанням працівниками навушників, вкладишів, т. п.

Джерелом шуму та вібрацій у робочому середовищі працівників сфери програмування загалом є різного роду апаратура. Основними представниками є:

– персональний комп‘ютер або ноутбук та його складові;
– факси та принтери;
– інша комп‘ютерна апаратура (телевізійні екрани, т. п.).

Головним та єдиним джерелом шуму та вібрацій у робочому середовищі, у якому розроблявся програмний продукт, був саме персональний комп‘ютер. Загальний рівень шуму становив 32 дБ, а рівень вібрацій ПК — 62,9 дБ. Слід відзначити, що рівень вібрації на самому робочому місці дорівнює нулю (не фіксується пристроями). Дані значення відповідно до ДСН 3.3.6.039-99 [65] відповідають вимогам.

5.3 Дії працівників в надзвичайних ситуаціях

5.3.1 Порядок надання домедичної допомоги

Відповідно до Наказу МОЗ №398 від 16.06.2014 «Про затвердження порядків надання домедичної допомоги особам при невідкладних станах» [71] встановлений перелік документів, кожний з яких являє собою порядок дій при наданні домедичної допомоги. Загалом даний документ включає в себе перелік із 29 протоколів дій на окремі випадки, які потребують невідкладної допомоги.

В контексті роботи над програмним додатком найбільший ризик для працівника походив від великої кількості електронних приладів, які у свою чергу є джерелом ризику ураження електричним струмом. Відповідно до Порядку надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою [72] нижче представлений порядок дій при наданні домедичної допомоги при ураженні електричним струмом:

- переконатися у відсутності небезпеки;
- якщо джерело струму продовжує діяти на постраждалого, необхідно вимкнути його від електромережі за допомогою електронепровідного засобу;
- провести огляд життєвих показників постраждалого;
- викликати бригаду медичної допомоги;
- при відсутності у постраждалого дихання необхідно провести серцево-легеневу реанімацію
- при відсутності свідомості надати постраждалому стабільного положення;
- накласти стерильну пов‘язку на місце опіку;
- забезпечити нагляд за потерпілим до прибуття бригади невідкладної допомоги;
- повторно зателефонувати диспетчеру при погіршенні стану потерпілого.

5.3.2 Пожежна безпека

Задля запобігання пожежі у робочому приміщенні необхідно дотримуватися загальних правил пожежної безпеки [73].

У виникненні ознак пожежі необхідно дотримуватися наступних правил:

- повідомити службу пожежної безпеки про факт пожежі за вашою адресою;
- за можливістю вжити заходів протидії пожежі за допомогою вогнегасника;
- розпочати процес евакуації приміщення;
- повідомити керівництво про факт пожежі;
- перед тим як схопитися за ручку дверей, доторкніться до неї тільною стороною долоні задля перевірки температури;
- обов‘язково зачиняйте за собою усі двері;
- при великій задимленості приміщення пересувайтесь поповзом.

Висновки до п‘ятого розділу

У даному розділі були розглянути основні питання, які стосувалися безпеки та охорони праці під час процесу розробки програмного додатку.

Були розглянуті основоположні документі які визначають необхідні норми та вимоги до робочого процесу а також в цілому до середовища. Були визначені основні параметри мікроклімату, вібрації, шуму та освітлення для конкретного робочого. Після їх визначення результати були співставленні з нормативними значеннями, описаними у відповідних документах.

Також були розглянуті порядки дій при наданні домедичної допомоги та дії при пожежній небезпеці. Були розглянуті відповідні нормативні документи з протоколами дій для запобігання та під час самої надзвичайної ситуації.

ВИСНОВКИ

В ході роботи над проектом було сформовано підхід для підвищення часової ефективності існуючого методу конструктивно-продукційного моделювання для визначення конструктивних моделей часових рядів.

На початку роботи було визначено інструменти для проектування та розробки програмної системи та основні концептуальні та теоретичні поняття та їх складові. Було проаналізовано аналогічне програмне забезпечення, яке використовує схожий підхід для вирішення проблем аналізу, моделювання та прогнозування часових рядів.

Основним додатком на який спирався процес проектування та розробки системи був додаток який є результатом попередньої дипломної роботи в якому застосовується метод для моделювання часових рядів.

На початковому етапі проектування були вирішені концептуальні особливості системи в контексті мультиагентного підходу. Основні аспекти даного підходу полягали в визначенні класів агентів мультиагентної системи та розділення обов’язків та знань між ними.

В результаті аналізу архітектури агентів було обрано гексагональну архітектуру як базову архітектурну модель.

Після проведення порівняльного аналізу між способами комунікації було обрано асинхронну чергу повідомлень на базі протоколу AMQP для забезпечення обміну повідомленнями між агентами.

Наступним етапом було визначення можливості оптимізації методу конструктивно-продукційного моделювання в наступних аспектах:

- підвищення часової ефективності конструювання моделей вхідного ряду;
- підвищення часової ефективності роботи генетичного алгоритму;
- підвищення часової ефективності за рахунок масштабування обчислювальних ресурсів.

В ході даного етапу була визначена концепція розподілення роботи генетичного алгоритму на множину агентів, розширення функціональності

генетичного алгоритму за рахунок імплементації множини способів проведення фаз мутації та схрещування, інтеграція механізму катаклізму популяції, організація способу міграції генофондів між генетичними алгоритмами.

Після проведення проектування системи та визначення плану проведення оптимізації та підвищення часової ефективності було розпочато процес розробки системи, в результаті якого було розроблено 4 класи агентів системи. Також була розроблена конфігурація розгортання агентів на робочих машинах.

Після завершення процесу розробки було розпочато етап експериментальних випробувань системи для визначення ефективності розроблених засобів для підвищення часової ефективності роботи методу конструктивно-продукційного моделювання. Було визначено наступні етапи випробувань:

- випробування на детермінованих часових рядах породжених одним та множиною правил L-системи;
- випробування на недетермінованих часових рядах породжених одним та множиною правил L-системи;
- випробування на часових рядах природнього характеру на предмет прогнозування значень.

Після проведення випробувань на детермінованих та недетермінованих рядах було проведення порівняння часової ефективності між розробленою системою та додатком який є результатом попередніх досліджень в даній області. Як результат, можна відмітити значне підвищення часової ефективності методу за рахунок розроблених в даній роботі оптимізацій.

В результаті проведених випробувань на предмет використання методу для задачі прогнозування та порівняння результатів з загальновідомими методами було з’ясовано, що метод є більш ефективним для задач прогнозування ніж поліноміальне прогнозування, але при цьому ненабагато поступається в ефективності методу експоненціального зглажування.

Підводячи підсумки можна сказати про досягнення поставленої мети в контексті досліджень данної роботи.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. «Самоподібність» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Self-similarity. [Дата звертання: 5.11.2021].
2. «Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems 2021 (ITTMM-2021)» [Ел. ресурс]. Available: http://ceur-ws.org/Vol-2946/. [Дата звертання: 5.11.2021]
3. «Stochastic Modeling and Applied Research of Technology» [Ел. ресурс]. Available: http://ceur-ws.org/Vol-2792/. [Дата звертання 5.11.2021]
4. «Mathematical Modeling and Scientific Computing: Focus on Complex Processes and Systems 2020» [Ел. ресурс]. Available: http://ceur-ws.org/Vol-2783/. [Дата звертання: 5.11.2021]
5. «Mathematical Modeling. Information Technology and Nanotechnology» [Ел. ресурс]. Available: http://ceur-ws.org/Vol-1904/. [Дата звертання: 5.11.2021].
6. Shynkarenko V.I., Zhadan A.A. «Constructive-synthesizing modeling of the existing time series fractal components». DNURT 2020, in print.
7. «Ковзаюча середня» [Ел. ресурс]. Available: https://ua.nesrakonk.ru/movingaverage/ . [Дата звертання: 5.11.2021].
8. «Медіанний фільтр» [Ел. ресурс]. Available: https://radioman.com.ua/viewtopic.php?t=30 . [Дата звертання: 5.11.2021].
9. «Нейромережевий аналіз з роздільною здатністю» [Ел. ресурс]. Available: https://nauka-online.com/ua/publications/informatsionnye-tehnologii/2019/6/efektivnist-vikoristannya-nejromerezhevih-modelej-dlya-prognozuvannya-ruhu-tsin-aktsij-kompanij-na-rinku . [Дата звертання: 5.11.2021].
10. «Нейромережеві багаторівневі джерела уваги» [Ел. ресурс]. Available: http://oldconf.neasmo.org.ua/node/139. [Дата звертання: 5.11.2021].
11. «Стохастичні диференціальні рівняння» [Ел. ресурс]. Available: https://uk.freejournal.org/577913/1/stokhastichne-diferentsialne-rivnyannya.html . [Дата звертання: 5.11.2021].

12. «Байєсіва мережа» [Ел. ресурс]. Available: https://uk.wikiqube.net/wiki/Bayesian_network . [Дата звертання: 5.11.2021].
13. «Багатоагентна система» [Ел. ресурс]. Available: https://uk.wikipedia.org/wiki/Багатоагентна_система . [Дата звертання: 5.11.2021].
14. «NAT» [Ел. Ресурс]. Available: https://uk.wikipedia.org/wiki/NAT [Дата звертання: 5.11.2021].
15. «AMQP» [Ел. ресурс]. Available: www.cloudamqp.com/blog/what-is-amqp-and-why-is-it-used-in-rabbitmq.html . [Дата звертання: 5.11.2021].
16. «AMQP specification» [Ел. ресурс]. Available: https://www.rabbitmq.com/protocol.html. [Дата звертання: 5.11.2021].
17. «Мова програмування C#» [Ел. ресурс]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/. [Дата звертання: 5.11.2021].
18. «Платформа .NET» [Ел. ресурс]. Available: https://docs.microsoft.com/en-us/dotnet . [Дата звертання: 5.11.2021].
19. «IP address specification» [Ел. ресурс]. Available: https://datatracker.ietf.org/doc/html/rfc5737 . [Дата звертання: 5.11.2021].
20. Shynkarenko V., Zhadan A., “Modeling of the Deterministic Fractal Time Series by One Rule Constructor”, Computer Science and Information Technologies 2020 (CSIT 2020), vol. 1, pp. 336-340, in print.
21. «L-система» [Ел. ресурс]. Available: http://www1.biologie.uni-hamburg.de/b-online/e28_3/lsys.html. [Дата звертання: 5.11.2021].
22. Shynkarenko V.I. and Ilman V.M., “Constructive-Synthesizing Structures and Their Grammatical Interpretations. Part II. Refining Transformations”. Cybernetics and Systems Analysis, 50(6), 2014, 829 – 841. doi: 10.1007/s10559-014-9674-9; “Part I. Generalized Formal Constructive-Synthesizing Structure”. Cybernetics and Systems Analysis, vol. 50(5), 2014, pp. 665 – 662. doi: 10.1007/s10559-014-9655-z
23. «Евристичні методи» [Ел. ресурс]. Available: https://pidru4niki.com/11631018/ekonomika/evristichni_metodi_zastosuvannya_ekonomichnomu_analizi . [Дата звертання: 5.11.2021].

24. «Генетичний алгоритм» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm . [Дата звертання: 5.11.2021].
25. «Масштабування обчислювальних ресурсів» [Ел. ресурс]. Available: https://azure.microsoft.com/ru-ru/overview/scaling-out-vs-scaling-up . [Дата звертання: 5.11.2021].
26. «Генетичні оператори» [Ел. ресурс]. Available: http://qai.narod.ru/GA/genoperators.html . [Дата звертання: 5.11.2021].
27. «Генетичні алгоритми» [Ел. ресурс]. Available: https://neurohive.io/ru/osnovy-data-science/chto-takoe-geneticheskie-algoritmy . [Дата звертання: 5.11.2021].
28. «Методи селекції в генетичних алгоритмах» [Ел. ресурс]. Available: http://www.aiportal.ru/articles/genetic-algorithms/methods-selection.html . [Дата звертання: 5.11.2021].
29. «Програмний агент» [Ел. ресурс]. Available: https://livepcwiki.ru/wiki/Software_agent . [Дата звертання: 5.11.2021].
30. «Класи програмних агентів» [Ел. ресурс]. Available: https://studfile.net/preview/3021420/page:2 . [Дата звертання: 5.11.2021].
31. «Пропускна здатність» [Ел. ресурс]. Available: https://altitudetvm.com/uk/komputer/1253-pengertian-bandwidth-beserta-fungsi-dan-cara-kerja-bandwidth-yang-perlu-diketahui.html . [Дата звертання: 5.11.2021].
32. «Архітектура програмного забезпечення» [Ел. ресурс]. Available: https://uk.hrvwiki.net/wiki/Software_Architecture . [Дата звертання: 5.11.2021].
33. «Програмні агенти» [Ел. ресурс]. Available: http://studbase.com/books/13/76 . [Дата звертання: 5.11.2021].
34. «Telegram API documentation» [Ел. ресурс]. Available: https://core.telegram.org/bots/api . [Дата звертання: 5.11.2021].
35. «RabbitMQ documentation» [Ел. ресурс]. Available: https://www.rabbitmq.com/documentation.html . [Дата звертання: 5.11.2021].
36. «Гексагональна архітектура» [Ел. ресурс]. Available: https://habr.com/ru/post/267125/ . [Дата звертання: 5.11.2021].

37. «Business logic in software» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Business_logic . [Дата звертання: 5.11.2021].
38. «Виродженість в генетичних алгоритмах» [Ел. ресурс]. Available: http://www.mathnet.ru/links/d44915a3286f0954954d04a583887099/pu146.pdf . [Дата звертання: 5.11.2021].
39. «Kubernetes documentation» [Ел. ресурс]. Available: https://kubernetes.io/ru/docs/home/ . [Дата звертання: 5.11.2021].
40. «Software deployment process» [Ел. ресурс]. Available: https://2wtech.com/stages-of-a-software-deployment-process/#:~:text=Software%20deployment%20process%20mainly%20consists,Deployment . [Дата звертання: 5.11.2021].
41. «Подійно-орієнтоване програмування» [Ел. ресурс]. Available: https://studfile.net/preview/7249031/page:9/ . [Дата звертання: 5.11.2021].
42. «Dependency Injection» [Ел. ресурс]. Available: https://www.tutorialsteacher.com/ioc/dependency-injection . [Дата звертання: 5.11.2021].
43. «Inversion of control» [Ел. ресурс]. Available: https://habr.com/ru/post/131993/ . [Дата звертання: 5.11.2021].
44. «SOLID principles in software development» [Ел. ресурс]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design . [Дата звертання: 5.11.2021].
45. «Зчіплення між модулями» [Ел. ресурс]. Available: https://life-prog.ru/2_89280_stseplenie-moduley.html . [Дата звертання: 5.11.2021].
46. «Зв’язність модулів» [Ел. ресурс]. Available: https://tstu.ru/book/elib3/mm/2015/loskutov/pages/2_24_zavisimosti_mezhdu_moduljami.html . [Дата звертання: 5.11.2021].
47. «Software bot» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Software_bot . [Дата звертання: 5.11.2021].
48. «Кінцевий автомат» [Ел. ресурс]. Available: https://habr.com/ru/post/358304/ . [Дата звертання: 5.11.2021].

49. «Основи тестування програмного забезпечення» [Ел. ресурс]. Available: https://qalearning.com.ua/theory/lectures/material/testing-intro/ . [Дата звертання: 5.11.2021].
50. «Центральний процесор» [Ел. ресурс]. Available: https://stud.com.ua/62391/menedzhment/tsentralniy_protsesor_mikroprotsesor . [Дата звертання: 5.11.2021].
51. «CPU-Z documentation» [Ел. ресурс]. Available: https://www.cpuid.com/softwares/cpu-z.html . [Дата звертання: 5.11.2021].
52. «Intel core processors technical resources» [Ел. ресурс]. Available: https://www.intel.com/content/www/us/en/products/docs/processors/core/core-technical-resources.html [Дата звертання: 5.11.2021].
53. «Speed test service» [Ел. ресурс]. Available: https://www.speedtest.net/insights/blog/introducing-speedtest-cli/ . [Дата звертання: 5.11.2021].
54. «DDR4 RAM» [Ел. ресурс]. Available: https://docs.opalkelly.com/ecm1900/ddr4-memory/ . [Дата звертання: 5.11.2021].
55. «Генетичні алгоритми. Локальний оптимум» [Ел. ресурс]. Available: https://cyberpedia.su/11x2a08.html . [Дата звертання: 5.11.2021].
56. «Стохастичне моделювання» [Ел. ресурс]. Available: https://ua.nesrakonk.ru/stochastic-modeling/ . [Дата звертання: 5.11.2021].
57. «Генетичні алгоритми. Фітнес функція» [Ел. ресурс]. Available: https://coderlessons.com/tutorials/akademicheskii/izuchite-geneticheskie-algoritmy/geneticheskie-algoritmy-fitnes-funktsiia . [Дата звертання: 5.11.2021].
58. «Прогнозування часових рядів» [Ел. ресурс]. Available: https://4analytics.ru/prognozirovanie/vremennie-ryadi-i-modeli-prognozirovaniya.html . [Дата звертання: 5.11.2021].
59. «Прогнозування» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Forecasting . [Дата звертання: 5.11.2021].
60. «Поліноміальна регресія» [Ел. ресурс]. Available: https://docs.infor.com/ln/10.4/ru-ru/lnolh/help/cp/onlinemanual/000111.html . [Дата звертання: 5.11.2021].

61. «Методи лінійного прогнозування» [Ел. ресурс]. Available: https://scask.ru/h_book_stv.php?id=99 . [Дата звертання: 5.11.2021].
62. «Метод експоненціального зглажування» [Ел. ресурс]. Available: https://fnow.ru/algorithm-comparison/jeksponencial-noe-sglazhivanie . [Дата звертання: 5.11.2021].
63. Закон України «Про охорону праці» прийнятий від 14 жовтня 1992 року № 2695-12;
64. ДСН 2.3.6.037-99, Державні санітарні норми виробничого шуму, ультразвуку та інфразвуку затверджені від 01.12.99 р. № 37;
65. ДСН 3.3.6.039-99 Державні санітарні норми виробничої загальної та локальної вібрації, затверджені від 01.12.99 р. № 39;
66. ДСН 3.3.6.042-99, Державні санітарні норми мікроклімату виробничих приміщень затверджені від 01.12.99 р. № 42;
67. НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207;
68. НПАОП 80.0-1.12-04 Правила безпеки під час навчання в кабінетах інформатики навчальних закладів системи загальної середньої освіти, затверджений 16.03.2004 р. № 81;
69. Інструкція щодо надання послуг з ремонту побутової радіоелектронної апаратури, затверджені від 27.08.2000 р. № 20;
70. ДСанПіН 3.3.2.007-98 Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджені від 10.12.1998 р. № 7;
71. Наказ №398 Про затвердження порядків надання домедичної допомоги особам при невідкладних станах від 16.06.2014;
72. Порядку надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою, затверджений від 16.06.2014 р. № 398;
73. НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений від 30.12.2014 Наказом № 1417

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Борис БОДНАР

МУЛЬТИАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130.01199-01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

Керівник розробки
Віктор ШИНКАРЕНКО

Виконавець
Олександр ГАЛУШКА

Нормоконтролер
Олена КУРОП'ЯТНИК

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01199 -01

МУЛЬТИАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Технічне завдання

Аркушів 21

2021

АНОТАЦІЯ

Документ 1116130.01199-01 ««Мультиагентна система моделювання фрактальних часових рядів». Технічне завдання» входить до складу програмної документації до системи для моделювання конструктивної моделі вхідного часового ряду, яка використовується для моделювання вхідного ряду та прогнозування майбутніх значень ряду.

У даному документі представлено призначення системи, область застосування програмного продукту, основні вимоги, стадії та строки розробки проекту, технічні та техніко-економічні показники, що пред’являються до програмного продукту.

ЗМІСТ

ВСТУП

В ході розробок методів моделювання та прогнозування часових рядів було винайдено новий метод, який базується на застосуванні фрактальних систем (L-систем) для моделювання часових рядів. Даний підхід використовує фрактальну природу часових рядів для відтворення його конструктивної моделі. В роботі системи використовується генетичний алгоритм для побудови конструктивної моделі. Методи еволюційних обчислень застосовуються для знаходження оптимальних рішень в задачах, в яких не можна отримати результат аналітичним шляхом. Як відомо, обчислення за допомогою генетичного алгоритму мають низьку часову ефективність, тому для підвищення часової ефективності методу було застосовано агентний підхід.

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ № 690 ст. від 18.11.2020 ректора Дніпровського національного університету залізничного транспорту імені академіка В. Лазаряна «Про затвердження керівників та затвердження тем магістерських робіт».

Тема дипломної роботи: «Засоби підвищення часової ефективності генетичних алгоритмів».

Керівник дипломної роботи – Шинкаренко В.І.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

Функціональним призначенням системи є моделювання та прогнозування заданого часового ряду.

2.2 Експлуатаційне призначення

Експлуатаційне призначення програми полягає в тому, що система являє собою інструмент для аналізу та прогнозування часових рядів. Використання системи дозволяє відтворювати конструктивну модель вхідного часового ряду для його подальшого аналізу, моделювання та прогнозування.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик системи наступні:

– відтворення конструктивної моделі вхідного часового ряду методом конструктивно-продукційного моделювання;
– використання генетичного алгоритму в процесі проведення обчислень;
– обмін повідомленнями між агентами системи використовуючи асинхронну чергу повідомлень;
– взаємодія з користувачем за допомогою текстових команд та файлів;
– конфігурація агентів в контексті мережевої взаємодії, генетичного алгоритму, процесу протоколювання.

Вхідні дані:

– файл в форматі .json з вхідними даними для обчислень;
– файли в форматі .json з конфігурацією агентів;
– текстові користувацькі команди.

Вихідні дані:

– файл .json з відтвореною конструктивної моделлю вхідного часового ряду;
– файли протоколювання процесу роботи системи;
– текстові повідомлення системи.

3.2 Вимоги до надійності

Вимоги до надійності системи наступні:

– система не повинна пошкоджувати вхідні дані в процесі своєї роботи;
– кожен агентний додаток, як складова системи не повинен допускати витоків оперативної пам’яті в процесі своєї роботи;
– кількість відмов системи не повинна перевищувати однієї відмови на 2000 запусків.

3.3 Умови експлуатації

Дане програмне забезпечення повинно використовуватись в умовах, відповідних до тих, які описані в документі [1].

Для забезпечення стабільного функціонування системи необхідно дотримуватись таких умов:

- мінімальні навички роботи з ПК;
- системою може користуватись людина, яка ознайомилася з керівництвом користувача;
- ЕОМ, на яких розгортуються агентні додатки системи, повинні відповідати чинним вимогам та стандартам в Україні, нормативних актами з охорони праці [2];
- програмний комплекс повинен використовуватись в приміщеннях, призначених для роботи ЕОМ з наступними кліматичними умовами: температура – 21-25 °С, відносна вологість повітря 40-60%.

3.4 Вимоги до складу і параметрів технічних засобів

Для роботи з додатком необхідний ПК, що має наступні мінімальні характеристики:

- процесор з тактовою частотою не менше 1.6 ГГц;
- оперативна пам’ять за об’ємом не менше ніж 2 ГБ та покоління DDR4;
- пам’ять на жорсткому диску за об’ємом не менше ніж 500 МБ;
- стабільне з’єднання з мережею Інтернет з пропускною спроможністю не менше 1 Мбіт/с.
- периферійні пристрої: VGA-монітор з мінімальним розширенням екрану 1024x768, клавіатура, миша.

3.5 Вимоги до інформаційної і програмної сумісності

Програмний продукт має бути розроблений з використанням мови програмування C#, платформи .NET та середовища розробки MS Visual Studio 2019.

Для роботи з системою необхідно щонайменше 500 МБ пам’яті для розгортання агентних додатків згідно з конфігурацією розгортання. Конфігурація розгортання системи:

– сервер RabbitMQ в кількості один;
– кластерний агент в кількості один;
– агент взаємодії з користувачем в кількості один;
– міграційний агент в кількості один;
– обчислювальні агенті в кількості від 1 до 100.

3.6 Вимоги до маркування і упаковки

У разі необхідності випуску даного програмного продукту у фізичному форматі, програмний продукт, а також електронна документація користувача повинні зберігатись на USB-носії.

Приклад маркування приведений на рис. 3.1:

Мультиагентна система моделювання фрактальних часових рядів
Галушка О.В.
ДНУЗТ Каф. КІТ
2021

Рисунок 3.1 – Приклад маркування

3.7 Вимоги до транспортування і зберігання

Транспортування повинно забезпечувати збереження програмного продукту, його цілісність та запобігання несанкціонованого доступу до нього. Транспортування фізичної упаковки програмного продукту повинно проводитись довіреною особою та здійснюватися комплектами в упаковці з термоплівки та піни, яка захищає носій від різного виду пошкоджень. Місце зберігання програмного продукту повинне бути сухим з відсутністю пилу при температурі 21-25°С і вологості 40-60%, з уникненням впливу вологи та шкідників.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація представляє собою перелік наступних документів:

– специфікація;
– текст програми;
– опис програми;
– керівництво користувача. Керівництво з моделювання фрактальних часових рядів.

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів ГОСТ 19.101-77 «Єдина система програмної документації. Види програм та програмних документів» [3].

5 ТЕХНІКО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Основна мета розробки техніко-економічного обґрунтування (ТЕО) – дати фінансову оцінку передбачуваних витрат та одержуваного корисного результату, а також оцінити прибутковість проекту і, в кінцевому підсумку, економічну доцільність його розробки та впровадження.

Початковим етапом розрахунку величини трудових витрат розробників є оцінка розміру програмного забезпечення. Основні відмінності методик, що застосовуються в оцінці трудовитрат, полягають у використовуваному типі критерію оцінки якості (кількісний або якісний).

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях.

$$E = a \cdot S^b \cdot EAF, \tag{5.1}$$

де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем $a = 2{,}4$; $b = 1{,}05$.

Припустимо, що розмір програмного коду програмного засобу – 5500 рядків:

$$E = 2{,}4 \cdot 5{,}5^{1{,}05} \cdot 1 = 14{,}3. \tag{5.2}$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 14,3 людино-місяці.

Нижче наведені розрахунки вартості розробки «Автоматизована система оцінки схожості програм». Основними статтями витрат прийняті:

– основна заробітна плата;
– відрахування на соціальні потреби;
– накладні витрати;
– витрати на персональний комп’ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) оцінює працю інженера-програміста зі створення програмного продукту і визначається виходячи з кількості розробників,

часу виконання розробки (годин), а також заробітної плати в розрахунку на одну годину. Розрахунок заробітної платні проводиться по формі табл. 5.1. [7]

Таблиця 5.1 – Фонд місячної заробітної плати

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	21135,89	1	14,3	302243,22

Описаний в проекті програмний продукт буде розроблений одним програмістом в період з 01.10.20 до 01.9.21, що складає 335 днів або 67 робочих тижнів. Витрати робочого часу прийняті за 40 годин у тиждень. Погодинна ставка кваліфікованого інженера–програміста складає 132 грн/год. Таким чином, витрачено робочого часу:

$$t_{розробки} = N_{чол} \cdot N_{тиж} \cdot N_{год}, \quad (5.3)$$

де $N_{чол}$ – кількість виконавців, *чол*;

$N_{тиж}$ – тривалість розробки;

$N_{год}$ – витрати робочого часу, год;

$$t_{розробки} = 1 \cdot 67 \cdot 40 = 2680 \text{ чол/год}, \quad (5.4)$$

ОЗП визначається за формулою:

$$ОЗП = t_{розробки} \cdot N \cdot K_{КВ}, \quad (5.5)$$

де $t_{розробки}$ – витрати праці у чол/год;

N – погодинна ставка;

$K_{КВ}$ – коефіцієнт кваліфікації програміста, приймається 0,75.

ОЗП складає:

$$ОЗП = 2680 \cdot 132 \cdot 0.75 = 265320 \text{ грн} \quad (5.6)$$

Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати:

$$C_{соц} = \frac{ОЗП \cdot 22\%}{100\%}, \quad (5.7)$$

$$C_{соц} = \frac{265320 \cdot 22\%}{100\%} = 58370,4 \text{ грн} \quad (5.8)$$

Отримані результати за (5.7) та (5.8) підсумовуються. Вони складають 229 893 грн. та визначають основні прямі витрати.

Накладні витрати враховують загальногосподарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються в процентах (30 – 40%) від суми прямих витрат:

$$C_{накл} = \frac{(ОЗП + C_{соц}) \cdot 35\%}{100\%}, \quad (5.9)$$

$$C_{накл} = \frac{(265320 + 58370,4) \cdot 35\%}{100\%} = 113291,64 \text{ грн} \quad (5.10)$$

Протягом усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Експлуатаційні витрати на персональний комп’ютер визначаються протягом терміну розробки програмного засобу в залежності від вартості комп’ютеру. В експлуатаційні витрати входять:

- вартість витратних матеріалів;
- витрати на ремонт;
- заробітна плата ремонтника;
- оренда приміщення;
- додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги;
- амортизаційні витрати на персональний комп'ютер і програмне забезпечення;
- витрати на електроенергію ($C_{ел}$), які визначаються за формулою:

$$C_{ел} = P \cdot B \cdot T_{розр} \tag{5.11}$$

де P – потужність комп'ютера та допоміжних споживачів електричної енергії, приймається 0,45 кВт/год;

B – вартість 1 кВт/година, складає 3,61 грн [8];

$T_{розр}$ – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{ел} = 0,45 \cdot 3,61 \cdot 2680 = 4353,6 \tag{5.12}$$

Витрати на витратні матеріали ($C_{ВМ}$) протягом всього терміну експлуатації приблизно 10% від вартості комп'ютеру. Вартість робочої станції приймається 25000 грн. [9], термін експлуатації – 5 років. Отже, можна визначити ці витрати за період створення програмного засобу:

$$C_{ВМ} = B_{ком} \cdot \frac{N_{Д}}{N_{експ} \cdot 365} \cdot \frac{10\%}{100\%}, \tag{5.13}$$

де $B_{ком}$ – вартість персонального комп'ютеру [9];

$N_{Д}$ – кількість днів розробки програмного продукту;

$N_{експ}$ – термін експлуатації персонального комп'ютеру.

Витрати на витратні матеріали визначаються так:

$$C_{ВМ} = 25000 \cdot \frac{335}{5 \cdot 365} \cdot \frac{10}{100} = 458,9 \tag{5.14}$$

Заробітна плата ремонтника ($C_{рем}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата приймається 18000 грн [10]. Тоді в перерахунку на один комп'ютер його заробітна плата за період розробки програмного продукту складає:

$$C_{рем} = \frac{C`_{рем}}{N_{ком}} \cdot T_{міс}, \tag{5.15}$$

де $C`_{рем}$ – середньомісячна заробітна плата;

$N_{ком}$ – кількість комп’ютерів на одного ремонтника;

$T_{міс}$ – час розробки програмного продукта, міс.

Заробітна плата ремонтника ($C_{рем}$) буде складати:

$$C_{рем} = \frac{18000}{50} \cdot 11 = 3960 \text{ грн.} \quad (5.16)$$

За статистикою витрати на комплектуючи вироби ($C_{ком}$) для ремонту персонального комп’ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{ком} = C_{ВМ} = 458,9 \text{ грн.} \quad (5.17)$$

Амортизаційні відрахування на персональний комп’ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 3 роки. Отже, за 3 роки амортизаційні відрахування на персональний комп’ютер дорівнюють вартості комп’ютера. За період проектування амортизаційні відрахування складуть:

$$АКП = B_{КОМ} \cdot \frac{N_{Д}}{N_{експ} \cdot 365}, \quad (5.18)$$

$$АКП = 25000 \cdot \frac{335}{3 \cdot 365} = 7648,4. \quad (5.19)$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для Windows 5 років та MATLAB за 1 рік то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Для функціонування персонального комп’ютера використовувалася операційна система Windows 10, для написання програмного забезпечення - програмне середовище Visual Studio Community Edition.

$$АКП_{w} = 6633 \cdot \frac{335}{5 \cdot 365} = 1217,5, \quad (5.20)$$

$$АКП_{w} = 14000 \cdot \frac{335}{3 \cdot 365} = 4283,1. \quad (5.21)$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Вартість програмного забезпечення, грн	Джерело придбання	Амортизаційні відрахування, грн
WINDOWS 10 PROFESSIONAL	6633	http://mtsoft.kiev.ua/product/windows-10-professional	1217,5
MS VISUAL STUDIO 2019	14000	https://visualstudio.microsoft.com/ru/vs/pricing/	4283,1
Всього	20633	–	5500,6

Додаткові витрати ($C_{дод}$): прибирання приміщень, охорона, комунальні послуги важко оцінити точно і прийняти рівними 50% заробітної плати інженера-програміст, тобто 10567,9 гривень на місяць.

$$C_{дод} = C_{дод}^{`} \cdot N_{міс,} \tag{5.22}$$

$$C_{дод} = 10567,9 \cdot 11 = 116246,9. \tag{5.23}$$

Оренду ($C_{ор}^{`}$) приміщень приймемо рівною 3500 гривень на місяць [11].

$$C_{ор} = C_{ор}^{`} \cdot N_{міс}, \tag{5.24}$$

$$C_{ор} = 3500 \cdot 11 = 38500. \tag{5.25}$$

Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{експ} = C_{ел} + C_{ВМ} + C_{рем} + АПК + АПЗ + C_{ор} + C_{дод,} \tag{5.26}$$

$$C_{експ} = 4353,6 + 458,9 + 3960 + 7648,4 + 5500,6 + 38500 + 116246,9 = 176668,4 \tag{5.27}$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ

Найменування витрат	Витрати, грн
Витрати на електроенергію	4353,6
Вартість витратних матеріалів	458,9
Витрати на ремонт	3960
Амортизація персонального комп'ютера	7648,4
Амортизація програмного забезпечення	5500,6
Оренда приміщення	38500
Додаткові витрати	116246,9
Всього	**176668,4**

Таким чином, витрати на створення програмного продукту складають:

$$C_{розробки} = ОЗП + C_{соц} + C_{накл} + C_{експ}, \quad (5.28)$$

$$C_{розробки} = 265320 + 58370,4 + 113291,6 + 176668,4 = 613650,4 \text{ грн.} \quad (5.29)$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	265320,0
Відрахування на соціальні потреби	58370,4
Накладні витрати	113291,6
Експлуатаційні витрати	176668,4
Всього	613650,4

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розробку системи моделювання фрактальних часових рядів. За результатами, приблизна вартість розробки складає 613650,4 грн.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадії розробки наведені в табл. 6.1.

Таблиця 6.1 – Стадії та етапи розробки

Стадії	Етапи розробки	Терміни виконання
1	2	3
1. Технічне завдання	Постановка завдання Збір початкових матеріалів Вибір і обґрунтування критеріїв ефективності і якості програми, що розробляється Обґрунтування необхідності проведення науково-дослідницьких робіт Визначення структури вхідних і вихідних даних Попередній вибір методів рішення задач Обґрунтування доцільності застосування раніше розроблених програм Визначення вимог до технічних засобів Обґрунтування принципової можливості рішення поставленої задачі Визначення вимог до програми Техніко-економічне обґрунтування розробки програми Визначення стадій, етапів і термінів розробки програми і документації на неї Вибір мов програмування Визначення необхідності проведення науково-дослідницьких робіт на подальших стадіях Узгодження і затвердження технічного завдання	25.01.2021 – 10.03.2021

Продовження таблиці 6.1

1	2	3
2. Робочий проект	Програмування і налагодження програми Розробка програмних документів відповідно до вимог ГОСТ 19.101-77 Розробка, узгодження і затвердження програми і методики випробувань Проведення попередніх і інших видів випробувань Коригування програми і програмної документації за наслідками випробувань	11.03.2021 - 24.05.2021
3. Впровадження	Підготовка і передача програми і програмної документації для супроводу і (або) виготовлення	25.05.2021 - 11.06.2021

7 ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль за виконанням роботи виконує керівник розробки.

Прийом програмного продукту здійснюється прийомною комісією і керівником розробки.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. ДСанПіН 3.3.2-007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Текст] / Постанова Головного державного санітарного лікаря України від 10 грудня 1998 р. № 7 – К.,1998.
2. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Текст]/ Постанова Головного Державного санітарного лікаря України від 01.12.1999 № 42 – К., 1999.
3. ГОСТ 19.101-77. Виды програм и программных документов [Текст]/ Постановление Государственного комитета стандартов Совета Министров СССР от 20 мая 1977 г. – М., 1977.
4. Борисенков, Евгений. "Методики оценки трудозатрат по разработке программного обеспечения информационных систем."
5. «Методики оценки трудозатрат по разработке программного обеспечения информационных систем» [Ел. ресурс]. Available: http://repository.enu.kz/bitstream/handle/data/12881/metodika-trudozatrat.pdf. [Дата звернення: 05.07.2021].
6. «Методики оценки трудозатрат» [Ел. ресурс]. Available: http://www.hups.mil.gov.ua/periodic-app/article/11953/soi_2014_8_33.pdf. [Дата звернення: 05.07.2021].
7. «Dou.ua Статистика зарплат програміств в Україні» [Ел. ресурс]. Available: https://jobs.dou.ua/salaries/#period=jun2021&city=Dnipro&title=Junior%20Software%20Engineer&language=C%23%2F.NET&spec=&exp1=0&exp2=10 [Дата звернення: 05.07.2021].
8. «Тарифы на распределение электроэнергии для предприятий» [Ел. ресурс]. Available: https://index.minfin.com.ua/tariff/electric/prom/2021-01-01 [Дата звернення: 05.07.2021].

9. «Ноутбук Lenovo ThinkBook 15 G2 ITL (20VE00FLRA) Mineral Grey» [Ел. ресурс]. Available: https://rozetka.com.ua/lenovo-20ve00flra/p322923148/ [Дата звернення: 05.07.2021].
10. «Сайт пошуку роботи Work.ua» [Ел. ресурс]. Available: https://www.work.ua/ru/salary-dnipro-%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D1%89%D0%B8%D0%BA/ [Дата звернення: 05.07.2021].
11. «Сайт оголошень OLX» [Ел. ресурс]. Available: https://www.olx.ua/d/obyavlenie/5000-uzhe-s-kommunalkoy-1-komn-r-n-gradetskiy-mebel-tehnika-IDNeda3.html#525a5dd1a3;promoted [Дата звернення: 05.07.2021].

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Борис БОДНАР

МУЛЬТИАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Робочий проект

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130.01199-01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

Керівник розробки
Віктор ШИНКАРЕНКО

Виконавець
Олександр ГАЛУШКА

Нормоконтролер
Олена КУРОП'ЯТНИК

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01199-01

МУЛЬТИАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Специфікація

Аркушів 2

2021

Позначення	Найменування	Примітка
1116130.01199-01-ЛЗ	Лист затвердження	
1116130.01199-01 12 01-ЛЗ	Лист затвердження	
1116130.01199-01 12 01	Текст програми	
1116130.01199-01 13 01-ЛЗ	Лист затвердження	
1116130.01199-01 13 01	Опис програми	
1116130.01199-01 І3 01-ЛЗ	Лист затвердження	
1116130.01199-01 І3 01	Керівництво користувача. Керівництво з моделювання фрактальних часових рядів	

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО
01116130.01199-01 13 01-ЛЗ

СИСТЕМА «МУЛЬТАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ»

Опис програми

Аркушів 18

2021

АНОТАЦІЯ

Документ 01116130.01199-01 13 01 ««Мультиагентна система моделювання фрактальних часових рядів». Опис програми» входить до складу програмної документації до системи для моделювання конструктивної моделі вхідного часового ряду, яка використовується для моделювання вхідного ряду та прогнозування майбутніх значень ряду.

У даному документі представлений опис програми системи: функціональне призначення, опис логічної структури, використані технічні засоби, виклик і завантаження, вхідні і вихідні дані, опис інтерфейсу користувача, порядок роботи з системою.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Система «мультигентна система моделювання стохастичних часових рядів» призначена для фахівців з аналізу даних та прогнозування. Система являє собою інструментарій для відтворення конструктивної моделі вхідного фрактального часового ряду.

Для розгортання агентних додатків необхідно щоб на робочих машинах була встановлена одна з наступних операційних систем:

- Windows 7;
- Windows 8;
- Windows 8.1;
- Windows 10.

Для користування системою необхідно:

- встановити додаток Telegram на смартфон або персональний комп’ютер;
- забезпечити доступ до мережі Інтернет.

Програма реалізована на мові C#, платформі .NET у програмному середовищі Visual Studio 2019.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Метою програмної системи є відтворення конструктивної моделі вхідного фрактального часового ряду.

Функціональне призначення розробки:

– відтворення конструктивної моделі вхідного часового ряду методом конструктивно-продукційного моделювання та генетичного алгоритму;
– аналіз роботи системи за допомогою протоколювання.

Відповідно до функціонального призначення були визначені наступні функціональні вимоги до системи:

– дані для обчислень які включають в себе вхідний часовий ряд та необхідну точність обчислень повинні бути представлені у вигляді JSON-файлу та мати наступний формат: config[TimeSeries, Accuracy], де TimeSeries – масив чисел з плаваючою комою, які представляють собою часовий ряд, Accuracy – число з плаваючою комою, яке представляю собою максимально допустиме середнє квадратичне відхилення між вхідним та реконструйованим рядами.
– файли протоколювання з інформацією про роботу системи повинні бути представлені у вигляді текстових файлів та описувати дії системи в процесі відтворення конструктивної моделі.
– результат роботи системи, а саме конструктивна модель вхідного фрактального часового ряду повинна бути представлена у вигляді JSON-файлу в наступному форматі: config[MathExpectation, Deviation, Fitness, CalculationTime, LSystem[Axiom, Rule][]], де MathExpectation – математичне очікування як складова конструктивної моделі, Deviation – відхилення математичного очікування як складова конструктивної моделі, Fitness – середньоквадратичне відхилення сконструйованого ряду від вхідного, CalculationTime – проміжок часу проведення процесу обчислень, LSystem – L-система, набір правил заміщення, Axiom – ліва частина правила заміщення, Rule – права частина правила заміщення.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1. Використані методи

Програма використовує наступні методи:

- агенти системи були розроблені на мові програмування C# та платформі .NET;
- обмін повідомленнями між агентами всередині системи проводиться з використанням асинхронної черги повідомлень RabbitMQ;
- обмін повідомленнями між користувачем та системою проводиться з використанням Telegram API;
- відтворення конструктивної моделі ряду проводиться з використанням методу еволюційних обчислень на базі генетичного алгоритму.

Бізнес-логіка у вигляді знань та обов’язків розподілена між кожним класом агентів. Кожен окремий клас агентів виконує свою власну роль в процесі обчислень та функціонування системи.

3.2. Алгоритм роботи системи

Після ініціювання процесу обчислень та надання системі вхідних даних система проводить наступні дії:

- кластерний агент будує конфігурацію агентів системи які будуть приймати участь у процесі обчислень;
- після успішного завершення стадії побудови конфігурації роботи проводиться почергова конфігурація кожного з обчислювальних та міграційних агентів. Для цього кластерний агент обирає налаштування роботи для кожного з агентів та відправляє їм налаштування та вхідні дані для обчислень;
- після успішного завершення стадії налаштування починається виконання обчислень за допомогою відправлення кластерним агентом команди.
- в процесі проведення обчислень кожен агент опираючись свій власний стан в контексті досягнутих результатів в обчисленнях приймає рішення про проведення міграції хромосом або завершення процесу обчислень;

– при знаходженні конструктивної моделі яка задовольняє початкові умови обчислювальний агент повідомляє про свій результат кластерного агента;
– після успішної верифікації результату кластерний агент відправляє команду повідомлення про завершення обчислень та знайдений результат у вигляді конструктивної моделі агенту користувача який в свою чергу нотифікує користувача та надає результуючу модель.

3.3. Структура системи

На рис 3.2 відображені основні сутності системи та взаємозв’язки між ними.

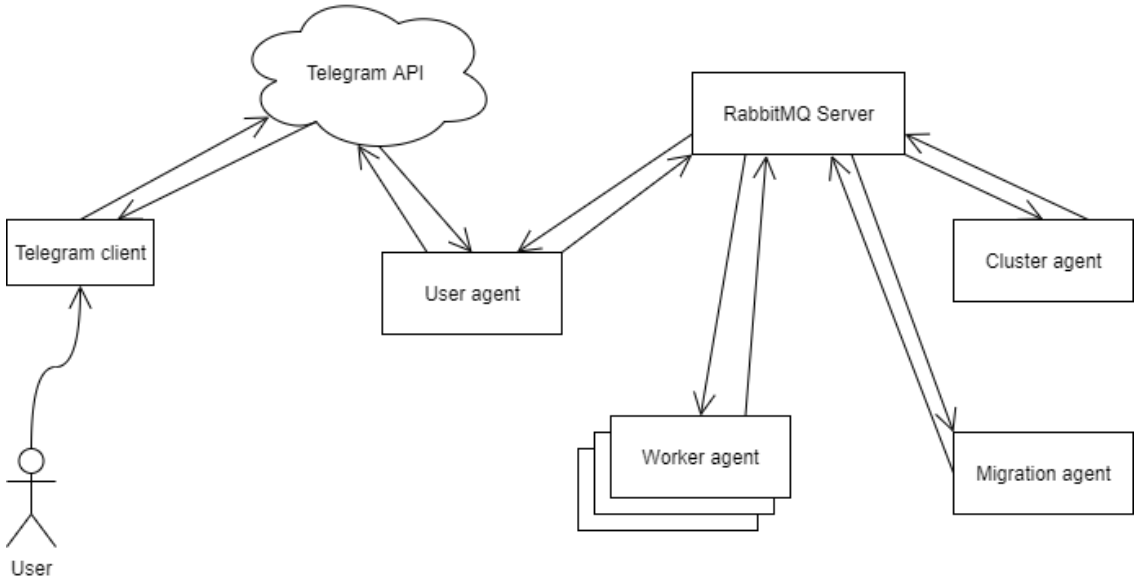


Рисунок 3.1 – Взаємозв‘язки сутностей системи

Нижче представлений опис представлених наступних сутностей:

– Telegram client;
– Telegram API;
– RabbitMQ Server;
– User Agent;
– Cluster Agent;
– Worker Agent;
– Migration Agent.

Telegram client – сторонній додаток сервісу повідомлень.

Telegram API – програмний інтерфейс платформи Telegram.

RabbitMQ Server – сервер асинхронної черги повідомлень. Використовується для обміну повідомленнями між агентами в межах системи. Сервер являє собою брокер повідомлень та зв'язуючу ланку між агентами. Для доставки повідомлень використовується мережевий протокол AMQP.

User agent – використовується для ведення діалогу з користувачем системи та делегування користувацьких команд. Даний клас агентів побудований для взаємодії користувача з системою і використовує для обміну повідомленнями як міжагентну взаємодію так і взаємодію з користувачем за допомогою Telegram API.

Cluster agent – використовується для управління групою обчислювальних агентів. Даний клас агентів являється головним агентом в кластері в обов'язки якого входить:

- взаємодія з користувачем через агента користувача від імені системи;
- інтерпретація користувацьких команд та проведення дій в системі на основі цих команд;
- ведення обліку активних агентів в системі;
- верифікація результатів обчислень.

Worker agent – використовується для проведення обчислень на базі вхідних даних. Даний клас агентів використовується для формування конструктивної моделі вхідного часового ряду за допомогою методу конструктивно-продукційного моделювання та генетичного алгоритму.

Migration agent – використовується для управління міграціями генофондів між різними обчислювальними агентами. Головним обов'язком даного класу агентів являється контроль поточних результатів обчислень обчислювальних агентів та проведення процесу міграції хромосом за допомогою запитів та відправки хромосом між обчислювальними агентами. Слідує відмітити, що спосіб проведення міграції виводиться самим міграційним агентом, обчислювальні агенти не мають знань про результати обчислень інших агентів, процес міграції в контексті обчислювального агенту проводиться за допомогою отримання та відправки хромосом за командою міграційного агенту.

3.4. Зв‘язки програми з іншими програмами

Система складається з агентів, комунікація між якими проходить за допомогою асинхронної черги повідомлень RabbitMQ. Сервер RabbitMQ використовує протокол AMQP для обміну повідомлення між клієнтами. Комунікація користувача з системою проводиться між користувачем та користувацьким агентом, за допомогою Telegram API яке працює за протоколом HTTP.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для функціонування системи необхідно розгорнути потрібну кількість агентів на робочих машинах, на всіх робочих машинах повинен бути доступ до мережі Інтернет та встановлена одна з операційних систем Windows 7, 8, 8.1, 10.

Для мінімального функціонування системи необхідно розгорнути такі агенти:

- сервер асинхронної черги повідомлень RabbitMQ в кількості 1;
- користувацький агент в кількості 1;
- кластерний агент в кількості 1;
- міграційний агент в кількості 1;
- обчислювальні агенти у кількості від 1 до 100. Слідує відмітити що кількість обчислювальних агентів може бути різної, але не рекомендується розгортувати одночасно більше 100 агентів в одному кластері через можливе перевантаження серверу RabbitMQ.

Для розгортання агентного додатку, робоча машина повинна мати наступні мінімально допустимі технічні характеристики:

- процесор з тактовою частотою не менше 800 Мгц;
- оперативна пам’ять не менше 256 Мб ОЗУ;
- жорсткий диск з об’ємом вільної пам’яті не менше 100 Мб;
- архітектура x86 або x64;
- периферійні пристрої – VGA-монітор з мінімальним розширенням екрану 1024x768, клавіатура, миша.

5 ВИКЛИК ТА ЗАВАНТАЖЕННЯ

Для початку роботи з системою необхідно створити чат с telegram ботом системи та відправити йому повідомлення «start» (команда для початку обчислень). Після цього бот запитає необхідні дані для початку обчислень і делегує їх системі, яка в свою чергу розпочне процес обчислень.

Рис. 5.1 – Екран взаємодії з системою

6 ВХІДНІ ДАНІ

Вхідними даними системи є:

1. команди користувача у текстовому форматі:
   – команда «start» – розпочати процес ініціювання процесу обчислень;
   – команда «get» – запросити поточні результати обчислень;
   – команда «stop» – завершити процес обчислень.
2. вхідні дані для обчислень - файл .json в наступному форматі: config[TimeSeries, Accuracy], де TimeSeries – масив чисел з плаваючою комою, які представляють собою часовий ряд, Accuracy – число з плаваючою комою, яке представляю собою максимально допустиме середнє квадратичне відхилення між вхідним та реконструйованим рядами.
3. конфігурація агентів – файли .json:
   – файл конфігурації мережевого з’єднання в наступному форматі: config[hostUrl, exchange-name], де hostUrl – URL сервера асинхронної черги повідомлень, exchange-name – ідентифікатор обміну повідомленнями на сервері RabbitMQ;
   – файл конфігурації ведення журналу протоколювання в наступному форматі: loggers[type, path], де type – тип логування, може приймати значення console та file;
   – файл конфігурації обчислювальних агентів.

Файл конфігурації обчислювальних агентів має наступний формат: до якого має доступ кластерний агент в наступному форматі: config[worker-configs[algorithm[initial-population-size,survive-size,crossover-chance, crossover-type, mutation-chance, mutation-type, old-chromosome-age, max-oldest-percent], fractal-system[axiom-size-max, axiom-size-min, rule-count-max, rule-count-min, rule-size-max, rule-size-min]]], де worker-configs – набір налаштувань для кожного обчислювального агенту, дане налаштування визначає стратегію та поведінку агенту в межах своїх знань та обов’язків, algorithm – налаштування генетичного алгоритму, initial-population-size – ціле позитивне число, початкова кількість хромосом в популяції, survive-size – ціле позитивне число, кількість відбору хромосом в процесі

проведення фази селекції генетичного алгоритму, crossover-chance – число з плаваючою комою від 0 до 1, вірогідність проведення схрещування при огляді хромосоми, crossover-type – ціле число в діапазоні від 0 до 4, спосіб проведення схрещування, mutation-chance – число з плаваючою комою від 0 до 1, вірогідність проведення мутації при огляді хромосоми, mutation-type – ціле число від 0 до 4, спосіб проведення мутації над хромосомою, old-chromosome-age – ціле невід'ємне число, максимально допустимий вік хромосом при якому вони можуть залишатися в популяції, max-oldest-percent – число з плаваючою комою від 0 до 100, максимально допустимий відсоток хромосом з максимальним віком, при перевищенні даного показника проводиться фаза катаклізму популяції, fractal-system – налаштування побудови L-системи, являє собою обмеження при побудові L-систем в процесі генерації або мутації хромосом, axiom-size-max – ціле число більше нуля, максимальна довжина аксіоми, axiom-size-min – ціле число більше нуля, мінімальна довжина аксіоми, rule-count-max – ціле число більше нуля, максимальна кількість правил заміщення в L-системі, rule-count-min – ціле число більше нуля, мінімальна кількість правил в L-системі, rule-size-max – ціле число більше одиниці, максимальна довжина правила заміщення, rule-size-min – ціле число більше одиниці, мінімальна довжин правила заміщення.

7 ВИХІДНІ ДАНІ

Вихідними даними є результат обчислень та файли протоколювання роботи системи.

Конструктивна модель вхідного фрактального часового ряду представлена у вигляді JSON-файлу в наступному форматі: config[MathExpectation, Deviation, Fitness, CalculationTime, LSystem[Axiom, Rule][]], де MathExpectation – математичне очікування як складова конструктивної моделі, Deviation – відхилення математичного очікування як складова конструктивної моделі, Fitness – середньоквадратичне відхилення сконструйованого ряду від вхідного, CalculationTime – проміжок часу проведення процесу обчислень, LSystem – L-система як складова частина конструктивної моделі ряду, являє собою набір правил заміщення, Axiom – ліва частина правила заміщення, Rule – права частина правила заміщення.

Файли протоколювання системи представлені у вигляді текстових файлів з записами про роботу системи, системний шлях до файлів задається в налаштуваннях механізму протоколювання.

8 ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

8.1 Опис станів програми

Після розгортання та запуску агентних додатків системи необхідно відкрити чат с ботом системи та почати роботу.

Стани програми наведено у табл. 8.1.

Таблиця 8.1 – Стани програми

№ стану	Назва стану	Опис стану	Рекомендовані дії
1	Запит на початок обчислень	Користувацький запит до системи на початок обчислень.	Відправити команду “start”.
2	Запит вхідних даних від користувача	Система запитує у користувача вхідні дані для обчислень	Відправити файл формату json з такими вхідними даними.
3	Початок процесу обчислень	Система ініціює процес обчислень та нотифікує про це користувача	Чекати завершення процесу обчислень
4	Запит проміжних результатів обчислень	Користувач запитує систему про кращий результат обчислень на даний момент	Відправити команду “get”
5	Інформування користувача про кращий результат обчислень на даний момент	Система відправляє звіт з поточними результатами обчислень (процес обчислень продовжується)	Очікувати повідомлення від системи з даними про проміжні результати обчислень
6	Інформування користувача про завершення процесу обчислень	Система відправляє конструктивну модель ряду, середнє квадратичне відхилення між вхідним та змодельованим рядом.	Очікувати повідомлення з результатами обчислень.
7	Припинення процесу обчислень	Користувач відправляє команду зупинення обчислень, система припиняє обчислення та відправляє користувачу поточний результат.	Відправити команду “stop”

8.2 Опис переходів між станами програми

Схема переходів програми наведена на рис. 8.1. Вийти з програми можливо під час усіх станів.

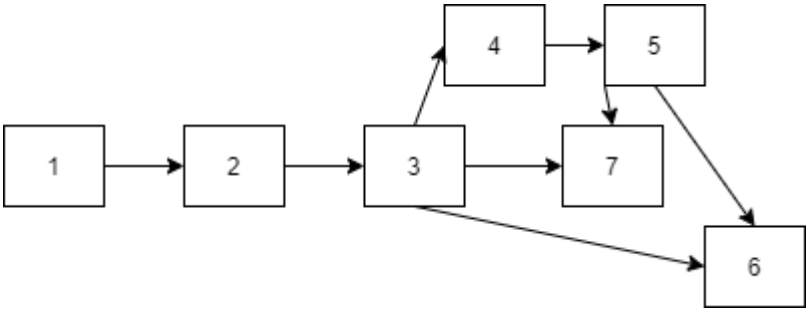


Рисунок 8.1 – Схема переходів між станами програми

8.3 Опис керування діалогом

Діалог між користувачем та додатком відбувається за допомогою інтерфейсу користувача, а саме користувач взаємодіє з системою за допомогою текстових команд та відправкою файлів формату JSON.

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Система спроектована для моделювання фрактальних моделей часових рядів. Порядок дій при роботі з системою наведено нижче:

– відкрийте додаток Telegram та перейдіть в чат з ботом системи під назвою «GneticBot»;
– в текстове поле повідомлення введіть та відправте текстову команду «start» для початку обчислень. Після отримання команди система верифікує свій стан на предмет відсутності обчислювальних процесів на даний момент. При невдалій верифікації система сповіщує користувача за допомогою повідомлення про неможливість почати новий процес обчислень;
– система запитає вхідні дані для обчислень за допомогою повідомлення «Provide source data». Відправте файл в форматі JSON, який представляє собою наступну форму: data[TimeSeries, Accuracy], де TimeSeries – масив чисел з плаваючою комою, що представляє собою вхідний фрактальний часовий ряд, Accuracy – максимально допустиме середньоквадратичне відхилення вхідного та реконструйованого рядів;
– для перегляду поточного результату введіть та відправте текстову команду «get» в чат з ботом. Після цього система відправить поточний результат обчислень в чат у вигляді тексту;
– при отриманні результату який відповідає заданим початковим умовам, система відправить конструктивну модель користувачу та завершить процес обчислень;
– для завершення процесу обчислень введіть та відправте текстову команду «stop» в чат. Після цього система відправить кращий результат обчислень якого вдалось досягти та припинить процес обчислень.

10 ПОВІДОМЛЕННЯ

У табл. 10.1 наведені повідомлення користувачу, що можуть з’явитися під час роботи з програмою.

Таблиця 10.1 – Повідомлення користувачеві

Текст повідомлення	Опис ситуації	Рекомендовані дії
Неправильний формат даних	Користувач відправив вхідні дані в неправильному форматі	Перевірити вхідні дані та привести їх в формат який потребує система
Надайте вхідні дані	Система запитує вхідні дані в користувача	Відправити файл з вхідними даними
Поточний результат	Система відправила поточний результат обчислень на запит користувача	Завантажити файл з результатами
Результат знайдений	Система знайшла оптимальне рішення задачі в процесі обчислень	Завантажити файл з результатами
Немає активних процесів обчислення	Користувач спробував зупинити або отримати поточний результат обчислень коли процес обчислень не був ініційований	Перевірити вірність команди
Процес обчислень запущений	Система прийняла запит від користувача та вхідні дані і розпочала процес обчислень	Очікувати повідомлення про завершення обчислень

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01199-01 І3 01

МУЛЬТИАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Керівництво користувача. Керівництво з моделювання фрактальних часових рядів

Аркушів 7

2021

АНОТАЦІЯ

Документ 1116130.01199-01 І3 01 ««мультиагентна система моделювання фрактальних часових рядів». Керівництво користувача. Керівництво з моделювання фрактальних часових рядів» входить до складу програмної документації до системи для моделювання конструктивної моделі вхідного часового ряду, яка використовується для моделювання вхідного ряду та прогнозування майбутніх значень ряду.

У даному документі представлено призначення та умови застосування програмного продукту, інструкції з підготовки до роботи, а також опис основних операцій.

ЗМІСТ

ВСТУП

Програмна система «мультиагентна система моделювання фрактальних часових рядів» використовується фахівцями з аналізу даних для моделювання та прогнозування часових рядів детермінованої та стохастичної природи.

Система надає можливість відтворення конструктивної моделі вхідного часового ряду для його подальшого аналізу та прогнозування.

Для використання системи необхідні мінімальні навички роботи з ПК та операційною системою Windows 7 або вище.

Для коректної роботи з системою необхідно ознайомитися з описом програми та керівництвом викладача.

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1 Функціональне призначення програми

До функціональних можливостей програмного додатку входить:

- відтворення конструктивної моделі вхідного часового ряду за допомогою генетичного алгоритму;
- надання результату відтворення конструктивної моделі ряду у вигляді файлу формату JSON.

Основне призначення системи полягає в автоматизації процесу моделювання часових рядів.

1.2 Вимоги до складу і параметрів технічних засобів

Для коректного функціонування кожного з агентних додатків системи достатньо мати робочий персональний комп’ютер, що має наступні технічні характеристики:

- процесор з тактовою частотою 1.6 Ггц;
- не менше 2 Гб ОЗУ покоління DDR4;
- 500 Мб вільного місця на жорсткому диску;
- архітектура x86 або x64;
- периферійні пристрої – VGA-монітор з мінімальним розширенням екрану 1024x768, клавіатура, миша.

1.3 Вимоги до інформаційної і програмної сумісності

Для коректного функціонування системи необхідно щоб на кожній робочій машині для розгортання агентного додатку було встановлено операційну систему Windows 7 або вище.

2 ПІДГОТОВКА ДО РОБОТИ

Перед початком використання системи необхідно розгорнути агентні додатки як складові частини системи та сервер асинхронної черги повідомлень RabbitMQ згідно з необхідною конфігурацією. Мінімальна конфігурація розгортання системи:

- сервер RabbitMQ в кількості один;
- кластерний агент в кількості один;
- міграційний агент в кількості один;
- обчислювальний агент в кількості від 1 до 100.

Для розгортання агентного додатку на робочій машині необхідно редагувати файл конфігурації агенту який поставляється разом з виконавчим файлом, надаючи необхідні налаштування агенту.

Запуск агентів системи виконується шляхом подвійного натискання лівою кнопкою миші по виконавчому файлу «agent.exe», який знаходиться у папці куди було встановлено програмний додаток.

Для роботи з програмним додатком необхідно відкрити додаток Telegram та ввести ім’я боту в поле пошуку, ім’я боту є одним з налаштувань агенту взаємодії з користувачем. Після знаходження боту в необхідно перейти в чат з ним шляхом натискання на профіль бота в списку знайдених профілів.

Для початку обчислень відправте текстову команду «start» в чат з ботом. Після отримання даної команди агент взаємодії з користувачем запросить вхідні дані для обчислень шляхом відправки текстового повідомлення. Після цього відправте файл формату JSON з вхідними даними в форматі dara[Accuracy, TimeSeries], де Accuracy – число з плаваючою комою, максимально допустиме середньоквадратичне відхилення вхідного та реконструйованого рядів, TimeSeries – масив чисел з плаваючою комою, що являють собою значення вхідного ряду. Після успішного ініціювання процесу обчислень агент проінформує користувача про початок роботи над відтворенням конструктивної моделі вхідного часового ряду.

3 ПОРЯДОК РОБОТИ З СИСТЕМОЮ

Сформуйте файл JSON з вхідними даними в наступному форматі: data[TimeSeries, Accuracy], де TimeSeries – масив чисел з плаваючою комою, що являю собою значення вхідного часового ряду, Accuracy – число з плаваючою комою, що являє собою максимально допустиме середньоквадратичне відхилення між вхідним та реконструйованим рядами (точнічсть реконструювання). Приклад файлу з вхідними даними зображено на рис. 3.1.

```
{
  "TimeSeries": [
    10.528,
    -0.068,
    19.861,
    9.544,
    -0.194,
    -10.083,
    10.002,
    -0.442,
    20.156,
    10.015,
    30.456,
    19.708
  ],
  "Accuracy": 100
}
```

Рисунок 3.1 – приклад файлу JSON з вхідними даними.

Відкрийте додаток Telegram та перейдіть в чат з ботом під назвою GneticBot. виконайте команду «start» для початку обчислень, після запиту вхідних даних у вигляді текстового повідомлення від імені системи відправте файл в форматі JSON з вхідними даними. Після відправки файлу система перевірить його на валідність та розпочне обчислення. Після успішної ініціалізації процесу обчислень система проінформує користувача про початок роботи над моделюванням вхідного часового ряду. Ініціювання процесу обчислень за допомогою інтерфейсу користувача зображено на рис. 3.2.

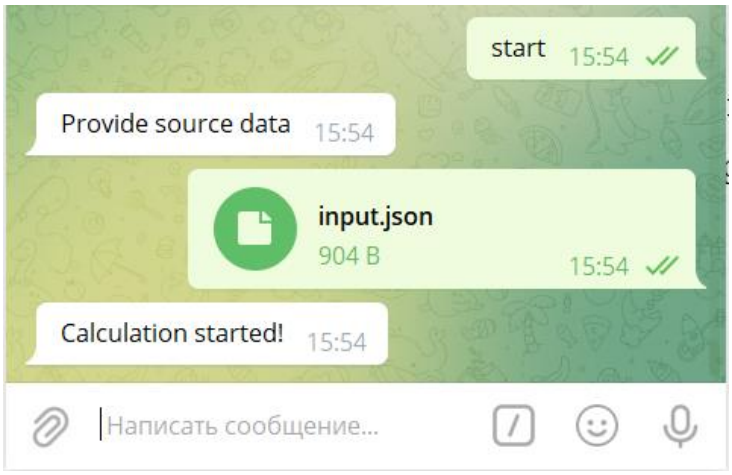


Рисунок 3.2 – початок процесу обчислень за допомогою інтерфейсу користувача.

Виконайте команду «get» для отримання поточного результату, після отримання команди система відправить файл фомарту JSON з поточним результатом у вигляді конструктивної моделі. Запит поточного результату на рис 3.3.

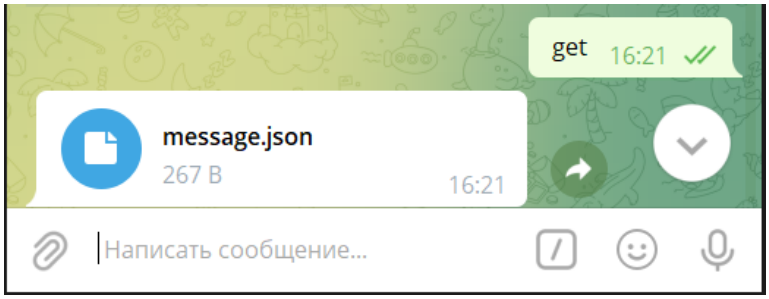


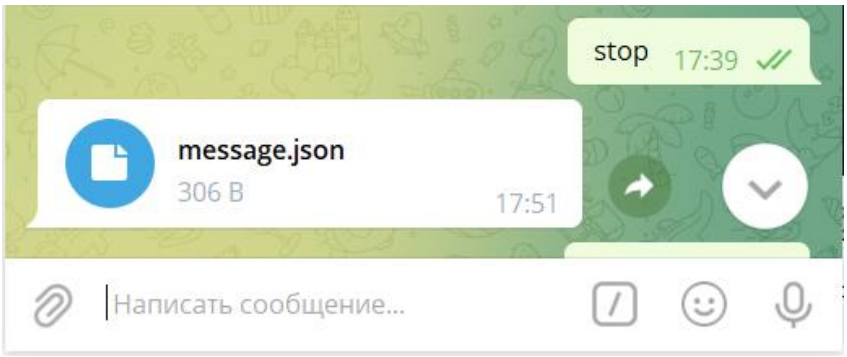
Рисунок 3.3 – запит поточного результату обчислень за допомогою інтерфейсу користувача.

Приклад JSON файлу з конструктивною моделлю зображено на рис. 3.4.


```
{
  "MathExpectation": 59.999576378613511,
  "Deviation": 59.782002102482124,
  "Fitness": 108.30221747606045,
  "LSystem": [
    {
      "Axiom": "s",
      "Production": "f"
    },
    {
      "Axiom": "f",
      "Production": "f-++f-f-f+f"
    }
  ],
  "CalculationTime": "00:02:15.1584531"
}
```

Рисунок 3.4 – приклад JSON файлу з поточним результатом обчислень.

Виконайте команду «stop» для завершення процесу обчислень, після отримання даної команди система відправить файл формату JSON з кращим результатом обчислень якого вдалось досягти за відведений час проведення обчислень. Припинення процесу обчислень за допомогою інтерфейсу користувача зображено на рис. 3.5.



На рис. 3.6 зображено приклад JSON файлу з результатом обчислень у вигляді конструктивної моделі.

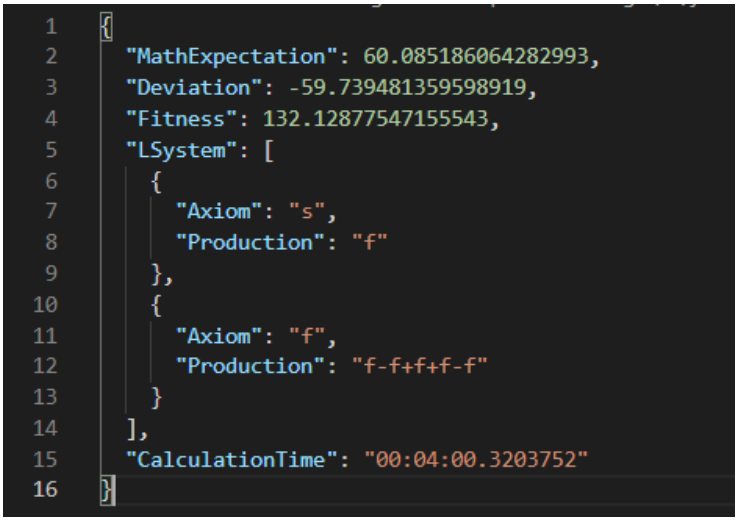


Рисунок 3.6 – приклад JSON файлу з результатом обчислень.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО

1116130.01199 -01 12 01

МУЛЬТИАГЕНТНА СИСТЕМА МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Текст програми

Аркушів 58

2021

АНОТАЦІЯ

Документ 1116130.01199-01 12 01 ««мультиагентна система моделювання фрактальних часових рядів». Текст програми» входить до складу програмної документації до системи для моделювання конструктивної моделі вхідного часового ряду, яка використовується для моделювання вхідного ряду та прогнозування майбутніх значень ряду.

В документі міститься текст програми. Програма реалізована на мові C# у програмному середовищі MS Visual Studio.

ЗМІСТ

1 СТРУКТУРА ПРОГРАМИ

Проект мультиагентної системи складається з чотирьох класів агентів, кожен з яких функціонує на рівні логіки.

Для рівня представлення використовується додаток Telegram за допомогою якого користувач встановлює зв’язок з агентом взаємодії з користувачем. Агент користувача використовує Telegram Bot API для взаємодії з користувачем.

Рівень логіки системи відповідає за відтворення конструктивної моделі вхідного фрактального часового ряду за допомогою генетичного алгоритму. Нижче наведено обов’язки кожного класу агентів в контексті системи.

Таблиця 1.1 – Обов’язки класів агентів системи

Клас агентів	Обов’язки
Агент взаємодії з користувачем	Взаємодія з користувачем, валідація команд, делегування команд користувача, делегування запитів кластерного агенту користувачеві.
Кластерний агент	Облік активних агентів в системі, конфігурація обчислювальних та міграційних агентів, ініціювання та завершення процесу обчислень, валідація результатів обчислень, контроль та управління складом системи.
Міграційний агент	Контроль та управління процесом міграції генофондів обчислювальних агентів.
Обчислювальний агент	Проведення обчислень за допомогою генетичного алгоритму

Нижче наведено діаграму компонентів рівня логіки (рис. 1.1).

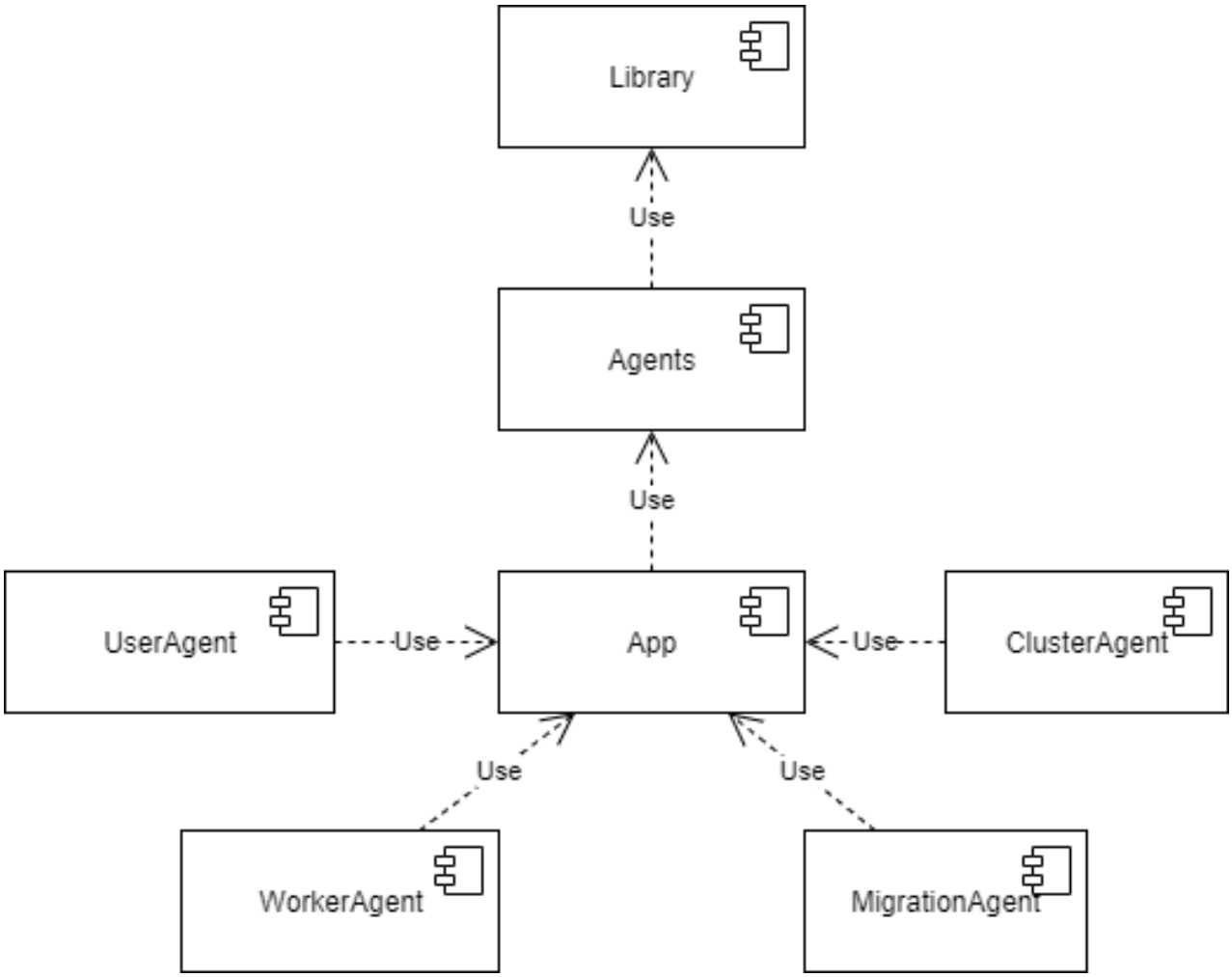


Рисунок 1.1 – Діаграма компонентів для рівня логіки

Проектна структура проекту (рис. 1.2).

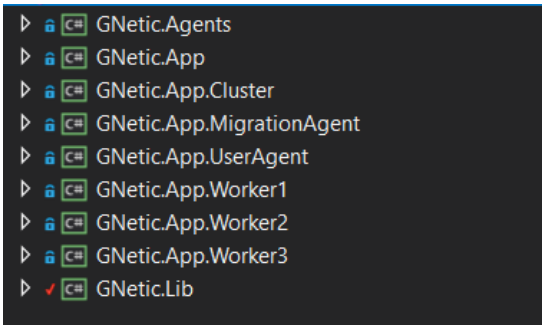


Рисунок 1.2 – Проектна структура проекту програмного продукту

Наведемо опис кожного проекту зі структури:

GNetic.Agents – проект рівня логіки, вміщає в собі бізнес-логіку для кожного з класів агентів системи.

GNetic.App – проект рівня логіки, вміщає в собі логіку ініціалізації та конфігурування агентних додатків для кожного з класів агентів.

GNetic.App.Cluster – проект рівня логіки, являє собою готовий для розгортання та запуску агентний додаток для класу кластерних агентів.

GNetic.App.MigrationAgent – проект рівня логіки, являє собою готовий для розгортання та запуску додаток для класу агентів міграції.

GNetic.App.UserAgent – проект рівня логіки, являє собою готовий для розгортання та запуску додаток для класу агентів взаємодії з користувачем.

GNetic.App.Worker1, GNetic.App.Worker2, GNetic.App.Worker3 – проекти рівня логіки, являє собою готовий для розгортання та запуску додаток для класу обчислювальних агентів.

GNetic.Lib – проект рівня логіки, вміщає в собі логіку з відтворення конструктивної моделі вхідного фрактальнго часового ряду, а саме функціонал роботи генетичного алгоритму та конструктор L-систем та всі зв’язані базові операції.

2 ТЕКСТ ПРОГРАМ

1.1 Текст модуля GNetic.Agents

AmqpConfig.cs

```
using Newtonsoft.Json;

namespace GNetic.Agents.agent.communication.amqp
{
    [JsonObject]
    public class AmqpConfig
    {
        [JsonProperty("isLocalHost")]
        public bool IsLocalHost { get; set; }
        [JsonProperty("hostUrl")]
        public string HostUrl { get; set; }
        [JsonProperty("exchange-name")]
        public string ExchangeName { get; set; }
    }
}
```

AmqpPeer.cs

```
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.agent.communication.events;
using GNetic.Lib;
using LightInject;
using Newtonsoft.Json;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;

namespace GNetic.Agents.agent.communication
{
    public interface IAmqpPeer :
IConfigurable<AmqpConfig>
    {
        string QueueId { get; }
        void Subscribe<TRequest, TResponse>();
        void Subscribe<TEvent>()
            where TEvent : IIdentityEvent;
        void Publish<TEvent>(TEvent evt, string
receiverId)
            where TEvent : IIdentityEvent;
        Task<Response<TResponse>> Request<TRequest,
TResponse>(string receiverId)
            where TRequest : new();
        Task<Response<TResponse>> Request<TRequest,
TResponse>(TRequest payload, string receiverId);
        void AssertPublisher(string publisherId);
        void RejectPublisher(string publisherId);
        void Run();
        void Stop();
    }

    public class AmqpPeer : IAmqpPeer
    {
        private string HostUrl { get; set; }
        private bool IsLocalhost { get; set; }
        private string ExchangeName { get; set; }
        private IConnection Connection { get; set;
}
        private IModel Channel { get; set; }
        public string QueueId { get; private set; }
        public string CallbackQueueId { get;
private set; }
        private readonly IServiceContainer
ServiceContainer;

        private event
EventHandler<BasicDeliverEventArgs>
MessageReceived;
        private event
EventHandler<BasicDeliverEventArgs>
ResponseReceived;

        public AmqpPeer(IServiceContainer
container)
        {
            ServiceContainer = container;
        }

        public void Run()
        {
            ConnectionFactory factory = null;
            if (IsLocalhost)
                factory = new ConnectionFactory {
HostName = "localhost" };
            else
                factory = new ConnectionFactory {
Uri = new Uri(HostUrl) };

            Connection =
factory.CreateConnection();
            Channel = Connection.CreateModel();
            QueueId =
Channel.QueueDeclare().QueueName;
            CallbackQueueId =
Channel.QueueDeclare().QueueName;

            Channel.ExchangeDeclare(ExchangeName,
ExchangeType.Direct);
            EventingBasicConsumer consumer = new
EventingBasicConsumer(Channel);
            consumer.Received += (o, e) =>
MessageReceived.Invoke(o, e);
            Channel.BasicConsume(QueueId, true,
consumer);

            EventingBasicConsumer callbackConsumer
= new EventingBasicConsumer(Channel);
            callbackConsumer.Received += (o, e) =>
ResponseReceived.Invoke(o, e);
            Channel.BasicConsume(CallbackQueueId,
true, callbackConsumer);

        }

        public void Stop()
        {
```

```
            if (Channel != null)
                Channel.Close();
            if (Connection != null)
                Connection.Close();
            QueueId = string.Empty;
            HostUrl = string.Empty;
            IsLocalhost = default;
            ExchangeName = string.Empty;
        }

        public void AssertPublisher(string
publisherId) =>
            Channel.QueueBind(QueueId,
ExchangeName, publisherId);
        public void RejectPublisher(string
publisherId) =>
            Channel.QueueUnbind(QueueId,
ExchangeName, publisherId);
        public void Subscribe<TEvent>()
            where TEvent : IIdentityEvent
        {
            Console.WriteLine($"Subscribe to EVENT:
{typeof(TEvent).Name}");
            MessageReceived += (o, e) =>
            {
                if (e.BasicProperties.AppId ==
QueueId)
                    return;
                if (e.BasicProperties.Type ==
typeof(TEvent).Name)
                {

IEnumerable<IEventHandler<TEvent>> handlers =
ServiceContainer.GetAllInstances<IEventHandler<TEve
nt>>()
                        .GroupBy(h =>
h.GetType().Name)
                        .Select(g => g.First())
                        .ToArray();
                    string json =
Encoding.UTF8.GetString(e.Body.ToArray());
                    TEvent evt =
JsonConvert.DeserializeObject<TEvent>(json);
                    foreach(var handler in
handlers)
                        handler.Handle(evt);
                }
            };
        }

        public void Subscribe<TRequest,
TResponse>()
        {
            Console.WriteLine($"Subscribe to
REQUEST: req: {typeof(TRequest).Name}, res:
{typeof(TResponse).Name}");
            MessageReceived += async (o, e) =>
            {
                if (e.BasicProperties.AppId ==
QueueId)
                    return;
                if (e.BasicProperties.ReplyTo ==
string.Empty || e.BasicProperties.CorrelationId ==
string.Empty)
                    return;
                if (e.BasicProperties.Type !=
typeof(TRequest).Name)
                    return;

                string json =
Encoding.UTF8.GetString(e.Body.ToArray());
                Request<TRequest> request =
JsonConvert.DeserializeObject<Request<TRequest>>(js
on);
                try
                {
                    IRequestHandler<TRequest,
TResponse> handler =
ServiceContainer.GetInstance<IRequestHandler<TReque
st, TResponse>>();
                    Console.WriteLine($"Request
{typeof(TRequest).Name} arrived");
                    Response<TResponse> result =
await handler.Handle(request);
                    result.SenderId = QueueId;

                    string responseJson =
JsonConvert.SerializeObject(result);
                    byte[] responseBody =
Encoding.UTF8.GetBytes(responseJson);

                    IBasicProperties props =
Channel.CreateBasicProperties();
                    props.CorrelationId =
e.BasicProperties.CorrelationId;
                    props.Type =
typeof(TResponse).Name;
                    Console.WriteLine($"Sending
response {typeof(TResponse).Name} to
{e.BasicProperties.ReplyTo}");
                    Channel.BasicPublish("",
e.BasicProperties.ReplyTo, props, responseBody);
                } catch (Exception ex)
                {
                    // not able to handle request
properly
                    // reasons:
                    // 1. Listener does not exist
                    // 2. Exception was thrown
inside handler
                    Response<TResponse>
failedResponse = new Response<TResponse>
                    {
                        Payload = default,
                        SenderId = QueueId,
                        Status =
ResponseResultStatus.Failure,
                        Message = ex.Message
                    };
                    string responseJson =
JsonConvert.SerializeObject(failedResponse);
                    byte[] responseBytes =
Encoding.UTF8.GetBytes(responseJson);

                    IBasicProperties props =
Channel.CreateBasicProperties();
                    props.CorrelationId =
e.BasicProperties.CorrelationId;
                    props.Type =
typeof(TResponse).Name;
                    Channel.BasicPublish("",
e.BasicProperties.ReplyTo, props, responseBytes);
                }
            };
```

```
        }

        public async Task<Response<TResponse>>
Request<TRequest, TResponse>(string receiverId)
            where TRequest : new()
        {
            return await Request<TRequest,
TResponse>(new TRequest(), receiverId);
        }

        public Task<Response<TResponse>>
Request<TRequest, TResponse>(TRequest payload,
string receiverId)
        {

TaskCompletionSource<Response<TResponse>> source =
new TaskCompletionSource<Response<TResponse>>();

            Request<TRequest> request = new
Request<TRequest>
            {
                SenderId = QueueId,
                Payload = payload
            };

            IBasicProperties props =
Channel.CreateBasicProperties();
            props.AppId = QueueId;
            props.ReplyTo = CallbackQueueId;
            props.CorrelationId =
Guid.NewGuid().ToString();
            props.Type = typeof(TRequest).Name;

            string requestJson =
JsonConvert.SerializeObject(request);
            byte[] requestBytes =
Encoding.UTF8.GetBytes(requestJson);

            Timer timer = new Timer();
            timer.Interval = 60000;
            timer.Elapsed += (o, e) =>
            {
                timer.Stop();
                timer.Dispose();
                Response<TResponse> failedResponse
= new Response<TResponse>()
                {
                    SenderId = string.Empty,
                    Payload = default,
                    Status =
ResponseResultStatus.NotReachable
                };
                source.SetException(new
Exception("Request does not reach to
receiver..."));
            };

            EventHandler<BasicDeliverEventArgs>
responseHandler = null;
            responseHandler = (o, e) =>
            {
                if (e.BasicProperties.CorrelationId
!= props.CorrelationId)
                    return;
                timer.Stop();
                timer.Dispose();
                Console.WriteLine($"Response
{typeof(TResponse).Name} received");
                string responseJson =
Encoding.UTF8.GetString(e.Body.ToArray());
                Response<TResponse> response =
JsonConvert.DeserializeObject<Response<TResponse>>(
responseJson);
                if (response.Status ==
ResponseResultStatus.Failure)
                    source.SetException(new
Exception($"Request failure. {response.Message}"));
                else
                    source.SetResult(response);
                ResponseReceived -=
responseHandler;
            };
            ResponseReceived += responseHandler;
            Console.WriteLine($"Send request
{typeof(TRequest).Name} to ${receiverId}");
            Channel.BasicPublish("", receiverId,
props, requestBytes);
            timer.Start();

            return source.Task;
        }

        public void Publish<TEvent>(TEvent evt,
string receiverId)
            where TEvent : IIdentityEvent
        {
            IBasicProperties props =
Channel.CreateBasicProperties();
            props.Type = typeof(TEvent).Name;
            props.AppId = QueueId;
            evt.SenderId = QueueId;
            byte[] payload =
Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(
evt));
            Channel.BasicPublish(ExchangeName,
receiverId, props, payload);
        }

        public void Configure(AmqpConfig config)
        {
            HostUrl = config.HostUrl;
            IsLocalhost = config.IsLocalHost;
            ExchangeName = config.ExchangeName;
        }
    }
}
```

AgentActivationEvent.cs

```

namespace
GNetic.Agents.agent.communication.events.models
{
    public class AgentActivationEvent :
IdentityEvent
    {
        public AgentType AgentType { get; set; }
        public AgentActivationEvent(AgentType
agentType)
        {
            AgentType = agentType;
        }
    }
```

```
}
```

AgentDeactivationEvent.cs

```
namespace
GNetic.Agents.agent.communication.events.models
{
    public class AgentDeactivationEvent :
IdentityEvent
    {
        public AgentType AgentType { get; set; }
        public AgentDeactivationEvent(AgentType
agentType)
        {
            AgentType = agentType;
        }
    }
}
```

ClusterMomentResult.cs

```
using GNetic.Lib.dtos;
namespace
GNetic.Agents.agent.communication.events.models
{
    public class ClusterMomentResultRequest
    {

    }

    public class ClusterMomentResultResponse
    {
        public SolutionDto Solution { get; set; }
    }
}
```

ClusterStartTask.cs

```
using GNetic.Lib.dtos;


namespace
GNetic.Agents.agent.communication.events.models
{
    public class ClusterStartTaskRequest
    {
        public TaskDto Task { get; set; }
    }

    public class ClusterStartTaskResponse
    {

    }
}
```

ClusterStopTask.cs

```
namespace
GNetic.Agents.agent.communication.events.models
{
    public class ClusterStopTaskRequest
    {
    }

    public class ClusterStopTaskResponse
    {

    }
}
```

MigrationMomentResult.cs

```
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace
GNetic.Agents.agent.communication.events.models
{
    public class MigrationMomentResultRequest
    {
    }

    public class MigrationMomentResultResponse
    {
        public SolutionDto Solution { get; set; }
    }
}
```

MigrationStartTask.cs

```
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class MigrationMomentResultRequest
    {
    }

    public class MigrationMomentResultResponse
    {
        public SolutionDto Solution { get; set; }
    }
}
```

MigrationStopTask.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class MigrationStopTaskRequest
    {
    }

    public class MigrationStopTaskResponse
    {

    }
}
```

PullChromosome.cs

```
using GNetic.Lib.gnetic;
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class PullChromosomeRequest
    {
    }

    public class PullChromosomeResponse
    {
        public IEnumerable<Chromosome> Chromosomes
{ get; set; }
    }
}
```

PullChromosomeRequiredEvent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class PullChromosomeRequiredEvent :
IdentityEvent
    {

    }
}
```

PushChromosome.cs

```
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class PushChromosomeRequest
    {
        public Chromosome Chromosome { get; set; }
    }

    public class PushChromosomeResponse
    {

    }
}
```

PushChromosomeRequiredEvent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class PushChromosomeRequiredEvent :
IdentityEvent
    {
    }
}
```

SolutionFound.cs

```
using GNetic.Lib.dtos;


namespace
GNetic.Agents.agent.communication.events.models
{
    public class SolutionFoundRequest
    {
        public SolutionDto Solution { get; set; }
    }

    public class SolutionFoundResponse
    {

    }
}
```

UserMomentResult.cs

```
namespace
GNetic.Agents.agent.communication.events.models
{
    public class UserMomentResultRequest
    {

    }

    public class UserMomentResultResponse
    {

    }
}
```

UserStartTask.cs

```
using GNetic.Lib.dtos;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class UserStartTaskRequest
    {
        public TaskDto Task { get; set; }
    }

    public class UserStartTaskResponse
    {

    }
}
```

UserStopTask.cs

```
namespace
GNetic.Agents.agent.communication.events.models
{
    public class UserStopTaskRequest
    {
    }
```

```
    public class UserStopTaskResponse
    {
    }
}
```

WorkerStartTask.cs

```
using GNetic.Lib.dtos;

namespace
GNetic.Agents.agent.communication.events.models
{
    public class WorkerStartTaskRequest
    {
        public WorkerTaskDto Task { get; set; }
    }

    public class WorkerStartTaskResponse
    {

    }
}
```

WorkerStopTask.cs

```
namespace
GNetic.Agents.agent.communication.events.models
{
    public class WorkerStopTaskRequest { }
    public class WorkerStopTaskResponse { }
}
```

Events.cs

```
using System.Threading.Tasks;

namespace GNetic.Agents.agent.communication.events
{
    public interface IEvent
    {
    }
    public class Event : IEvent
    {
    }
    public interface IIdentityEvent : IEvent
    {
        string SenderId { get; set; }
    }
    public class IdentityEvent : Event,
IIdentityEvent
    {
        public string SenderId { get; set; }
    }

    public enum ResponseResultStatus
    {
        Ok,
        Failure,
        NotReachable,
    }

    public interface IResponse<TPayload>
    {
        string SenderId { get; set; }
        TPayload Payload { get; set; }
    }

    public class Response<TPayload> :
IResponse<TPayload>
    {
        public string SenderId { get; set; }
        public ResponseResultStatus Status { get;
set; }
        public string Message { get; set; }
        public TPayload Payload { get; set; }

        public static Response<TPayload> Ok(string
message = "")
        {
            return new Response<TPayload>
            {
                Status = ResponseResultStatus.Ok,
                Message = message,
                Payload = default
            };
        }

        public static Response<TPayload>
Ok(TPayload payload, string message = "")
        {
            return new Response<TPayload>
            {
                Status = ResponseResultStatus.Ok,
                Message = message,
                Payload = payload
            };
        }

        public static Response<TPayload>
Fail(string message)
        {
            return new Response<TPayload>()
            {
                Status =
ResponseResultStatus.Failure,
                Message = message,
                Payload = default
            };
        }
    }

    public interface IRequest<TPayload>
    {
        string SenderId { get; set; }
        TPayload Payload { get; set; }
    }

    public class Request<TPayload> :
IRequest<TPayload>
    {
        public string SenderId { get; set; }
        public TPayload Payload { get; set; }
    }

    public interface IEventHandler<TEvent>
        where TEvent : IEvent
    {
        void Handle(TEvent evt);
    }

    public interface IRequestHandler<TRequest,
TResponse>
    {
        Task<Response<TResponse>>
Handle(Request<TRequest> request);
    }
}
```

TelegramConfig.cs

```
using Newtonsoft.Json;


namespace
GNetic.Agents.agent.communication.telegram
{
    [JsonObject]
    public class TelegramConfig
    {
        [JsonProperty("bot-token")]
        public string Token { get; set; }
        [JsonProperty("authorized-username")]
        public int AuthorizedUsername { get; set; }
    }
}
```

TelegramPeer.cs

```
using GNetic.Agents.agent.communication.events;
using GNetic.Agents.agent.communication.telegram;
using GNetic.Lib;
using GNetic.Lib.common;
using LightInject;
using Newtonsoft.Json;
using Newtonsoft.Json.Schema.Generation;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Args;
using Telegram.Bot.Types;
using Telegram.Bot.Types.InputFiles;

namespace GNetic.Agents.agent.communication
{

    public interface ITelegramPeer
    {
        void Configure(TelegramConfig config);
        void Send<TMessage>(TMessage message,
string messageText = "")
            where TMessage : class;
        Task<TResponse>
RequestUser<TResponse>(RequestUserModel model);
        void Run();
        void Stop();
    }
    public class TelegramPeer : ITelegramPeer,
IConfigurable<TelegramConfig>
    {
        private string Token { get; set; }
        private int UserName { get; set; }
        private TelegramBotClient Bot { get; set; }
        private IServiceContainer ServiceContainer
{ get; set; }
        private bool
IsUserInteractionPipelineInProcess { get; set; }
        public TelegramPeer(IServiceContainer
container)
        {
            ServiceContainer = container;
        }
        public void Run()
        {
            Bot = new TelegramBotClient(Token);
            Bot.OnMessage += async (o, e) =>
            {
                try
                {
                    if (e.Message.Chat.Id !=
UserName)
                        throw new Exception("You
are not allowed to use system!");
                    if
(IsUserInteractionPipelineInProcess)
                        return;

                    ITelegramPipelineHandler
handler =
ServiceContainer.GetAllInstances<ITelegramPipelineH
andler>()
                        .GroupBy(h =>
h.GetType().Name)
                        .Select(g => g.First())
                        .ToArray()
                        .First();


IsUserInteractionPipelineInProcess = true;
                    string response =  await
handler.HandleMessageCommand(e.Message.Text);

IsUserInteractionPipelineInProcess = false;
                    await
Bot.SendTextMessageAsync(e.Message.Chat.Id,
response);
                } catch (Exception ex)
                {
                    await
Bot.SendTextMessageAsync(e.Message.Chat.Id,
ex.Message);

IsUserInteractionPipelineInProcess = false;
                }
            };
            Bot.StartReceiving();
        }
        public void Stop()
        {
            Bot.StopReceiving();
            Bot = null;
            Token = null;
            UserName = 0;
        }

        public void Configure(TelegramConfig
config)
        {
            Token = config.Token;
            UserName = config.AuthorizedUsername;
        }

        public async Task<TResponse>
RequestUser<TResponse>(RequestUserModel model)
        {
            if (model.Type ==
TelegramUserResponseType.Boolean &&
typeof(TResponse).Name != typeof(bool).Name)
                throw new Exception("Boolean user
request must be a bool");
```

```
            if (
                model.Type ==
TelegramUserResponseType.Command &&
                (typeof(TResponse).Name !=
typeof(string).Name ||
model.ExpectedCommands.Length == 0)
               )
                throw new Exception("Command user
request must contains ExpectedCommand property and
response type must be a string");

            ChatId chatId = new ChatId(UserName);

            TaskCompletionSource<TResponse> source
= new TaskCompletionSource<TResponse>();
            EventHandler<MessageEventArgs> handler
= null;
            handler = async (o, e) =>
            {
                try
                {
                    switch (model.Type)
                    {
                        case
TelegramUserResponseType.Boolean:
                            string answer =
e.Message.Text.ToUpper();
                            if (answer != "YES" &&
answer != "NO")
                                throw new
Exception("Expected Yes or No.");

source.SetResult((TResponse)Convert.ChangeType(answ
er == "YES" ? true : false, TypeCode.Boolean));
                            break;
                        case
TelegramUserResponseType.Command:
                            if
(!model.ExpectedCommands.Contains(e.Message.Text))
                                throw new
Exception($"Expected one of commands:
{string.Join(",", model.ExpectedCommands)}.");

source.SetResult((TResponse)Convert.ChangeType(e.Me
ssage.Text, TypeCode.String));
                            break;
                        case
TelegramUserResponseType.Json:
                            if (e.Message.Document
== null || e.Message.Document.MimeType !=
"application/json")
                                throw new
Exception("Expected json file!");
                            MemoryStream blob = new
MemoryStream();
                            Telegram.Bot.Types.File
file = await
Bot.GetInfoAndDownloadFileAsync(e.Message.Document.
FileId, blob);
                            string json =
Encoding.UTF8.GetString(blob.ToArray());
                            try
                            {
                                TResponse response
= JsonConvert.DeserializeObject<TResponse>(json);

source.SetResult(response);
                            } catch (Exception)
                            {
                                JSchemaGenerator
generator = new JSchemaGenerator();
                                string schema =
generator.Generate(typeof(TResponse)).ToString();
                                throw new
Exception($"Invalid json schema! Expected schema:
{schema}.");
                            }
                            break;
                    }
                    Bot.OnMessage -= handler;
                } catch (Exception ex)
                {
                    await
Bot.SendTextMessageAsync(chatId, $"{ex.Message} Try
again!");
                }
            };
            Bot.OnMessage += handler;
            await Bot.SendTextMessageAsync(chatId,
model.Message);
            return await source.Task;
        }

        public async void Send<TMessage>(TMessage
message, string messageText = "")
            where TMessage : class
        {
            ChatId chatId = new ChatId(UserName);
            if (messageText != string.Empty)
                await
Bot.SendTextMessageAsync(chatId, messageText);
            if (message != null)
            {
                string json = message.ToJson();
                MemoryStream fileStream = new
MemoryStream(Encoding.UTF8.GetBytes(json));
                InputOnlineFile file = new
InputOnlineFile(fileStream, "message.json");
                await Bot.SendDocumentAsync(chatId,
file);
            }
        }
    }
}
```

UserInteraction.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telegram.Bot.Types;

namespace
GNetic.Agents.agent.communication.telegram
{
    public enum TelegramUserResponseType
    {
        Command,
        Json,
```

```
        Boolean
    }


    public class RequestUserModel
    {
        public TelegramUserResponseType Type { get;
set; }
        public string[] ExpectedCommands { get;
set; }
        public string Message { get; set; }
    }

    public interface ITelegramPipelineHandler
    {
        Task<string> HandleMessageCommand(string
command);
        Task<string> HandleStartProcess();
        Task<string> HandleStopProcess();
        Task<string> HandleGetResult();
    }


}
```

ActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.agent.layers
{
    public interface
IActuatorLayer<TCommunicationPeer>
    {
        TCommunicationPeer Endpoint { get; set; }
    }
    public class ActuatorLayer<TCommunicationPeer>
: IActuatorLayer<TCommunicationPeer>
    {
        public TCommunicationPeer Endpoint { get;
set; }
        public ActuatorLayer(TCommunicationPeer
peer)
        {
            Endpoint = peer;
        }
    }

}
```

EnvironmentActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.agent.layers
{
    public interface IEnvironmentActuatorLayer
    {
        void SendActivationSignal();
        void SendDeactivationSignal();
    }
    public class EnvironmentActuatorLayer :
ActuatorLayer<IAmqpPeer>, IEnvironmentActuatorLayer
    {
        private readonly IAgentContext Context;
        public
EnvironmentActuatorLayer(IAgentContext context,
IAmqpPeer peer) : base(peer)
        {
            Context = context;
        }

        public void SendActivationSignal()
        {
            AgentActivationEvent ev = new
AgentActivationEvent(Context.AgentInfo.AgentType);
            Endpoint.Publish(ev, "all");
        }

        public void SendDeactivationSignal()
        {
            AgentDeactivationEvent ev = new
AgentDeactivationEvent(Context.AgentInfo.AgentType)
;
            Endpoint.Publish(ev, "all");
        }
    }
}
```

EnvironmentSensorLayer.cs

```
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Lib.logger;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.agent.layers
{
    public class EnvironmentSensorLayer :
        IEventHandler<AgentActivationEvent>,
        IEventHandler<AgentDeactivationEvent>
    {
        private readonly IAgentContext Context;
        private readonly IAmqpPeer Peer;
        private readonly IEnvironmentActuatorLayer
Actuator;
        private readonly ILogger Logger;
        public EnvironmentSensorLayer(IAgentContext
context, IAmqpPeer peer, IEnvironmentActuatorLayer
actuator, ILogger logger)
        {
            Context = context;
            Peer = peer;
            Actuator = actuator;
            Logger = logger;
        }

        public virtual void
Handle(AgentActivationEvent evt)
```

```
        {
            AgentInfo agent =
Context.RemoteAgents.FirstOrDefault(a =>
                a.AgentId == evt.SenderId &&
a.AgentType == evt.AgentType
            );
            if (agent != null)
                return;
            agent = new AgentInfo(evt.SenderId,
evt.AgentType);
            Context.RemoteAgents.Add(agent);
            Logger.Log($"Agent added: Id:
{evt.SenderId} Type: {evt.AgentType}");
            Peer.AssertPublisher(evt.SenderId);
            Actuator.SendActivationSignal();
        }

        public virtual void
Handle(AgentDeactivationEvent evt)
        {
            Logger.Log($"Agent removed: Id:
{evt.SenderId} Type: {evt.AgentType}");
            Context.RemoteAgents = (
                from agent in Context.RemoteAgents
                where agent.AgentId != evt.SenderId
&& agent.AgentType != evt.AgentType
                select agent
            ).ToList();
            Peer.RejectPublisher(evt.SenderId);
        }

    }
}
```

Agent.cs

```
using LightInject;


namespace GNetic.Agents.agent
{

    public interface IAgent
    {
        void
Configure<TStartup>(IAgentInitialConfiguration
configuration)
            where TStartup : IAgentStartup, new();
        void Run();
        void Stop();
    }

    public abstract class Agent : IAgent
    {
        protected IServiceContainer Container;
        public virtual void
Configure<TStartup>(IAgentInitialConfiguration
configuration)
            where TStartup : IAgentStartup, new()
        {
            Container = new
ServiceContainer(options => options.EnableVariance
= true);
            TStartup startup = new TStartup();
            startup.Configure(Container,
configuration);
        }
        public abstract void Run();
        public abstract void Stop();
    }
}
```

AgentContext.cs

```
using System.Collections.Generic;


namespace GNetic.Agents.agent
{
    public interface IAgentContext
    {
        AgentInfo AgentInfo { get; set; }
        List<AgentInfo> RemoteAgents { get; set; }
    }
    public class AgentContext : IAgentContext
    {
        public AgentInfo AgentInfo { get; set; }
        public List<AgentInfo> RemoteAgents { get;
set; } = new List<AgentInfo>();
    }
}
```

AgentInfo.cs

```
using System;

namespace GNetic.Agents.agent
{
    [Serializable]
    public class AgentInfo
    {
        public string AgentId { get; set; }
        public AgentType AgentType { get; set; }
        public AgentInfo(string agentId, AgentType
agentType)
        {
            AgentId = agentId;
            AgentType = agentType;
        }
    }
}
```

AgentInitialConfiguration.cs

```
using System.Collections.Generic;

namespace GNetic.Agents.agent
{
    public interface IAgentInitialConfiguration
    {
        void SetConfig<TConfig>(TConfig config)
            where TConfig : class;
        TConfig GetConfig<TConfig>()
            where TConfig : class;
    }
    public class AgentInitialConfiguration :
Dictionary<string, object>,
IAgentInitialConfiguration
    {
        public TConfig GetConfig<TConfig>()
            where TConfig : class =>
            (TConfig)this[typeof(TConfig).Name];
```

```
        public void SetConfig<TConfig>(TConfig
config)
            where TConfig : class =>
            Add(config.GetType().Name, config);
    }
}
```

AgentStartup.cs

```
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.logger;
using LightInject;

namespace GNetic.Agents.agent
{
    public interface IAgentStartup
    {
        void Configure(IServiceContainer container,
IAgentInitialConfiguration configuration);
    }

    public class AgentStartup : IAgentStartup
    {
        public virtual void
Configure(IServiceContainer container,
IAgentInitialConfiguration configuration)
        {
            container.RegisterInstance(container);
            container.RegisterSingleton<IAmqpPeer,
AmqpPeer>();
            container.RegisterSingleton<ILogger,
Logger>();
            container.RegisterMultiple<

IEventHandler<AgentActivationEvent>,

IEventHandler<AgentDeactivationEvent>,
                EnvironmentSensorLayer>();
            // navigation in agents environment

container.RegisterSingleton<IEnvironmentActuatorLay
er, EnvironmentActuatorLayer>();
        }


    }
}
```

AgentType.cs

```
namespace GNetic.Agents.agent
{
    public enum AgentType
    {
        WorkerAgent,
        ClusterAgent,
        MigrationAgent,
        UserAgent
    }
}
```

ClusterManagingLayer.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.ClusterAgent.models;
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.layers
{
    public interface IClusterManagingLayer
    {
        Task HandleStartCalculationRequest(TaskDto
task);
        Task HandleStopCalculationRequest();
        Task<SolutionDto> GetMomentSolution();
        Task HandleSolutionFoundRequest(SolutionDto
solution);
    }
    public class ClusterManagingLayer :
IClusterManagingLayer
    {
        private readonly IClusterAgentContext
Context;
        private readonly IWorkerActuatorLayer
WorkerActuator;
        private readonly
IMigrationAgentActuatorLayer MigrationActuator;
        private readonly IUserAgentActuatorLayer
UserAgentActuator;
        public ClusterManagingLayer(
            IClusterAgentContext context,
            IWorkerActuatorLayer workerActuator,
            IMigrationAgentActuatorLayer
migrationActuator,
            IUserAgentActuatorLayer
userAgentActuator
        )
        {
            Context = context;
            WorkerActuator = workerActuator;
            MigrationActuator = migrationActuator;
            UserAgentActuator = userAgentActuator;
        }

        public async Task<SolutionDto>
GetMomentSolution()
        {
            var migrator = GetMigrator();
            var solution = await
MigrationActuator.RequestMomentResult(migrator.Agen
tId);
            return solution;
        }

        public async Task
HandleSolutionFoundRequest(SolutionDto solution)
        {
            await
UserAgentActuator.ProvideFoundSolution(solution);
            await HandleStopCalculationRequest();
        }

        public async Task
HandleStartCalculationRequest(TaskDto task)
        {
            if (Context.DistributedAlgorithmState
!= null &&
Context.DistributedAlgorithmState.InProcess)
```

```
                throw new Exception("Algorithm is
already in process");

            Context.DistributedAlgorithmState = new
DistributedAlgorithmState
            {
                InProcess = true,
                IsCompleted = false,
                WorkerUnits = new
List<WorkerUnitState>()
            };

            var migrator = GetMigrator();
            await
MigrationActuator.RequestStart(task,
migrator.AgentId);

            var workers =
Context.RemoteAgents.Where(agent => agent.AgentType
== AgentType.WorkerAgent).ToList();
            var workerConfigs =
Context.AgentsConfiguration.WorkerConfigs;
            for (int i = 0; i < workers.Count; i++)
            {
                WorkerConfig config =
workerConfigs.ElementAt(i % workerConfigs.Length);
                WorkerUnitState workerState = new
WorkerUnitState
                {
                    WorkerId = workers[i].AgentId,
                    Settings = config
                };
                await
WorkerActuator.RequestWorkerStart(workerState,
task);

Context.DistributedAlgorithmState.WorkerUnits.Add(w
orkerState);
            }
        }

        public async Task
HandleStopCalculationRequest()
        {
            if (Context.DistributedAlgorithmState
== null ||
!Context.DistributedAlgorithmState.InProcess)
                throw new Exception("Algorithm does
not in process");
            var migrator = GetMigrator();
            await
MigrationActuator.RequestStop(migrator.AgentId);
            var units =
Context.DistributedAlgorithmState.WorkerUnits;
            foreach(var unit in units)
                await
WorkerActuator.RequestWorkerStop(unit.WorkerId);
            Context.DistributedAlgorithmState =
null;
        }



        private AgentInfo GetMigrator()
        {
            var migrator =
Context.RemoteAgents.FirstOrDefault(a =>
a.AgentType == AgentType.MigrationAgent);
            if (migrator == null)
                throw new Exception("Migrator does
not exist");
            return migrator;
        }
    }
}
```

MigrationAgentActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.dtos;
using GNetic.Lib.logger;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.layers
{
    public interface IMigrationAgentActuatorLayer
    {
        Task RequestStart(TaskDto task, string
migratorId);
        Task RequestStop(string migratorId);
        Task<SolutionDto>
RequestMomentResult(string migratorId);
    }
    public class MigrationAgentActuatorLayer :
ActuatorLayer<IAmqpPeer>,
IMigrationAgentActuatorLayer
    {
        private readonly ILogger Logger;
        public
MigrationAgentActuatorLayer(IAmqpPeer amqp, ILogger
logger) : base(amqp)
        {
            Logger = logger;
        }
        public async Task<SolutionDto>
RequestMomentResult(string migratorId)
        {
            Logger.Log($"Request to migrator for
current best solution, migratorId: {migratorId}");
            var response = await
Endpoint.Request<MigrationMomentResultRequest,
MigrationMomentResultResponse>(migratorId);
            return response.Payload.Solution;
        }

        public async Task RequestStart(TaskDto
task, string migratorId)
        {
            var req = new MigrationStartTaskRequest
            {
                Task = task
            };
            await
Endpoint.Request<MigrationStartTaskRequest,
MigrationStartTaskResponse>(req, migratorId);
        }

        public async Task RequestStop(string
migratorId)
```

```
        {
            await
Endpoint.Request<MigrationStopTaskRequest,
MigrationStopTaskResponse>(migratorId);
        }
    }
}
```

MigrationAgentSensorLayer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;

namespace GNetic.Agents.ClusterAgent.layers
{
    public class MigrationAgentSensorLayer :
        IRequestHandler<SolutionFoundRequest,
SolutionFoundResponse>
    {
        private readonly IClusterManagingLayer
Cluster;
        public
MigrationAgentSensorLayer(IClusterManagingLayer
cluster)
        {
            Cluster = cluster;
        }
        public async
Task<Response<SolutionFoundResponse>>
Handle(Request<SolutionFoundRequest> request)
        {
            await
Cluster.HandleSolutionFoundRequest(request.Payload.
Solution);
            return
Response<SolutionFoundResponse>.Ok();
        }
    }
}
```

UserAgentActuatorLayer.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.layers
{
    public interface IUserAgentActuatorLayer
    {
        Task ProvideFoundSolution(SolutionDto
solution);
    }
    public class UserAgentActuatorLayer :
ActuatorLayer<IAmqpPeer>, IUserAgentActuatorLayer
    {
        private readonly IAgentContext Context;
        public UserAgentActuatorLayer(IAgentContext
context, IAmqpPeer peer) : base(peer)
        {
            Context = context;
        }
        public async Task
ProvideFoundSolution(SolutionDto solution)
        {
            var userAgent = GetUserAgent();
            await
Endpoint.Request<SolutionFoundRequest,
SolutionFoundResponse>(
                new SolutionFoundRequest
                {
                    Solution = solution
                },
                userAgent.AgentId
            );
        }

        private AgentInfo GetUserAgent()
        {
            var userAgent =
Context.RemoteAgents.FirstOrDefault(a =>
a.AgentType == AgentType.UserAgent);
            if (userAgent == null)
                throw new Exception("No user agent
found");
            return userAgent;
        }
    }
}
```

UserAgentSensorLayer.cs

```
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.layers
{
    public class UserAgentSensorLayer :
        IRequestHandler<ClusterStartTaskRequest,
ClusterStartTaskResponse>,
        IRequestHandler<ClusterStopTaskRequest,
ClusterStopTaskResponse>,
        IRequestHandler<ClusterMomentResultRequest,
ClusterMomentResultResponse>
    {
        private readonly IClusterManagingLayer
Cluster;
        public
UserAgentSensorLayer(IClusterManagingLayer
clusterManaging)
        {
            Cluster = clusterManaging;
        }

        public async
Task<Response<ClusterStartTaskResponse>>
Handle(Request<ClusterStartTaskRequest> request)
        {
```

```
            await
Cluster.HandleStartCalculationRequest(request.Paylo
ad.Task);
            return
Response<ClusterStartTaskResponse>.Ok();
        }

        public async
Task<Response<ClusterStopTaskResponse>>
Handle(Request<ClusterStopTaskRequest> request)
        {
            await
Cluster.HandleStopCalculationRequest();
            return
Response<ClusterStopTaskResponse>.Ok();
        }

        public async
Task<Response<ClusterMomentResultResponse>>
Handle(Request<ClusterMomentResultRequest> request)
        {
            var solution = await
Cluster.GetMomentSolution();
            return
Response<ClusterMomentResultResponse>.Ok(
                new ClusterMomentResultResponse {
                    Solution = solution
                }
            );
        }
    }
}
```

WorkerActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Agents.ClusterAgent.models;
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.layers
{
    public interface IWorkerActuatorLayer
    {
        Task RequestWorkerStart(WorkerUnitState
worker, TaskDto task);
        Task RequestWorkerStop(string workerId);
    }
    public class WorkerActuatorLayer :
ActuatorLayer<IAmqpPeer>, IWorkerActuatorLayer
    {
        public WorkerActuatorLayer(IAmqpPeer amqp)
: base(amqp)
        {

        }

        public async Task
RequestWorkerStart(WorkerUnitState worker, TaskDto
task)
        {
            WorkerStartTaskRequest req = new
WorkerStartTaskRequest
            {
                Task = new WorkerTaskDto
                {
                    TaskDescription = task,
                    AlgorithmSettings = new
AlgorithmSettingsDto
                    {
                        CrossoveringChance =
worker.Settings.AlgorithmConfiguration.Crossovering
Chance,
                        CrossoverType =
(Lib.gnetic.crossover.CrossoverType)worker.Settings
.AlgorithmConfiguration.CrossoverType,
                        InitialPopulationSize =
worker.Settings.AlgorithmConfiguration.InitialPopul
ationSize,
                        MaxOldestPercent =
worker.Settings.AlgorithmConfiguration.MaxOldestPer
cent,
                        MutationChance =
worker.Settings.AlgorithmConfiguration.MutationChan
ce,
                        MutationType =
(Lib.gnetic.mutators.MutationType)worker.Settings.A
lgorithmConfiguration.MutationType,
                        OldestChromosomeAge =
worker.Settings.AlgorithmConfiguration.OldestAge,
                        SurviveChromosomeCount =
worker.Settings.AlgorithmConfiguration.SurviveSize
                    },
                    LSystemSettings = new
LSystemSettingsDto
                    {
                        AxiomLengthMax =
worker.Settings.LSystemConfiguration.AxiomSizeMax,
                        AxiomLengthMin =
worker.Settings.LSystemConfiguration.AxiomSizeMin,
                        RuleCountMax =
worker.Settings.LSystemConfiguration.RuleCountMax,
                        RuleCountMin =
worker.Settings.LSystemConfiguration.RuleCountMin,
                        RuleLengthMax =
worker.Settings.LSystemConfiguration.RuleSizeMax,
                        RuleLengthMin =
worker.Settings.LSystemConfiguration.RuleSizeMin
                    },
                    MathSettings = new MathSettings
                    {
                        DeviationMax =
worker.Settings.MathConfiguration.DeviationMax,
                        DeviationMin =
worker.Settings.MathConfiguration.DeviationMin,
                        MathExpectationMax =
worker.Settings.MathConfiguration.MathExpectationMa
x,
                        MathExpectationMin =
worker.Settings.MathConfiguration.MathExpectationMi
n,
                        DispersionMax =
worker.Settings.MathConfiguration.DispersionMax,
                        DispersionMin =
worker.Settings.MathConfiguration.DispersionMin
                    }
                }
            };
```

```
            await
Endpoint.Request<WorkerStartTaskRequest,
WorkerStartTaskResponse>(req, worker.WorkerId);
        }

        public async Task RequestWorkerStop(string
workerId)
        {
            await
Endpoint.Request<WorkerStopTaskRequest,
WorkerStartTaskResponse>(workerId);
        }
    }
}
```

WorkerAgentSensorLayer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.layers
{
    public class WorkerAgentSensorLayer
    {
    }
}
```

AgentsConfiguration.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.models
{
    [JsonObject]
    public class AgentsConfiguration
    {
        [JsonProperty("worker-configs")]
        public WorkerConfig[] WorkerConfigs { get;
set; }
    }
    [JsonObject]
    public class WorkerConfig
    {
        [JsonIgnore]
        public string ConfigId { get; set; } =
Guid.NewGuid().ToString();
        [JsonProperty("algorithm")]
        public AlgorithmConfigModel
AlgorithmConfiguration { get; set; }
        [JsonProperty("fractal-system")]
        public LSystemConfigModel
LSystemConfiguration { get; set; }
        [JsonProperty("math")]
        public MathConfigModel MathConfiguration {
get; set; }
    }
    [JsonObject]
    public class AlgorithmConfigModel
    {
        [JsonProperty("initial-population-size")]
        public int InitialPopulationSize { get;
set; }
        [JsonProperty("survive-size")]
        public int SurviveSize { get; set; }
        [JsonProperty("crossover-chance")]
        public int CrossoveringChance { get; set; }
        [JsonProperty("crossover-type")]
        public int CrossoverType { get; set; }
        [JsonProperty("mutation-chance")]
        public int MutationChance { get; set; }
        [JsonProperty("mutation-type")]
        public int MutationType { get; set; }
        [JsonProperty("old-chromosome-age")]
        public int OldestAge { get; set; }
        [JsonProperty("max-oldest-percent")]
        public int MaxOldestPercent { get; set; }
    }
    [JsonObject]
    public class LSystemConfigModel
    {
        [JsonProperty("axiom-size-max")]
        public int AxiomSizeMax { get; set; }
        [JsonProperty("axiom-size-min")]
        public int AxiomSizeMin { get; set; }
        [JsonProperty("rule-count-max")]
        public int RuleCountMax { get; set; }
        [JsonProperty("rule-count-min")]
        public int RuleCountMin { get; set; }
        [JsonProperty("rule-size-max")]
        public int RuleSizeMax { get; set; }
        [JsonProperty("rule-size-min")]
        public int RuleSizeMin { get; set; }
    }

    [JsonObject]
    public class MathConfigModel
    {
        [JsonProperty("math-expectation-max")]
        public double MathExpectationMax { get;
set; }
        [JsonProperty("math-expectation-min")]
        public double MathExpectationMin { get;
set; }
        [JsonProperty("deviation-max")]
        public double DeviationMax { get; set; }
        [JsonProperty("deviation-min")]
        public double DeviationMin { get; set; }
        [JsonProperty("dispersion-max")]
        public double DispersionMax { get; set; }
        [JsonProperty("dispersion-min")]
        public double DispersionMin { get; set; }
    }
}
```

DistributedAlgorithmState.cs

```
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent.models
{

    public class DistributedAlgorithmState
    {
```

```
        public bool InProcess { get; set; } =
false;
        public bool IsCompleted { get; set; } =
false;
        public List<WorkerUnitState> WorkerUnits {
get; set; } = new List<WorkerUnitState>();
    }

    public class WorkerUnitState
    {
        public string WorkerId { get; set; }
        public WorkerConfig Settings { get; set; }
    }
}
```

ClusterAgent.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LightInject;
using GNetic.Agents.agent.layers;

namespace GNetic.Agents.ClusterAgent
{
    public class ClusterAgent : Agent
    {
        public override void Run()
        {
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Run();
            amqp.AssertPublisher("all");
            IAgentContext context =
Container.GetInstance<IAgentContext>();
            context.AgentInfo = new
AgentInfo(amqp.QueueId, AgentType.ClusterAgent);
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendActivationSignal();
        }

        public override void Stop()
        {
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendDeactivationSignal();
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Stop();
        }
    }
}
```

ClusterAgentContext.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.ClusterAgent.models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent
{
    public interface IClusterAgentContext :
IAgentContext
    {
        DistributedAlgorithmState
DistributedAlgorithmState { get; set; }
        AgentsConfiguration AgentsConfiguration {
get; set; }
    }
    public class ClusterAgentContext :
AgentContext, IClusterAgentContext
    {
        public DistributedAlgorithmState
DistributedAlgorithmState { get; set; }
        public AgentsConfiguration
AgentsConfiguration { get; set; }
    }
}
```

ClusterAgentStartup.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.ClusterAgent.layers;
using GNetic.Agents.ClusterAgent.models;
using GNetic.Lib.logger;
using LightInject;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.ClusterAgent
{
    public class ClusterAgentStartup : AgentStartup
    {
        public override void
Configure(IServiceContainer container,
IAgentInitialConfiguration configuration)
        {
            base.Configure(container,
configuration);


container.RegisterMultiple<IAgentContext,
IClusterAgentContext, ClusterAgentContext>();

container.RegisterSingleton<IClusterManagingLayer,
ClusterManagingLayer>();

container.RegisterSingleton<IUserAgentActuatorLayer
, UserAgentActuatorLayer>();

container.RegisterSingleton<IWorkerActuatorLayer,
WorkerActuatorLayer>();

            container.RegisterMultiple<

IRequestHandler<ClusterStartTaskRequest,
ClusterStartTaskResponse>,
```

```
IRequestHandler<ClusterStopTaskRequest,
ClusterStopTaskResponse>,

IRequestHandler<ClusterMomentResultRequest,
ClusterMomentResultResponse>,
                UserAgentSensorLayer>();

            container.RegisterSingleton<

IRequestHandler<SolutionFoundRequest,
SolutionFoundResponse>,
                MigrationAgentSensorLayer
                >();


container.RegisterSingleton<IMigrationAgentActuator
Layer, MigrationAgentActuatorLayer>();

            IAmqpPeer amqp =
container.GetInstance<IAmqpPeer>();
            SetupAmqp(amqp, configuration);
            ILogger logger =
container.GetInstance<ILogger>();
            SetupLogger(logger, configuration);

            IClusterAgentContext context =
container.GetInstance<IClusterAgentContext>();
            context.AgentsConfiguration =
configuration.GetConfig<AgentsConfiguration>();
        }

        public void SetupAmqp(IAmqpPeer amqp,
IAgentInitialConfiguration configuration)
        {
            AmqpConfig config =
configuration.GetConfig<AmqpConfig>();
            amqp.Configure(config);
            amqp.Subscribe<AgentActivationEvent>();

amqp.Subscribe<AgentDeactivationEvent>();
            amqp.Subscribe<ClusterStartTaskRequest,
ClusterStartTaskResponse>();
            amqp.Subscribe<ClusterStopTaskRequest,
ClusterStopTaskResponse>();

amqp.Subscribe<ClusterMomentResultRequest,
ClusterMomentResultResponse>();
            amqp.Subscribe<SolutionFoundRequest,
SolutionFoundResponse>();
        }

        public void SetupLogger(ILogger logger,
IAgentInitialConfiguration configuration)
        {
            LoggerConfig config =
configuration.GetConfig<LoggerConfig>();
            logger.Configure(config);
        }


    }
}
```

ClusterActuatorLayer.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.dtos;
using GNetic.Lib.logger;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.layers
{
    public interface IClusterActuatorLayer
    {
        Task SendFoundResult(SolutionDto solution,
string clusterId);
    }
    public class ClusterActuatorLayer :
ActuatorLayer<IAmqpPeer>, IClusterActuatorLayer
    {
        private readonly ILogger Logger;
        public ClusterActuatorLayer(IAmqpPeer amqp,
ILogger logger): base(amqp)
        {
            Logger = logger;
        }
        public async Task
SendFoundResult(SolutionDto solution, string
clusterId)
        {
            Logger.Log("Send found result to
cluster");
            var req = new SolutionFoundRequest
            {
                Solution = solution
            };
            await
Endpoint.Request<SolutionFoundRequest,
SolutionFoundResponse>(req, clusterId);
        }

    }
}
```

ClusterSensorLayer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Lib.logger;

namespace GNetic.Agents.MigrationAgent.layers
{
    public class ClusterSensorLayer :

IRequestHandler<MigrationMomentResultRequest,
MigrationMomentResultResponse>,
        IRequestHandler<MigrationStartTaskRequest,
MigrationStartTaskResponse>,
        IRequestHandler<MigrationStopTaskRequest,
MigrationStopTaskResponse>
    {
```

```
        private readonly IMigrationLayer
MigrationLayer;
        private readonly ILogger Logger;
        public ClusterSensorLayer(IMigrationLayer
migrationLayer, ILogger logger)
        {
            MigrationLayer = migrationLayer;
            Logger = logger;
        }
        public async
Task<Response<MigrationMomentResultResponse>>
Handle(Request<MigrationMomentResultRequest>
request)
        {
            Logger.Log("Moment result request
arrived");
            var solution = await
MigrationLayer.GetCurrentSolution();
            var response = new
MigrationMomentResultResponse
            {
                Solution = solution
            };
            return
Response<MigrationMomentResultResponse>.Ok(response
);
        }

        public async
Task<Response<MigrationStartTaskResponse>>
Handle(Request<MigrationStartTaskRequest> request)
        {
            Logger.Log("Start calculation request
arrived");
            await
MigrationLayer.HandleStartTask(request.Payload.Task
);
            return
Response<MigrationStartTaskResponse>.Ok();
        }

        public async
Task<Response<MigrationStopTaskResponse>>
Handle(Request<MigrationStopTaskRequest> request)
        {
            Logger.Log("Stop calculation request
arrived");
            await MigrationLayer.HandleStopTask();
            return
Response<MigrationStopTaskResponse>.Ok();
        }
    }
}
```

MigrationLayer.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.MigrationAgent.models;
using GNetic.Lib.dtos;
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.layers
{
    public interface IMigrationLayer
    {
        Task HandleStartTask(TaskDto task);
        Task HandleStopTask();
        Task<IEnumerable<Chromosome>>
HandlePullRequest(string workerId);
        Task HandlePushRequest(Chromosome
chromosome, string workerId);
        Task<SolutionDto> GetCurrentSolution();

    }
    public class MigrationLayer : IMigrationLayer
    {
        private readonly IMigrationAgentContext
Context;
        private readonly IClusterActuatorLayer
Cluster;
        private readonly IWorkerActuatorLayer
WorkerActuator;
        private readonly IPollSchedulerLayer
PollScheduler;
        public MigrationLayer(
            IMigrationAgentContext context,
            IClusterActuatorLayer clusterActuator,
            IWorkerActuatorLayer workerActuator,
            IPollSchedulerLayer pollScheduler
        )
        {
            Context = context;
            Cluster = clusterActuator;
            WorkerActuator = workerActuator;
            PollScheduler = pollScheduler;
        }
        public async Task<IEnumerable<Chromosome>>
HandlePullRequest(string workerId)
        {
            var worker = GetWorker(workerId);
            if (Context.MigrationFlow == null)
                throw new Exception("No calculation
process tasks specified!");

            var currentState =
Context.MigrationFlow.Workers.Find(w => w.WorkerId
== worker.AgentId);
            var chromosomes = (
                from w in
Context.MigrationFlow.Workers
                where w.WorkerId != worker.AgentId
&&
                w.Chromosome != null &&
                currentState.Chromosome != null ?
                w.Chromosome.Fitness <
currentState.Chromosome.Fitness :
                true
                select w.Chromosome
            ).ToList();
            return chromosomes;

        }

        public async Task
HandlePushRequest(Chromosome chromosome, string
workerId)
        {
            var worker = GetWorker(workerId);
            if (Context.MigrationFlow == null)
```

```
                throw new Exception("No calculation
process tasks specified!");

            if (chromosome.Fitness <=
Context.MigrationFlow.Task.Accuracy &&
Context.MigrationFlow.Durability == null)
            {
                var cluster = GetCluster();
                Context.MigrationFlow.Durability =
DateTime.Now - Context.MigrationFlow.StartedOn;
                var solution =
SolutionDto.FromChromosome(chromosome);
                solution.CalculationTime =
Context.MigrationFlow.Durability;
                await
Cluster.SendFoundResult(solution, cluster.AgentId);
                return;
            }
            var unit =
Context.MigrationFlow.Workers.FirstOrDefault(w =>
w.WorkerId == workerId);
            if (unit.Chromosome != null &&
unit.Chromosome.Fitness < chromosome.Fitness)
                return;

            unit.Chromosome = chromosome;

            var units = from w in
Context.MigrationFlow.Workers
                        where w.Chromosome != null
? w.Chromosome.Fitness > chromosome.Fitness : true
                        select w;
            foreach (var u in units)

WorkerActuator.SendPullRequiredEvent(u.WorkerId);
        }

        public async Task HandleStartTask(TaskDto
task)
        {
            Context.MigrationFlow = new
MigrationFlow
            {
                Task = task,
                Workers = Context.RemoteAgents
                    .Where(a => a.AgentType ==
AgentType.WorkerAgent)
                    .Select(a => new
WorkerUnitAnalyticModel
                    {
                        WorkerId = a.AgentId,
                    })
                    .ToList(),
                StartedOn = DateTime.Now
            };
            await PollScheduler.BeginPollJob();
        }

        public async Task HandleStopTask()
        {
            Context.MigrationFlow = null;
            await PollScheduler.EndPollJob();
        }

        private AgentInfo GetWorker(string
workerId)
        {
            var worker =
Context.RemoteAgents.FirstOrDefault(a => a.AgentId
== workerId && a.AgentType ==
AgentType.WorkerAgent);
            if (worker == null)
                throw new Exception("Worker does
not found");
            return worker;
        }

        private AgentInfo GetCluster()
        {
            var cluster =
Context.RemoteAgents.FirstOrDefault(a =>
a.AgentType == AgentType.ClusterAgent);
            if (cluster == null)
                throw new Exception("Cluster does
not found");
            return cluster;
        }

        public async Task<SolutionDto>
GetCurrentSolution()
        {
            if (Context.MigrationFlow == null)
                throw new Exception("No calculation
process tasks specified");

            var chromosomes = from w in
Context.MigrationFlow.Workers
                              where w.Chromosome !=
null
                              orderby
w.Chromosome.Fitness
                              select w.Chromosome;
            if (chromosomes.Count() == 0)
                throw new Exception("No chromosomes
presented");

            return
SolutionDto.FromChromosome(chromosomes.First());
        }
    }
}
```

PollSchedulerLayer.cs

```
using GNetic.Agents.MigrationAgent.schedule;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.layers
{
    public interface IPollSchedulerLayer
    {
        Task BeginPollJob();
        Task EndPollJob();
    }
    public class PollSchedulerLayer :
IPollSchedulerLayer
    {
        private readonly IMigrationAgentContext
Context;
        private readonly IWorkerActuatorLayer
WorkerActuator;
```

```
        private Job PullJob { get; set; }
        private Job PushJob { get; set; }
        public PollSchedulerLayer(
            IMigrationAgentContext context,
            IWorkerActuatorLayer workerActuator
        )
        {
            Context = context;
            WorkerActuator = workerActuator;
        }

        public async Task BeginPollJob()
        {
            // TODO: move intervals to agent
settings
            PullJob = Job.StartNew(15000,
StartPullRequiredJob);
            PushJob = Job.StartNew(15000,
StartPushRequiredJob);
        }

        public async Task EndPollJob()
        {
            if (PushJob != null)
                PushJob.Stop();
            if (PullJob != null)
                PullJob.Stop();
            PushJob = null;
            PullJob = null;
        }

        private void StartPushRequiredJob()
        {
            var flow = Context.MigrationFlow;
            if (flow == null)
                return;
            var units = flow.Workers;
            foreach (var unit in units)

WorkerActuator.SendPushRequiredEvent(unit.WorkerId)
;
        }

        private void StartPullRequiredJob()
        {
            var flow = Context.MigrationFlow;
            if (flow == null)
                return;
            var units = flow.Workers;
            foreach(var unit in units)
            {
                var betterUnits = from w in units
                                  where w.WorkerId
!= unit.WorkerId &&

w.Chromosome != null &&

unit.Chromosome != null ?

w.Chromosome.Fitness < unit.Chromosome.Fitness :
                                                 true
                                  select w;
                if (betterUnits.Count() != 0)

WorkerActuator.SendPullRequiredEvent(unit.WorkerId)
;
            }
        }
    }
}
```

WorkerActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.logger;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.layers
{
    public interface IWorkerActuatorLayer
    {
        Task SendPushRequiredEvent(string
workerId);
        Task SendPullRequiredEvent(string
workerId);

    }
    public class WorkerActuatorLayer :
ActuatorLayer<IAmqpPeer>, IWorkerActuatorLayer
    {
        private readonly ILogger Logger;
        public WorkerActuatorLayer(IAmqpPeer amqp,
ILogger logger) :base(amqp)
        {
            Logger = logger;
        }

        public async Task
SendPullRequiredEvent(string workerId)
        {
            Logger.Log($"Sending pull required
event to worker, workerId: {workerId}");
            var evt = new
PullChromosomeRequiredEvent();
            Endpoint.Publish(evt, workerId);
        }

        public async Task
SendPushRequiredEvent(string workerId)
        {
            Logger.Log($"Sending push required
event to worker, workerId: {workerId}");
            var evt = new
PushChromosomeRequiredEvent();
            Endpoint.Publish(evt, workerId);
        }
    }
}
```

WorkerSensorLayer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
```

```
using GNetic.Lib.logger;

namespace GNetic.Agents.MigrationAgent.layers
{
    public class WorkerSensorLayer :
        IRequestHandler<PullChromosomeRequest,
PullChromosomeResponse>,
        IRequestHandler<PushChromosomeRequest,
PushChromosomeResponse>
    {
        private readonly IMigrationLayer
MigrationLayer;
        private readonly ILogger Logger;
        public WorkerSensorLayer(IMigrationLayer
migrationLayer, ILogger logger)
        {
            MigrationLayer = migrationLayer;
            Logger = logger;
        }
        public async
Task<Response<PullChromosomeResponse>>
Handle(Request<PullChromosomeRequest> request)
        {
            Logger.Log($"Pull request arrived:
{request.SenderId}");
            var chromosomes = await
MigrationLayer.HandlePullRequest(request.SenderId);
            var response = new
PullChromosomeResponse
            {
                Chromosomes = chromosomes
            };
            return
Response<PullChromosomeResponse>.Ok(response);
        }

        public async
Task<Response<PushChromosomeResponse>>
Handle(Request<PushChromosomeRequest> request)
        {
            Logger.Log($"Push request arrived:
{request.SenderId}");
            await
MigrationLayer.HandlePushRequest(request.Payload.Ch
romosome, request.SenderId);
            return
Response<PushChromosomeResponse>.Ok();
        }
    }
}
```

MigrationFlow.cs

```
using GNetic.Lib.dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.models
{
    public class MigrationFlow
    {
        public TaskDto Task { get; set; }
        public List<WorkerUnitAnalyticModel>
Workers { get; set; } = new
List<WorkerUnitAnalyticModel>();
        public DateTime? StartedOn { get; set; } =
null;
        public TimeSpan? Durability { get; set; } =
null;
    }
}
```

WorkerUnitAnalyticModel.cs

```
using GNetic.Lib.dtos;
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.models
{
    public class WorkerUnitAnalyticModel
    {
        public string WorkerId { get; set; }
        public Chromosome Chromosome { get; set; }
    }
}
```

Job.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent.schedule
{
    public class Job
    {
        public Guid JobId { get; private set; }
        public int Interval { get; private set; }
        private bool IsRunning { get; set; }
        public Job(int interval)
        {
            Interval = interval;
            JobId = Guid.NewGuid();
        }

        public void Start(Action action)
        {
            IsRunning = true;
            Task.Run(() =>
            {
                while (IsRunning)
                {
                    Thread.Sleep(Interval);
                    action();
                }
            });
        }
        public void Stop()
        {
            IsRunning = false;
        }

        public static Job StartNew(int interval,
Action action)
        {
            Job job = new Job(interval);
```

```
            job.Start(action);
            return job;
        }
    }
}
```

MigrationAgent.cs

```
using GNetic.Agents.agent;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LightInject;
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.layers;

namespace GNetic.Agents.MigrationAgent
{
    public class MigrationAgent : Agent
    {
        public override void Run()
        {
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Run();
            amqp.AssertPublisher("all");
            IAgentContext context =
Container.GetInstance<IAgentContext>();
            context.AgentInfo = new
AgentInfo(amqp.QueueId, AgentType.MigrationAgent);
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendActivationSignal();
        }

        public override void Stop()
        {
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendDeactivationSignal();
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Stop();
        }
    }
}
```

MigrationAgentContext.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.MigrationAgent.models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.MigrationAgent
{
    public interface IMigrationAgentContext :
IAgentContext
    {
        MigrationFlow MigrationFlow { get; set; }
    }

    public class MigrationAgentContext :
AgentContext, IMigrationAgentContext
    {
        public MigrationFlow MigrationFlow { get;
set; }
    }
}
```

MigrationAgentStartup.cs

```
using GNetic.Agents.agent;
using LightInject;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.MigrationAgent.layers;
using GNetic.Agents.agent.communication;
using GNetic.Lib.logger;
using GNetic.Agents.agent.communication.amqp;

namespace GNetic.Agents.MigrationAgent
{
    public class MigrationAgentStartup :
AgentStartup
    {
        public override void
Configure(IServiceContainer container,
IAgentInitialConfiguration configuration)
        {
            base.Configure(container,
configuration);

container.RegisterMultiple<IAgentContext,
IMigrationAgentContext, MigrationAgentContext>();

            // core logic

container.RegisterSingleton<IMigrationLayer,
MigrationLayer>();

container.RegisterSingleton<IPollSchedulerLayer,
PollSchedulerLayer>();
            // cluster side
            container.RegisterMultiple<

IRequestHandler<MigrationStartTaskRequest,
MigrationStartTaskResponse>,

IRequestHandler<MigrationStopTaskRequest,
MigrationStopTaskResponse>,

IRequestHandler<MigrationMomentResultRequest,
MigrationMomentResultResponse>,
                ClusterSensorLayer
                >();

container.RegisterSingleton<IClusterActuatorLayer,
ClusterActuatorLayer>();

            // worker side
            container.RegisterMultiple<
```

```
IRequestHandler<PullChromosomeRequest,
PullChromosomeResponse>,

IRequestHandler<PushChromosomeRequest,
PushChromosomeResponse>,
                WorkerSensorLayer
                >();

container.RegisterSingleton<IWorkerActuatorLayer,
WorkerActuatorLayer>();

            IAmqpPeer amqp =
container.GetInstance<IAmqpPeer>();
            SetupAmqp(amqp, configuration);
            ILogger logger =
container.GetInstance<ILogger>();
            SetupLogger(logger, configuration);
        }

        private void SetupAmqp(IAmqpPeer amqp,
IAgentInitialConfiguration configuration)
        {
            AmqpConfig config =
configuration.GetConfig<AmqpConfig>();
            amqp.Configure(config);


amqp.Subscribe<MigrationStartTaskRequest,
MigrationStartTaskResponse>();

amqp.Subscribe<MigrationStopTaskRequest,
MigrationStopTaskResponse>();

amqp.Subscribe<MigrationMomentResultRequest,
MigrationMomentResultResponse>();
            amqp.Subscribe<PullChromosomeRequest,
PullChromosomeResponse>();
            amqp.Subscribe<PushChromosomeRequest,
PushChromosomeResponse>();

            amqp.Subscribe<AgentActivationEvent>();

amqp.Subscribe<AgentDeactivationEvent>();
        }

        private void SetupLogger(ILogger logger,
IAgentInitialConfiguration configuration)
        {
            LoggerConfig config =
configuration.GetConfig<LoggerConfig>();
            logger.Configure(config);
        }
    }
}
```

ClusterActuatorLayer.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.dtos;
using GNetic.Lib.logger;
using System.Linq;
using System.Threading.Tasks;

namespace GNetic.Agents.UserAgent.layers
{
    public interface IClusterActuatorLayer
    {
        Task StartProcess(TaskDto source);
        Task StopProcess();
        Task<SolutionDto> GetMomentResult();
    }
    public class ClusterActuatorLayer :
ActuatorLayer<IAmqpPeer>, IClusterActuatorLayer
    {
        private readonly IAgentContext Context;
        private readonly ILogger Logger;
        public ClusterActuatorLayer(IAgentContext
context, IAmqpPeer peer, ILogger logger) :
base(peer)
        {
            Context = context;
            Logger = logger;
        }
        private AgentInfo GetClusterInfo()
        {
            AgentInfo clusterAgent =
Context.RemoteAgents
                .Where(a => a.AgentType ==
AgentType.ClusterAgent)
                .FirstOrDefault();
            if (clusterAgent == null)
                throw new System.Exception("No
cluster agent available");
            return clusterAgent;
        }
        public async Task StartProcess(TaskDto
source)
        {
            Logger.Log("Sending start process
request to cluster");
            var cluster = GetClusterInfo();
            ClusterStartTaskRequest req = new
ClusterStartTaskRequest
            {
                Task = source
            };
            await
Endpoint.Request<ClusterStartTaskRequest,
ClusterStartTaskResponse>(req, cluster.AgentId);
        }

        public async Task StopProcess()
        {
            Logger.Log("Sending stop process
request to cluster");
            var cluster = GetClusterInfo();
            await
Endpoint.Request<ClusterStopTaskRequest,
ClusterStopTaskResponse>(cluster.AgentId);
        }

        public async Task<SolutionDto>
GetMomentResult()
        {
            Logger.Log("Send moment result request
to cluster");
            var cluster = GetClusterInfo();
            var response = await
Endpoint.Request<ClusterMomentResultRequest,
ClusterMomentResultResponse>(cluster.AgentId);
```

```
            return response.Payload.Solution;
        }
    }
}
```

ClusterSensorLayer.cs

```
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Lib.logger;
using System.Threading.Tasks;

namespace GNetic.Agents.UserAgent.layers
{
    public class ClusterSensorLayer :
        IRequestHandler<SolutionFoundRequest,
SolutionFoundResponse>
    {
        private readonly IUserActuatorLayer
UserActuator;
        private readonly ILogger Logger;
        public
ClusterSensorLayer(IUserActuatorLayer userActuator,
ILogger logger)
        {
            UserActuator = userActuator;
            Logger = logger;
        }

        public async
Task<Response<SolutionFoundResponse>>
Handle(Request<SolutionFoundRequest> request)
        {
            Logger.Log($"Solution found request
arrived, solution: {request.Payload.Solution}");

UserActuator.SendFoundResult(request.Payload.Soluti
on);
            return
Response<SolutionFoundResponse>.Ok();
        }
    }
}
```

UserActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.communication.telegram;
using GNetic.Agents.agent.layers;
using GNetic.Lib.dtos;
using System.Threading.Tasks;

namespace GNetic.Agents.UserAgent.layers
{
    public interface IUserActuatorLayer
    {
        void SendFoundResult(SolutionDto solution);
        Task<TaskDto> GetTaskSourceData();
    }
    public class UserActuatorLayer :
ActuatorLayer<ITelegramPeer>, IUserActuatorLayer
    {
        private readonly IUserAgentContext Context;
        public UserActuatorLayer(IUserAgentContext
context, ITelegramPeer peer) : base(peer)
        {
            Context = context;
        }

        public async Task<TaskDto>
GetTaskSourceData()
        {
            var data = await
Endpoint.RequestUser<TaskDto>(new RequestUserModel
            {
                Message = "Provide source data",
                Type =
TelegramUserResponseType.Json
            });
            return data;
        }

        public void SendFoundResult(SolutionDto
solution)
        {
            Endpoint.Send(solution, "Solution
found!");
        }
    }
}
```

UserSensorLayer.cs

```
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.communication.telegram;
using GNetic.Lib.common;
using System;
using System.Threading.Tasks;

namespace GNetic.Agents.UserAgent.layers
{

    public class UserSensorLayer :
        ITelegramPipelineHandler
    {
        private readonly IClusterActuatorLayer
Cluster;
        private readonly IUserActuatorLayer
UserActuator;
        public
UserSensorLayer(IClusterActuatorLayer
clusterActuator, IUserActuatorLayer userActuator)
        {
            Cluster = clusterActuator;
            UserActuator = userActuator;
        }

        public async Task<string>
HandleMessageCommand(string command)
        {
            switch (command)
            {
                case "start":
                    return await
HandleStartProcess();
                case "stop":
```

```
                    return await
HandleStopProcess();
                case "get":
                    return await HandleGetResult();
                default:
                    return "Unrecognizeable
command!";
            }
        }

        public async Task<string> HandleGetResult()
        {
            try
            {
                var solution = await
Cluster.GetMomentResult();
                string json = solution.ToJson();
                return $"Here is a current result!
{json}";
            } catch (Exception ex)
            {
                return ex.Message;
            }
        }

        public async Task<string>
HandleStartProcess()
        {
            // request data from user
            var dto = await
UserActuator.GetTaskSourceData();
            await Cluster.StartProcess(dto);
            return "Calculation started!";
        }

        public async Task<string>
HandleStopProcess()
        {
            try
            {
                string result = await
HandleGetResult();
                await Cluster.StopProcess();
                return result;
            } catch (Exception ex)
            {
                return ex.Message;
            }

        }
    }
}
```

UserAgent.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using LightInject;
using GNetic.Agents.agent.layers;

namespace GNetic.Agents.UserAgent
{
    public class UserAgent : Agent
    {
        public override void Run()
        {
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Run();
            amqp.AssertPublisher("all");
            IAgentContext context =
Container.GetInstance<IAgentContext>();
            context.AgentInfo = new
AgentInfo(amqp.QueueId, AgentType.UserAgent);
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendActivationSignal();
            ITelegramPeer telegram =
Container.GetInstance<ITelegramPeer>();
            telegram.Run();
        }

        public override void Stop()
        {
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            ITelegramPeer telegram =
Container.GetInstance<ITelegramPeer>();
            IEnvironmentActuatorLayer actuator =
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuator.SendDeactivationSignal();
            telegram.Stop();
            amqp.Stop();
        }
    }
}
```

UserAgentContext.cs

```
using GNetic.Agents.agent;

namespace GNetic.Agents.UserAgent
{
    public interface IUserAgentContext :
IAgentContext
    {
        int UserName { get; set; }
    }
    public class UserAgentContext : AgentContext,
IUserAgentContext
    {
        public int UserName { get; set; }
    }
}
```

UserAgentStartup.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.communication.telegram;
using GNetic.Agents.UserAgent.layers;
using GNetic.Lib.logger;
using LightInject;

namespace GNetic.Agents.UserAgent
{
    public class UserAgentStartup : AgentStartup
    {
        public override void
Configure(IServiceContainer container,
IAgentInitialConfiguration configuration)
        {
```

```
            base.Configure(container,
configuration);


container.RegisterSingleton<ITelegramPeer,
TelegramPeer>();


container.RegisterSingleton<ITelegramPipelineHandle
r, UserSensorLayer>();

container.RegisterSingleton<IRequestHandler<Solutio
nFoundRequest, SolutionFoundResponse>,
ClusterSensorLayer>();


container.RegisterMultiple<IAgentContext,
IUserAgentContext, UserAgentContext>();

container.RegisterSingleton<IClusterActuatorLayer,
ClusterActuatorLayer>();

container.RegisterSingleton<IUserActuatorLayer,
UserActuatorLayer>();

            IUserAgentContext context =
container.GetInstance<IUserAgentContext>();
            SetupContext(context, configuration);
            IAmqpPeer amqp =
container.GetInstance<IAmqpPeer>();
            SetupAmqp(amqp, configuration);
            ITelegramPeer telegram =
container.GetInstance<ITelegramPeer>();
            SetupTelegramApi(telegram,
configuration);
            ILogger logger =
container.GetInstance<ILogger>();
            SetupLogger(logger, configuration);
        }

        private void SetupContext(IUserAgentContext
context, IAgentInitialConfiguration configuration)
        {
            TelegramConfig config =
configuration.GetConfig<TelegramConfig>();
            context.UserName =
config.AuthorizedUsername;
        }

        private void SetupAmqp(IAmqpPeer amqp,
IAgentInitialConfiguration configuration)
        {
            AmqpConfig config =
configuration.GetConfig<AmqpConfig>();
            amqp.Configure(config);
            amqp.Subscribe<AgentActivationEvent>();

amqp.Subscribe<AgentDeactivationEvent>();
            amqp.Subscribe<SolutionFoundRequest,
SolutionFoundResponse>();
        }

        private void SetupTelegramApi(ITelegramPeer
telegram, IAgentInitialConfiguration configuration)
        {
            TelegramConfig telegramConfig =
configuration.GetConfig<TelegramConfig>();
            telegram.Configure(telegramConfig);
        }

        private void SetupLogger(ILogger logger,
IAgentInitialConfiguration configuration)
        {
            LoggerConfig loggerConfig =
configuration.GetConfig<LoggerConfig>();
            logger.Configure(loggerConfig);
        }
    }
}
```

ClusterSensorLayer.cs

```
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Lib.logger;
using System;
using System.Threading.Tasks;

namespace GNetic.Agents.WorkerAgent.layers
{

    public class ClusterSensorLayer :
        IRequestHandler<WorkerStartTaskRequest,
WorkerStartTaskResponse>,
        IRequestHandler<WorkerStopTaskRequest,
WorkerStopTaskResponse>
    {
        private readonly IWorkerBehaviorLayer
BehaviorLayer;
        private readonly ILogger Logger;
        public
ClusterSensorLayer(IWorkerBehaviorLayer
behaviorLayer, ILogger logger)
        {
            BehaviorLayer = behaviorLayer;
            Logger = logger;
        }

        public async
Task<Response<WorkerStartTaskResponse>>
Handle(Request<WorkerStartTaskRequest> request)
        {
            Logger.Log($"Start task request
arrived, clusterId: {request.SenderId}");
            try
            {

BehaviorLayer.StartCalculation(request.Payload.Task
);
                return
Response<WorkerStartTaskResponse>.Ok();
            }
            catch (Exception ex)
            {
                return
Response<WorkerStartTaskResponse>.Fail(ex.Message);
            };
        }

        public async
Task<Response<WorkerStopTaskResponse>>
Handle(Request<WorkerStopTaskRequest> request)
```

```
        {
            Logger.Log($"Stop task request arrived,
clusterId: {request.SenderId}");
            try
            {
                BehaviorLayer.StopCalculation();
                return
Response<WorkerStopTaskResponse>.Ok();
            } catch (Exception ex)
            {
                return
Response<WorkerStopTaskResponse>.Fail(ex.Message);
            }
        }
    }
}
```

MigrationAgentActuatorLayer.cs

```
using GNetic.Agents.agent.communication;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.agent.layers;
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.WorkerAgent.layers
{
    public interface IMigrationActuatorLayer
    {
        Task<IEnumerable<Chromosome>>
PullChromosomes(string migrationAgentId);
        Task PushChromosome(Chromosome chromosome,
string migrationAgentId);

    }
    public class MigrationAgentActuatorLayer :
ActuatorLayer<IAmqpPeer>, IMigrationActuatorLayer
    {
        public
MigrationAgentActuatorLayer(IAmqpPeer amqp) :
base(amqp)
        {

        }
        public async Task<IEnumerable<Chromosome>>
PullChromosomes(string migrationAgentId)
        {
            var response = await
Endpoint.Request<PullChromosomeRequest,
PullChromosomeResponse>(migrationAgentId);
            var chromosomes =
response.Payload.Chromosomes;
            return chromosomes;
        }

        public async Task PushChromosome(Chromosome
chromosome, string migrationAgentId)
        {
            var request = new PushChromosomeRequest
            {
                Chromosome = chromosome
            };
            await
Endpoint.Request<PushChromosomeRequest,
PushChromosomeResponse>(request, migrationAgentId);
        }
    }
}
```

MigrationAgentSensorLayer.cs

```
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.WorkerAgent.layers
{
    public class MigrationAgentSensorLayer :
        IEventHandler<PullChromosomeRequiredEvent>,
        IEventHandler<PushChromosomeRequiredEvent>
    {
        private readonly IWorkerBehaviorLayer
BehaviorLayer;
        public
MigrationAgentSensorLayer(IWorkerBehaviorLayer
behaviorLayer)
        {
            BehaviorLayer = behaviorLayer;
        }
        public void
Handle(PullChromosomeRequiredEvent evt)
        {
            BehaviorLayer.PullChromosomes();
        }

        public void
Handle(PushChromosomeRequiredEvent evt)
        {
            BehaviorLayer.PushChromosome();
        }
    }
}
```

WorkerBehaviorLayer.cs

```
using GNetic.Agents.agent;
using GNetic.Lib.common;
using GNetic.Lib.dtos;
using GNetic.Lib.gmatic;
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Agents.WorkerAgent.layers
{
    public interface IWorkerBehaviorLayer
    {
        Task PullChromosomes();
        Task PushChromosome();
        void StartCalculation(WorkerTaskDto task);
        void StopCalculation();
    }
```

```
    public class WorkerBehaviorLayer :
IWorkerBehaviorLayer
    {
        private readonly IWorkerGNeticLayer
GNeticLayer;
        private readonly IMigrationActuatorLayer
MigrationActuator;
        private readonly IAgentContext Context;
        public WorkerBehaviorLayer(
            IAgentContext context,
            IWorkerGNeticLayer gNeticLayer,
            IMigrationActuatorLayer
migrationActuator
        )
        {
            Context = context;
            GNeticLayer = gNeticLayer;
            MigrationActuator = migrationActuator;
        }



        public async Task PullChromosomes()
        {
            var migrator = GetMigrator();
            var chromosomes = await
MigrationActuator.PullChromosomes(migrator.AgentId)
;

GNeticLayer.ReceiveChromosomes(chromosomes);
        }

        public async Task PushChromosome()
        {
            var chromosome =
GNeticLayer.GetBestChromosome();
            var migrator = GetMigrator();
            await
MigrationActuator.PushChromosome(chromosome,
migrator.AgentId);
        }

        public void StartCalculation(WorkerTaskDto
task)
        {
            GNeticOptions algorithmOptions = new
GNeticOptions()
            {
                CrossoveringChance =
task.AlgorithmSettings.CrossoveringChance,
                MutationChance =
task.AlgorithmSettings.MutationChance,
                InitialPopulationSize =
task.AlgorithmSettings.InitialPopulationSize,
                SurviveChromosomeCount =
task.AlgorithmSettings.SurviveChromosomeCount,
                OldestChromosomeAge =
task.AlgorithmSettings.OldestChromosomeAge,
                MaxOldestPercent =
task.AlgorithmSettings.MaxOldestPercent
            };

            ChromosomeOptions chromosomeOptions =
new ChromosomeOptions()
            {
                Deviation = new
Interval<double>(task.MathSettings.DeviationMin,
task.MathSettings.DeviationMax),
                MathExpectation = new
Interval<double>(task.MathSettings.MathExpectationM
in, task.MathSettings.MathExpectationMax),
                Dispersion = new
Interval<double>(task.MathSettings.DispersionMin,
task.MathSettings.DispersionMax),
                CrossoverType =
task.AlgorithmSettings.CrossoverType,
                MutationType =
task.AlgorithmSettings.MutationType,
                LSystemConstraints = new
Constraints()
                {
                    AxiomLength = new
Interval<int>(task.LSystemSettings.AxiomLengthMin,
task.LSystemSettings.AxiomLengthMax),
                    RuleCount = new
Interval<int>(task.LSystemSettings.RuleCountMin,
task.LSystemSettings.RuleCountMax),
                    RuleLength = new
Interval<int>(task.LSystemSettings.RuleLengthMin,
task.LSystemSettings.RuleLengthMax)
                }
            };
            TaskOptions taskOptions = new
TaskOptions()
            {
                TimeSeries =
task.TaskDescription.TimeSeries,
            };

GNeticLayer.StartCalculation(taskOptions,
algorithmOptions, chromosomeOptions);
        }
        public void StopCalculation()
        {
            GNeticLayer.StopCalculation();
        }

        private AgentInfo GetMigrator()
        {
            var migrator =
Context.RemoteAgents.FirstOrDefault(a =>
a.AgentType == AgentType.MigrationAgent);
            if (migrator == null)
                throw new Exception("No migration
agent found");
            return migrator;
        }
    }
}
```

WorkerGNeticLayer.cs

```
using GNetic.Lib.gnetic;
using GNetic.Lib.logger;
using System.Collections.Generic;

namespace GNetic.Agents.WorkerAgent.layers
{
    public interface IWorkerGNeticLayer
    {
        void StartCalculation(TaskOptions task,
GNeticOptions algorithmOptions, ChromosomeOptions
chromosomeOptions);
```

```
        void StopCalculation();
        void
ReceiveChromosomes(IEnumerable<Chromosome>
chromosomes);
        Chromosome GetBestChromosome();
    }

    public class WorkerGNeticLayer :
IWorkerGNeticLayer
    {
        private readonly IWorkerAgentContext
Context;
        private readonly ILogger Logger;
        public
WorkerGNeticLayer(IWorkerAgentContext context,
ILogger logger)
        {
            Context = context;
            Logger = logger;
        }

        public Chromosome GetBestChromosome()
        {
            if (Context.Algorithm == null)
                throw new
System.Exception("Algorithm does not in process
state");
            var chromosome =
Context.Algorithm.GetBestChromosome();
            return chromosome;
        }

        public void
ReceiveChromosomes(IEnumerable<Chromosome>
chromosomes)
        {
            if (Context.Algorithm == null)
                throw new
System.Exception("Algorithm does not in process
state");

Context.Algorithm.ApplyExternalChromosomes(chromoso
mes);
        }

        public void StartCalculation(TaskOptions
task, GNeticOptions algorithmOptions,
ChromosomeOptions chromosomeOptions)
        {
            if (Context.Algorithm != null)
                throw new System.Exception("System
job already in process!");
            Context.Algorithm = new
GNeticAlgorithm(task, algorithmOptions,
chromosomeOptions, Logger);
            Context.Algorithm.Run();
        }
        public void StopCalculation()
        {
            if (Context.Algorithm == null)
                throw new System.Exception("System
is not in the calculation state");
            Context.Algorithm.Stop();
            Context.Algorithm = null;
        }
    }
}
```

WorkerAgent.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.layers;
using LightInject;
using GNetic.Agents.agent.communication;

namespace GNetic.Agents.WorkerAgent
{
    public class WorkerAgent : Agent
    {
        public override void Run()
        {
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Run();
            amqp.AssertPublisher("all");
            IAgentContext context =
Container.GetInstance<IAgentContext>();
            context.AgentInfo = new
AgentInfo(amqp.QueueId, AgentType.WorkerAgent);
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendActivationSignal();
        }

        public override void Stop()
        {
            IEnvironmentActuatorLayer actuatorLayer
=
Container.GetInstance<IEnvironmentActuatorLayer>();
            actuatorLayer.SendDeactivationSignal();
            IAmqpPeer amqp =
Container.GetInstance<IAmqpPeer>();
            amqp.Stop();
        }
    }
}
```

WorkerAgentContext.cs

```
using GNetic.Agents.agent;
using GNetic.Lib.gnetic;

namespace GNetic.Agents.WorkerAgent
{
    public interface IWorkerAgentContext
    {
        GNeticAlgorithm Algorithm { get; set; }
    }
    public class WorkerAgentContext : AgentContext,
IWorkerAgentContext
    {
        public GNeticAlgorithm Algorithm { get;
set; }
    }
}
```

WorkerAgentStartup.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.agent.communication.events;
using
GNetic.Agents.agent.communication.events.models;
using GNetic.Agents.WorkerAgent.layers;
using GNetic.Lib.logger;
```

```
using LightInject;

namespace GNetic.Agents.WorkerAgent
{
    public class WorkerAgentStartup : AgentStartup
    {
        public override void
Configure(IServiceContainer container,
IAgentInitialConfiguration configuration)
        {
            base.Configure(container,
configuration);

container.RegisterMultiple<IAgentContext,
IWorkerAgentContext, WorkerAgentContext>();
            container.RegisterMultiple<

IRequestHandler<WorkerStartTaskRequest,
WorkerStartTaskResponse>,

IRequestHandler<WorkerStopTaskRequest,
WorkerStopTaskResponse>,
                ClusterSensorLayer
                >();

            container.RegisterMultiple<

IEventHandler<PullChromosomeRequiredEvent>,

IEventHandler<PushChromosomeRequiredEvent>,
                MigrationAgentSensorLayer
                >();

container.RegisterSingleton<IMigrationActuatorLayer
, MigrationAgentActuatorLayer>();


container.RegisterSingleton<IWorkerBehaviorLayer,
WorkerBehaviorLayer>();

container.RegisterSingleton<IWorkerGNeticLayer,
WorkerGNeticLayer>();

            IAmqpPeer amqp =
container.GetInstance<IAmqpPeer>();
            SetupAmqp(amqp, configuration);
            ILogger logger =
container.GetInstance<ILogger>();
            SetupLogger(logger, configuration);
        }

        private void SetupAmqp(IAmqpPeer amqp,
IAgentInitialConfiguration configuration)
        {
            AmqpConfig config =
configuration.GetConfig<AmqpConfig>();
            amqp.Configure(config);
            amqp.Subscribe<WorkerStartTaskRequest,
WorkerStartTaskResponse>();
            amqp.Subscribe<WorkerStopTaskRequest,
WorkerStopTaskResponse>();

amqp.Subscribe<PushChromosomeRequiredEvent>();

amqp.Subscribe<PullChromosomeRequiredEvent>();
            amqp.Subscribe<AgentActivationEvent>();
amqp.Subscribe<AgentDeactivationEvent>();
        }

        private void SetupLogger(ILogger logger,
IAgentInitialConfiguration configuration)
        {
            LoggerConfig config =
configuration.GetConfig<LoggerConfig>();
            logger.Configure(config);
        }
    }
}
```

ServiceContainerExtensions.cs

```
using LightInject;

namespace GNetic.Agents
{
    public static class ServiceContainerExtensions
    {

        public static void RegisterMultiple<TI1,
TI2, TService>(this IServiceContainer container)
            where TService : TI1, TI2
        {

container.RegisterSingleton<TService>();
            container.Register(f =>
(TI1)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI2)container.GetInstance<TService>(),
typeof(TService).FullName);
        }
        public static void RegisterMultiple<TI1,
TI2, TI3, TService>(this IServiceContainer
container)
            where TService : TI1, TI2, TI3
        {

container.RegisterSingleton<TService>();
            container.Register(f =>
(TI1)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI2)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI3)container.GetInstance<TService>(),
typeof(TService).FullName);
        }
        public static void RegisterMultiple<TI1,
TI2, TI3, TI4, TService>(this IServiceContainer
container)
            where TService : TI1, TI2, TI3, TI4
        {

container.RegisterSingleton<TService>();
            container.Register(f =>
(TI1)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI2)container.GetInstance<TService>(),
typeof(TService).FullName);
```

```
            container.Register(f =>
(TI3)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI4)container.GetInstance<TService>(),
typeof(TService).FullName);
        }
        public static void RegisterMultiple<TI1,
TI2, TI3, TI4, TI5, TService>(this
IServiceContainer container)
            where TService : TI1, TI2, TI3, TI4,
TI5
        {

container.RegisterSingleton<TService>();
            container.Register(f =>
(TI1)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI2)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI3)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI4)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI5)container.GetInstance<TService>(),
typeof(TService).FullName);
        }
        public static void RegisterMultiple<TI1,
TI2, TI3, TI4, TI5, TI6, TService>(this
IServiceContainer container)
            where TService : TI1, TI2, TI3, TI4,
TI5, TI6
        {

container.RegisterSingleton<TService>();
            container.Register(f =>
(TI1)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI2)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI3)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI4)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI5)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI6)container.GetInstance<TService>(),
typeof(TService).FullName);
        }
        public static void RegisterMultiple<TI1,
TI2, TI3, TI4, TI5, TI6, TI7, TService>(this
IServiceContainer container)
            where TService : TI1, TI2, TI3, TI4,
TI5, TI6, TI7
        {

container.RegisterSingleton<TService>();
            container.Register(f =>
(TI1)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI2)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI3)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI4)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI5)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI6)container.GetInstance<TService>(),
typeof(TService).FullName);
            container.Register(f =>
(TI7)container.GetInstance<TService>(),
typeof(TService).FullName);
        }
    }
}
```

2.2 Текст модуля GNetic.App

ClusterAgentApp.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.ClusterAgent;
using GNetic.Agents.ClusterAgent.models;
using GNetic.Agents.WorkerAgent;
using GNetic.Lib.common;
using GNetic.Lib.logger;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.App
{
    public class ClusterAgentApp :
IAgentApp<ClusterAgent>
    {
        public ClusterAgent Agent { get; set; }
        public void Run(string rootDir)
        {
            IAgentInitialConfiguration agentConfig
= new AgentInitialConfiguration();
            LoggerConfig loggerConfig =
File.ReadAllText($"{rootDir}\\logger\\config.json")
.ToModel<LoggerConfig>();
            AmqpConfig amqpConfig =
File.ReadAllText($"{rootDir}\\amqp\\config.json").T
oModel<AmqpConfig>();
```

```
            AgentsConfiguration workerConfigs =
File.ReadAllText($"{rootDir}\\configs\\config.json"
).ToModel<AgentsConfiguration>();
            agentConfig.SetConfig(loggerConfig);
            agentConfig.SetConfig(amqpConfig);
            agentConfig.SetConfig(workerConfigs);

            Agent = new ClusterAgent();

Agent.Configure<ClusterAgentStartup>(agentConfig);
            Agent.Run();
        }

        public void Stop()
        {
            Agent.Stop();
        }
    }
}
```

IAgentApp.cs

```
using GNetic.Agents.agent;

namespace GNetic.App
{
    public interface IAgentApp<TAgent> where TAgent
: IAgent
    {
        TAgent Agent { get; set; }
        void Run(string rootDir);
        void Stop();
    }
}
```

MigrationAgentApp.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.MigrationAgent;
using GNetic.Lib.common;
using GNetic.Lib.logger;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.App
{
    public class MigrationAgentApp :
IAgentApp<MigrationAgent>
    {
        public MigrationAgent Agent { get; set; }

        public void Run(string rootDir)
        {
            IAgentInitialConfiguration agentConfig
= new AgentInitialConfiguration();
            LoggerConfig loggerConfig =
File.ReadAllText($"{rootDir}\\logger\\config.json")
.ToModel<LoggerConfig>();
            AmqpConfig amqpConfig =
File.ReadAllText($"{rootDir}\\amqp\\config.json").T
oModel<AmqpConfig>();

            agentConfig.SetConfig(loggerConfig);
            agentConfig.SetConfig(amqpConfig);

            Agent = new MigrationAgent();

Agent.Configure<MigrationAgentStartup>(agentConfig)
;
            Agent.Run();
        }

        public void Stop()
        {
            Agent.Stop();
        }
    }
}
```

UserAgentApp.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.agent.communication.telegram;
using GNetic.Agents.UserAgent;
using GNetic.Lib.common;
using GNetic.Lib.logger;
using System.IO;

namespace GNetic.App
{
    public class UserAgentApp :
IAgentApp<UserAgent>
    {
        public UserAgent Agent { get; set; }

        public void Run(string rootDir)
        {
            IAgentInitialConfiguration agentConfig
= new AgentInitialConfiguration();

            LoggerConfig loggerConfig =
File.ReadAllText($"{rootDir}\\logger\\config.json")
.ToModel<LoggerConfig>();
            TelegramConfig telegramConfig =
File.ReadAllText($"{rootDir}\\telegram\\config.json
").ToModel<TelegramConfig>();
            AmqpConfig amqpConfig =
File.ReadAllText($"{rootDir}\\amqp\\config.json").T
oModel<AmqpConfig>();

            agentConfig.SetConfig(loggerConfig);
            agentConfig.SetConfig(telegramConfig);
            agentConfig.SetConfig(amqpConfig);

            Agent = new UserAgent();

Agent.Configure<UserAgentStartup>(agentConfig);
            Agent.Run();
        }
        public void Stop()
        {
            Agent.Stop();
        }
    }
}
```

WorkerApp.cs

```
using GNetic.Agents.agent;
using GNetic.Agents.agent.communication.amqp;
using GNetic.Agents.WorkerAgent;
using GNetic.Lib.common;
```

```
using GNetic.Lib.logger;
using System.IO;
using System.Threading;

namespace GNetic.App
{
    public class WorkerApp : IAgentApp<WorkerAgent>
    {
        public WorkerAgent Agent { get; set; }
        public void Run(string rootDir)
        {
            Thread.Sleep(1000);
            IAgentInitialConfiguration agentConfig
= new AgentInitialConfiguration();
            LoggerConfig loggerConfig =
File.ReadAllText($"{rootDir}\\logger\\config.json")
.ToModel<LoggerConfig>();
            AmqpConfig amqpConfig =
File.ReadAllText($"{rootDir}\\amqp\\config.json").T
oModel<AmqpConfig>();

            agentConfig.SetConfig(loggerConfig);
            agentConfig.SetConfig(amqpConfig);

            Agent = new WorkerAgent();

Agent.Configure<WorkerAgentStartup>(agentConfig);
            Agent.Run();
        }

        public void Stop()
        {
            Agent.Stop();
        }
    }
}
```

2.3 Текст модуля GNetic.App.Cluster

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace GNetic.App.Cluster
{
    class Program
    {
        static void Main(string[] args)
        {
            ClusterAgentApp app = new
ClusterAgentApp();
            app.Run("D:\\agents-root\\cluster-
root");
            Thread.Sleep(Timeout.Infinite);
        }
    }}
```

2.4 Текст модуля GNetic.App.MigrationAgent

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace GNetic.App.MigrationAgent
{
    class Program
    {
        static void Main(string[] args)
        {
            MigrationAgentApp app = new
MigrationAgentApp();
            app.Run("D:\\agents-root\\migration-
agent-root");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

2.5 Текст модуля GNetic.App.UserAgent

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace GNetic.App.UserAgent
{
    class Program
    {
        static void Main(string[] args)
        {
            UserAgentApp app = new UserAgentApp();
```

```
            app.Run("D:\\agents-root\\user-agent-
root");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

2.6 Текст модуля GNetic.App.Worker

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace GNetic.App.Worker1
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkerApp app = new WorkerApp();
            app.Run("D:\\agents-root\\worker1-
root");
            Thread.Sleep(Timeout.Infinite);
        }
    }

}
```

2.7 Текст модуля GNetic.App.Lib

FractalChainBuilder.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.builders
{
    public class FractalChainBuildOptions
    {
        public int SignificantSymbolsCount { get;
set; }
        public LSystem LSystem { get; set; }
        public Alphabet Alphabet { get; set; }

        public FractalChainBuildOptions(LSystem
system, int significantSymbolsCount, Alphabet
alphabet = null)
        {
            LSystem = system;
            SignificantSymbolsCount =
significantSymbolsCount;
            Alphabet = alphabet ?? new Alphabet();
        }
    }
    public class FractalChainBuilder :
IBuilder<string, FractalChainBuildOptions>
    {
        public string
Build(FractalChainBuildOptions options)
        {
            string start =
options.LSystem.AxiomRule.Production.Join();
            while
(start.ToStringArray().Where(lexem =>
options.Alphabet.SignificantLiterals.Contains(lexem
)).Count() < options.SignificantSymbolsCount)
            {
                start =
options.LSystem.Iterate(start);
            }

            int significantCount = 0;
            string result =
start.TakeWhile((symbol) =>
            {
                if
(options.Alphabet.SignificantLiterals.ToList().Cont
ains(symbol.ToString()))
                {
                    significantCount++;
                }
                return significantCount <=
options.SignificantSymbolsCount;
            }).Join();
            return result;
        }
    }
}
```

FractalSeriesBuilder.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.builders
{
    public class FractalSeriesBuildOptions
    {
        public string Sequence { get; set; }
        public double MathExpectation { get; set; }
        public double Deviation { get; set; }
        public double Dispersion { get; set; }
        public FractalSeriesBuildOptions(string
sequence, double mathExpectation, double deviation,
double dispersion)
        {
            Sequence = sequence;
```

```
            MathExpectation = mathExpectation;
            Deviation = deviation;
            Dispersion = dispersion;
        }
    }
    public class FractalSeriesBuilder :
IBuilder<double[], FractalSeriesBuildOptions>
    {
        public double[]
Build(FractalSeriesBuildOptions options)
        {
            List<double> timeSeries = new
List<double>();

            double mathExpectation =
options.MathExpectation;
            foreach (var lexem in
options.Sequence.ToStringArray())
            {
                switch (lexem)
                {
                    case "f":
                    case "g":
                    case "h":
                    case "p":
                        double dx =
0;//Rand.RandDouble(-options.Dispersion,
options.Dispersion);

timeSeries.Add(mathExpectation + dx);
                        break;
                    case "+":
                        mathExpectation +=
options.Deviation;
                        break;
                    case "-":
                        mathExpectation -=
options.Deviation;
                        break;
                }
            }
            return timeSeries.ToArray();
        }

    }
}
```

GraphBuilder.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.builders
{
    public class GraphBuilder : IBuilder<Node,
LSystem>
    {
        public Node Build(LSystem options)
        {
            LSystem system = options;

            HashSet<string> usedLexemsSet = new
HashSet<string>();

usedLexemsSet.Add(system.AxiomRule.Axiom);

usedLexemsSet.AddRange(system.AxiomRule.Production)
;
            foreach (Rule rule in system.Rules)
            {
                usedLexemsSet.Add(rule.Axiom);

usedLexemsSet.AddRange(rule.Production);
            }
            List<Node> nodes =
usedLexemsSet.Select(symbol => new
Node(symbol)).ToList();

            List<Rule> rules = new List<Rule>();
            rules.Add(system.AxiomRule.DeepCopy());
            foreach(Rule rule in system.Rules)
            {
                rules.Add(rule.DeepCopy());
            }

            foreach(Rule rule in rules)
            {
                Node node = nodes.Find(n => n.Axiom
== rule.Axiom);
                foreach(string symbol in
rule.Production)
                {
                    Node child =
nodes.FirstOrDefault(n => n.Axiom == symbol);
                    if (child != null)
                    {

node.RelatedNodes.Add(child);
                    }
                }
            }
            return nodes.First();
        }
    }
}
```

IBuilder.cs

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

namespace GNetic.Lib.builders
{
    public interface IBuilder <out TConstruction,
in TOptions>
    {
        TConstruction Build(TOptions options);
    }
}
```

LSystemBuilder.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using System.Collections.Generic;
using System.Linq;

namespace GNetic.Lib.builders
{
```

```
    public class LSystemBuilder : IBuilder<LSystem,
Constraints>
    {
        public LSystem Build(Constraints options)
        {
            Alphabet sourceAlphabet = new
Alphabet();
            Constraints constraints = options;

            LSystem system = null;
            do
            {
                system = new LSystem();
                int ruleCount =
Rand.RandInt(constraints.RuleCount.Min,
constraints.RuleCount.Max);
                int axiomLength =
Rand.RandInt(constraints.AxiomLength.Min,
constraints.AxiomLength.Max);
                string[] significantLexems =
sourceAlphabet.SignificantLiterals.Take(ruleCount).
ToArray();
                Rule axiom =
GenerateRule(sourceAlphabet.StartSymbol,
significantLexems, new string[] { }, axiomLength);

system.AddAxiom(sourceAlphabet.StartSymbol,
axiom.Production);

                for (int i = 0; i < ruleCount; i++)
                {
                    string ruleAxiom =
significantLexems[i];
                    string lexem =
significantLexems[i < ruleCount - 1 ? i + 1 : 0];
                    int ruleLength =
Rand.RandInt(constraints.RuleLength.Min,
constraints.RuleLength.Max);
                    Rule rule =
GenerateRule(ruleAxiom, new string[] { lexem },
sourceAlphabet.FunctorLiterals, ruleLength);
                    //Rule rule =
GenerateRule(ruleAxiom, significantLexems,
sourceAlphabet.FunctorLiterals, ruleLength);

                    system.AddRule(rule);
                }

            } while (!Validator.IsValid(system,
sourceAlphabet));
            return system;
        }
        private static Rule GenerateRule(string
axiom, string[] significantLexems, string[]
functorLexems, int ruleLength)
        {
            List<string> composeLiterals = new
List<string>();

composeLiterals.AddRange(significantLexems);

composeLiterals.AddRange(functorLexems);
            composeLiterals.Add(string.Empty);

            Rule rule;
            do
            {
                List<string> production = new
List<string>();
                for (int i = 0; i < ruleLength;
i++)
                {

production.Add(Rand.RandItem(composeLiterals));
                }
                string stringProduction =
production.Join();

                //string[] unnecessaryStatements =
new string[] { "+-", "-+" };
                //while
(stringProduction.ContainsSomeOf(unnecessaryStateme
nts))
                //{
                //    for (int i = 0; i <
unnecessaryStatements.Length; i++)
                //    {
                //        string statement =
unnecessaryStatements[i];
                //        stringProduction =
stringProduction.Replace(statement, string.Empty);
                //    }
                //}
                rule = new Rule(axiom,
stringProduction.ToStringArray());
            } while (rule.Production.Length < 1 &&
rule.Production.Where(s => s != "+" && s != "-
").Count() > 0);
            return rule;
        }
    }
}
```

BinaryConvert.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using
System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.common
{
    public class BinaryConvert
    {
        public static byte[] ToByteArray<T>(T obj)
        {
            if (obj == null)
                return null;
            BinaryFormatter bf = new
BinaryFormatter();
            using (MemoryStream ms = new
MemoryStream())
            {
                bf.Serialize(ms, obj);
                return ms.ToArray();
            }
        }

        public static object FromByteArray(byte[]
data)
        {
```

```
            if (data == null)
                return default;
            BinaryFormatter bf = new
BinaryFormatter();
            using (MemoryStream ms = new
MemoryStream(data))
            {
                object obj = bf.Deserialize(ms);
                return obj;
            }
        }

        public static T FromByteArray<T>(byte[]
data)
        {
            return (T)FromByteArray(data);
        }
    }
}
```

Extensions.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.common
{
    public static class ArrayExtensions
    {
        public static string Join(this string[]
array)
        {
            return string.Join("", array);
        }
        public static string Join(this
IEnumerable<string> array)
        {
            return string.Join("", array);
        }

        public static string Join(this char[]
array)
        {
            return string.Join("", array);
        }
        public static string Join(this
IEnumerable<char> array)
        {
            return string.Join("", array);
        }
        public static bool ContainsSomeOf(this
string target, IEnumerable<string> statements)
        {
            return statements.Where(statement =>
target.Contains(statement)).Count() > 0;
        }
        public static string[] ToStringArray(this
string source)
        {
            return source.ToArray().Select(ch =>
ch.ToString()).ToArray();
        }

        public static void AddRange<T>(this
HashSet<T> hashSet, IEnumerable<T> values)
        {
            foreach(T value in values)
            {
                hashSet.Add(value);
            }
        }

        private static Random rng = new Random();

        public static void Shuffle<T>(this IList<T>
list)
        {
            int n = list.Count;
            while (n > 1)
            {
                n--;
                int k = rng.Next(n + 1);
                T value = list[k];
                list[k] = list[n];
                list[n] = value;
            }
        }

        public static void Shuffle<T>(this IList<T>
list, int from, int to)
        {
            IList<T> shuffleRange = new List<T>();
            for(int i = from; i <= to; i++)
            {
                shuffleRange.Add(list[i]);
            }
            shuffleRange.Shuffle();
            IList<T> resultRange = new List<T>();
            for(int i = 0; i < from; i++)
            {
                resultRange.Add(list[i]);
            }
            foreach(T item in shuffleRange)
            {
                resultRange.Add(item);
            }
            for(int i = to + 1; i < list.Count;
i++)
            {
                resultRange.Add(list[i]);
            }
            for(int i = 0; i < list.Count; i++)
            {
                list[i] = resultRange[i];
            }
        }

        public static string ToJson(this object
obj)
        {
            string json =
JsonConvert.SerializeObject(obj,
Formatting.Indented);
            return json;
        }

        public static TModel ToModel<TModel>(this
string json)
        {
            return
JsonConvert.DeserializeObject<TModel>(json);
        }
```

```
    }


}
```

IDeepCopyable.cs

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

namespace GNetic.Lib.common
{
    public interface IDeepCopyable<TEntity>
    {
        TEntity DeepCopy();
    }
}
```

Interval.cs

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

namespace GNetic.Lib.common
{
    public class Interval<T> :
IDeepCopyable<Interval<T>> where T : struct,
IComparable<T>
    {
        public T Min { get; set; }
        public T Max { get; set; }

        public Interval(T min, T max)
        {
            Min = min;
            Max = max;
        }
        public Interval()
        {
            Min = default;
            Max = default;
        }

        public Interval<T> DeepCopy() =>
            new Interval<T>(Min, Max);

        public override string ToString()
        {
            return $"[Min: {Min}, Max: {Max}]";
        }
    }
}
```

Rand.cs

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

namespace GNetic.Lib.common
{
    public class Rand
    {
        private static Random generator { get; set;
} = new Random();

        public static double RandDouble(double min,
double max)
        {
            return generator.NextDouble() * (max -
min) + min;
        }

        public static double RandDouble(double min,
double max, params double[] exceptValues)
        {
            double targetValue = 0;
            do
            {
                targetValue = RandDouble(min, max);
            } while
(exceptValues.Contains(targetValue));
            return targetValue;
        }
        public static int RandInt(int min, int max)
        {
            return generator.Next(min, max + 1);
        }
        public static int RandInt(int min, int max,
params int[] exceptValues)
        {
            int targetValue = 0;
            do
            {
                targetValue = RandInt(min, max);
            } while
(exceptValues.Contains(targetValue));
            return targetValue;
        }
        public static T RandItem<T>(IEnumerable<T>
array)
        {
            int index = RandIndex(array);
            return array.ElementAt(index);
        }

        public static int
RandIndex<T>(IEnumerable<T> array)
        {
            int index = RandInt(0, array.Count() -
1);
            return index;
        }

        public static bool Truthy()
        {
            return RandInt(0, 1) == 0;
        }



    }

}
```

AlgorithmSettingsDto.cs

```
using GNetic.Lib.gnetic.crossover;
using GNetic.Lib.gnetic.mutators;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace GNetic.Lib.dtos
{
    [Serializable]
    public class AlgorithmSettingsDto
    {
        public int InitialPopulationSize { get;
set; }
        public int SurviveChromosomeCount { get;
set; }
        public int CrossoveringChance { get; set; }
        public CrossoverType CrossoverType { get;
set; }
        public int MutationChance { get; set; }
        public MutationType MutationType { get;
set; }
        public int OldestChromosomeAge { get; set;
}
        public int MaxOldestPercent { get; set; }


    }
}
```

LSystemSettingsDto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.dtos
{
    [Serializable]
    public class LSystemSettingsDto
    {
        public int AxiomLengthMax { get; set; }
        public int AxiomLengthMin { get; set; }
        public int RuleCountMax { get; set; }
        public int RuleCountMin { get; set; }
        public int RuleLengthMax { get; set; }
        public int RuleLengthMin { get; set; }
    }
}
```

MathSettings.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace GNetic.Lib.dtos
{
    [Serializable]
    public class MathSettings
    {
        public double MathExpectationMax { get;
set; }
        public double MathExpectationMin { get;
set; }
        public double DeviationMax { get; set; }
        public double DeviationMin { get; set; }
        public double DispersionMax { get; set; }
        public double DispersionMin { get; set; }
    }
}
```

SolutionDto.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.dtos
{
    [Serializable]
    public class SolutionDto
    {
        public double MathExpectation { get; set; }
        public double Deviation { get; set; }
        public double Fitness { get; set; }
        public IEnumerable<RuleDto> LSystem { get;
set; }
        public TimeSpan? CalculationTime { get;
set; } = null;

        public static SolutionDto
FromChromosome(Chromosome chromosome)
        {
            List<RuleDto> rules = new
List<RuleDto>();
            rules.Add(new RuleDto
            {
                Axiom =
chromosome.LSystem.AxiomRule.Axiom,
                Production =
chromosome.LSystem.AxiomRule.Production.Join()
            });
            foreach(var rule in
chromosome.LSystem.Rules)
            {
                rules.Add(new RuleDto
                {
                    Axiom = rule.Axiom,
                    Production =
rule.Production.Join()
                });
            }
            return new SolutionDto
            {
                Deviation = chromosome.Deviation,
                LSystem = rules,
                Fitness = chromosome.Fitness,
```

```
                MathExpectation =
chromosome.MathExpectation
            };
        }
    }

    [Serializable]
    public class RuleDto
    {
        public string Axiom { get; set; }
        public string Production { get; set; }
    }
}
```

TaskDto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.dtos
{
    [Serializable]
    public class TaskDto
    {
        public double[] TimeSeries { get; set; }
        public double Accuracy { get; set; }
    }
}
```

WorkerSettingsDto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.dtos
{
    [Serializable]
    public class WorkerSettingsDto
    {
        public AlgorithmSettingsDto
AlgorithmSettings { get; set; }
        public LSystemSettingsDto LSystemSettings {
get; set; }
        public MathSettings MathSettings { get;
set; }
    }
}
```

WorkerTaskDto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.dtos
{
    [Serializable]
    public class WorkerTaskDto : WorkerSettingsDto
    {
        public TaskDto TaskDescription { get; set;
}
    }
```

```
}
```

Alphabet.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gmatic
{
    public class Alphabet : IDeepCopyable<Alphabet>
    {
        public string StartSymbol { get; set; } =
"s";
        public string[] SignificantLiterals { get;
set; } = new string[] { "f", "g", "h", "p",
"z","x","y" };
        public string[] FunctorLiterals { get; set;
} = new string[] { "+", "-" };
        public Alphabet DeepCopy() =>
            new Alphabet()
            {
                StartSymbol = StartSymbol,
                SignificantLiterals =
SignificantLiterals.ToArray(),
                FunctorLiterals =
FunctorLiterals.ToArray(),
            };
    }
}
```

Constraints.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gnetic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gmatic
{
    public class Constraints :
IDeepCopyable<Constraints>
    {
        public Interval<int> AxiomLength { get;
set; } = new Interval<int>();
        public Interval<int> RuleCount { get; set;
} = new Interval<int>();
        public Interval<int> RuleLength { get; set;
} = new Interval<int>();
        public void FillFromAlphabet(Alphabet
alphabet)
                        {
            AxiomLength.Min = 1;
            AxiomLength.Max =
alphabet.SignificantLiterals.Length;
            RuleCount.Min = 1;
            RuleCount.Max =
alphabet.SignificantLiterals.Length;
            RuleLength.Min = 2;
            RuleLength.Max =
alphabet.SignificantLiterals.Length +
alphabet.FunctorLiterals.Length;
        }
        public Constraints DeepCopy() =>
            new Constraints()
```

```
            {
                AxiomLength =
AxiomLength.DeepCopy(),
                RuleCount = RuleCount.DeepCopy(),
                RuleLength = RuleLength.DeepCopy()
            };

    }
}
```

LSystem.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gmatic
{
    public class LSystem : IDeepCopyable<LSystem>
    {
        public Rule AxiomRule { get; set; }
        public List<Rule> Rules { get; private set;
} = new List<Rule>();
        public void AddAxiom(string startSymbol,
string[] axiom)
        {
            AxiomRule = new Rule(startSymbol,
axiom);
        }
        public void AddRule(Rule rule)
        {
            Rules.Add(rule);
        }

        public void RemoveRule(string axiom)
        {
            Rules.RemoveAll(r => r.Axiom == axiom);
        }

        public string Iterate(string start = "")
        {
            string axiom = start != string.Empty ?
start : AxiomRule.Production.Join();
            string iterationResult = string.Empty;
            foreach(string symbol in
axiom.ToStringArray())
            {
                Rule targetRule =
Rules.FirstOrDefault(rule => rule.Axiom == symbol);
                iterationResult += targetRule !=
null ? targetRule.Production.Join() : symbol;
            }
            return iterationResult;
        }

        public static LSystem
FromSpecification(Specification spec)
        {
            LSystem lSystem = new LSystem();
            lSystem.AddAxiom(spec.Axiom.Axiom,
spec.Axiom.Production);
            foreach(Rule rule in spec.Rules)
            {
                lSystem.AddRule(rule.DeepCopy());
            }
            return lSystem;
        }

        public Specification ToSpecification()
        {
            Specification specification = new
Specification();
            specification.Axiom =
AxiomRule.DeepCopy();
            foreach(Rule rule in Rules)
            {

specification.Rules.Add(rule.DeepCopy());
            }
            return specification;
        }

        public LSystemStats GetStats() => new
LSystemStats()
        {
            AxiomLength =
AxiomRule.Production.Length,
            RuleCount = Rules.Count,
            AverageRuleLength = (from r in Rules
select r.Production.Length).Sum()
        };
        public LSystem DeepCopy() =>
            FromSpecification(ToSpecification());

        public override string ToString()
        {
            List<string> result = new
List<string>();

result.Add(AxiomRule.Production.Join());
            foreach(Rule rule in Rules)
            {
                result.Add(rule.ToString());
            }
            return string.Join(" ", result);
        }
    }
}
```

LSystemStats.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gmatic
{
    public class LSystemStats
    {
        public int AxiomLength { get; set; }
        public int RuleCount { get; set; }
        public int AverageRuleLength { get; set; }
    }
}
```

Node.cs

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

namespace GNetic.Lib.gmatic
```

```
{
    public class Node
    {
        public string Axiom { get; set; }
        public HashSet<Node> RelatedNodes { get;
set; } = new HashSet<Node>();
        private bool IsVisited { get; set; } =
false;
        public Node(string axiom)
        {
            Axiom = axiom;
        }

        public bool IsRelatedWith(string
relatedAxiom)
        {
            if (IsVisited)
            {
                return false;
            }

            IsVisited = true;

            Node targetNode = null;
            if (Axiom == relatedAxiom)
            {
                targetNode = this;
            }
            else
            {
                targetNode =
RelatedNodes.FirstOrDefault(node => node.Axiom ==
relatedAxiom || node.IsRelatedWith(relatedAxiom));
            }
            return targetNode != null;
        }
    }
}
```

Rule.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gmatic
{
    public class Rule : IDeepCopyable<Rule>
    {
        public string Axiom { get; set; }
        public string[] Production { get; set; }

        public Rule(string axiom, string[]
production)
        {
            Axiom = axiom;
            Production = production;
        }
        public Rule DeepCopy() =>
            new Rule(Axiom, Production.ToArray());
        public override string ToString() =>
            $"{Axiom} => {string.Join("",
Production)}";
    }
}
```

Specification.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gmatic
{
    public class Specification :
IDeepCopyable<Specification>
    {
        public Rule Axiom { get; set; }
        public List<Rule> Rules { get; set; } = new
List<Rule>();
        public Specification DeepCopy() =>
            new Specification()
            {
                Axiom = Axiom.DeepCopy(),
                Rules = Rules.Select(rule =>
rule.DeepCopy()).ToList()
            };

    }


}
```

Validator.cs

```
using GNetic.Lib.builders;
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gmatic
{
    public class Validator
    {
        public static bool IsValid(LSystem system,
Alphabet alphabet)
        {

            //return system.Rules.Where(r =>
r.Production.Where(s => s != "+" && s != "-
").Count() > 1).Count() > 0;
            GraphBuilder graphBuilder = new
GraphBuilder();

            HashSet<string> lexemSet = new
HashSet<string>();
            lexemSet.Add(system.AxiomRule.Axiom);

lexemSet.AddRange(system.AxiomRule.Production);
            foreach (Rule rule in system.Rules)
            {
                lexemSet.Add(rule.Axiom);
                lexemSet.AddRange(rule.Production);
            }

            bool lSystemContactivityResult =
lexemSet.All(literal =>
            {
                Node graph =
graphBuilder.Build(system);
```

```
                return
graph.IsRelatedWith(literal);
            });


            bool isReproducing = true;
            string start =
system.AxiomRule.Production.Join();
            for (int i = 0; i < system.Rules.Count;
i++)
            {
                string production =
system.Iterate(start);
                string significantProduction =
production.Where(symbol =>
alphabet.SignificantLiterals.Contains(symbol.ToStri
ng())).Join();
                string significantStart =
start.Where(symbol =>
alphabet.SignificantLiterals.Contains(symbol.ToStri
ng())).Join();
                if (significantProduction.Length <=
significantStart.Length)
                {
                    isReproducing = false;
                    break;
                }
                start = production;
            }
            return lSystemContactivityResult &&
isReproducing;
        }


    }
}
```

CrossoverFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover.managing
{
    public class CrossoverFactory
    {
        private Dictionary<CrossoverType,
ICrossover> crossovers = new
Dictionary<CrossoverType, ICrossover>();
        public CrossoverFactory()
        {

crossovers.Add(CrossoverType.RandomCrossover, new
RandomCrossover());

crossovers.Add(CrossoverType.ExpectationDeviationIn
heritance, new ExpectationDeviationCrossover());

crossovers.Add(CrossoverType.FractalDeviationInheri
tance, new FractalDeviationInheritanceCrossover());

crossovers.Add(CrossoverType.FractalExpectationInhe
ritance, new
FractalDeviationInheritanceCrossover());
        }

        public ICrossover
GetCrossoverByType(CrossoverType type)
        {
            return crossovers[type];
        }
    }
}
```

CrossoverType.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover
{
    public enum CrossoverType
    {
        RandomCrossover,
        FractalDeviationInheritance,
        FractalExpectationInheritance,
        ExpectationDeviationInheritance
    }
}
```

ICrossover.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover
{
    public interface ICrossover
    {
        Chromosome Crossover(Chromosome mom,
Chromosome dad);
    }
}
```

ExpectationDeviationCrossover.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover
{
    public class ExpectationDeviationCrossover :
ICrossover
    {
        public Chromosome Crossover(Chromosome mom,
Chromosome dad)
        {
            Chromosome child = mom.DeepCopy();
            child.LSystem = dad.LSystem.DeepCopy();
            return child;
        }
    }
}
```

FractalDeviationInheritanceCrossover.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover
{
    public class
FractalDeviationInheritanceCrossover : ICrossover
    {
        public Chromosome Crossover(Chromosome mom,
Chromosome dad)
        {
            Chromosome child = mom.DeepCopy();
            child.MathExpectation =
dad.MathExpectation;
            return child;
        }
    }
}
```

FractalExpectationInheritance.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover
{
    public class FractalExpectationInheritance :
ICrossover
    {
        public Chromosome Crossover(Chromosome mom,
Chromosome dad)
        {
            Chromosome child = mom.DeepCopy();
            child.Deviation = dad.Deviation;
            return child;
        }
    }
}
```

RandomCrossover.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.crossover
{
    public class RandomCrossover : ICrossover
    {
        public Chromosome Crossover(Chromosome mom,
Chromosome dad)
        {
            int firstGen = Rand.RandInt(0, 3);
            int secondGen = Rand.RandInt(0, 3,
firstGen);

            var child = mom.DeepCopy();

            switch (firstGen)
            {
                case 0:
                    child.LSystem =
dad.LSystem.DeepCopy();
                    break;
                case 1:
                    child.MathExpectation =
dad.MathExpectation;
                    break;
                case 2:
                    child.Deviation =
dad.Deviation;
                    break;
                case 3:
                    child.Dispersion =
dad.Dispersion;
                    break;
            }

            switch (secondGen)
            {
                case 0:
                    child.LSystem =
dad.LSystem.DeepCopy();
                    break;
                case 1:
                    child.MathExpectation =
dad.MathExpectation;
                    break;
                case 2:
                    child.Deviation =
dad.Deviation;
                    break;
                case 3:
                    child.Dispersion =
dad.Dispersion;
                    break;
            }

            return child;


        }
    }
}
```

IMutator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.mutators
{
    public interface IMutator
    {
        Chromosome Mutate(Chromosome chromosome);
    }
}
```

MutationType.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.mutators
```

```
{
    public enum MutationType
    {
        RandomCreator,
        FractalConstraintsInherits,
        Shuffler,
        MathZip,
        MathUnzip
    }
}
```

MutatorFactory.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.mutators
{
    public class MutatorFactory
    {
        private Dictionary<MutationType, IMutator>
mutators = new Dictionary<MutationType,
IMutator>();
        public MutatorFactory()
        {

mutators.Add(MutationType.RandomCreator, new
RandomMutator());

mutators.Add(MutationType.FractalConstraintsInherit
s, new FractalInheritanceMutator());
            mutators.Add(MutationType.Shuffler, new
ShuffleMutator());
        }

        public IMutator
GetMutatorByType(MutationType type)
        {
            return mutators[type];
        }
    }
}
```

BaseMutator.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.mutators
{
    public class BaseMutator : IMutator
    {
        public Chromosome Mutate(Chromosome
chromosome)
        {
            int randomGen = Rand.RandInt(0, 3);
            Chromosome mutant =
chromosome.DeepCopy();
            switch (randomGen)
            {
                case 0:
                    mutant.LSystem =
MutateFractal(chromosome);
                    break;
                case 1:
                    mutant.MathExpectation =
MutateExpectation(chromosome);
                    break;
                case 2:
                    mutant.Deviation =
MutateDeviation(chromosome);
                    break;
                case 3:
                    mutant.Dispersion =
MutateDispersion(chromosome);
                    break;
            }
            return mutant;
        }

        protected virtual LSystem
MutateFractal(Chromosome chromosome)
        {
            return chromosome.LSystem.DeepCopy();
        }

        protected virtual double
MutateExpectation(Chromosome chromosome)
        {
            return
Rand.RandDouble(chromosome.Options.MathExpectation.
Min, chromosome.Options.MathExpectation.Max);
        }
        protected virtual double
MutateDeviation(Chromosome chromosome)
        {
            return
Rand.RandDouble(chromosome.Options.Deviation.Min,
chromosome.Options.Deviation.Max);
        }

        protected virtual double
MutateDispersion(Chromosome chromosome)
        {
            return
Rand.RandDouble(chromosome.Options.Dispersion.Min,
chromosome.Options.Dispersion.Max);
        }
    }
}
```

FractalInheritanceMutator.cs

```
using GNetic.Lib.gmatic;
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using GNetic.Lib.builders;

namespace GNetic.Lib.gnetic.mutators
{
    public class FractalInheritanceMutator :
BaseMutator
    {
```

```
        protected override LSystem
MutateFractal(Chromosome chromosome)
        {
            LSystemStats systemStats =
chromosome.LSystem.GetStats();
            Constraints constraints = new
Constraints()
            {
                AxiomLength = new
Interval<int>(systemStats.AxiomLength,
systemStats.AxiomLength),
                RuleCount = new
Interval<int>(systemStats.RuleCount,
systemStats.RuleCount),
                RuleLength =
chromosome.Options.LSystemConstraints.RuleLength.De
epCopy()
            };
            LSystemBuilder builder = new
LSystemBuilder();
            LSystem system =
builder.Build(constraints);
            return system;
        }
    }
}
```

RandomMutator.cs

```
using GNetic.Lib.builders;
using GNetic.Lib.gmatic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.mutators
{
    public class RandomMutator : BaseMutator
    {
        protected override LSystem
MutateFractal(Chromosome chromosome)
        {
            LSystemBuilder builder = new
LSystemBuilder();
            LSystem system =
builder.Build(chromosome.Options.LSystemConstraints
);
            return system;
        }
    }
}
```

ShuffleMutator.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic.mutators
{
    public class ShuffleMutator : BaseMutator
    {
        protected override LSystem
MutateFractal(Chromosome chromosome)
        {
            LSystem system =
chromosome.LSystem.DeepCopy();
            int ruleCount =
chromosome.LSystem.Rules.Count;

            for (int i = 0; i < ruleCount; i++)
            {
                Rule rule = system.Rules[i];
                rule.Production.Shuffle();
            }
            return system;
        }
    }
}
```

Chromosome.cs

```
using GNetic.Lib.builders;
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gnetic
{
    public class Chromosome :
IDeepCopyable<Chromosome>
    {
        public ChromosomeOptions Options { get;
private set; }
        public TaskOptions TaskOptions { get;
private set; }
        public Guid Id { get; set; } =
Guid.NewGuid();
        public int Age { get; set; }
        public double Fitness { get; set; }
        public LSystem LSystem { get; set; }
        public double MathExpectation { get; set; }
        public double Deviation { get; set; }
        public double Dispersion { get; set; }
        public Chromosome(ChromosomeOptions
options, TaskOptions taskOptions)
        {
            Options = options;
            TaskOptions = taskOptions;
            Id = Guid.NewGuid();
        }
```

```
        public void MakeBase()
        {
            LSystem = MakeLSystem();
            MathExpectation =
Rand.RandDouble(Options.MathExpectation.Min,
Options.MathExpectation.Max);
            Deviation =
Rand.RandDouble(Options.Deviation.Min,
Options.Deviation.Max);
            CalculateFitness();
        }
        public Chromosome DeepCopy() =>
            new Chromosome(Options, TaskOptions)
            {
                Age = 0,
                Fitness = Fitness,
                LSystem = LSystem.DeepCopy(),
                MathExpectation = MathExpectation,
                Deviation = Deviation,
                Dispersion = Dispersion
            };

        public void CalculateFitness()
        {

            FractalChainBuilder chainBuilder = new
FractalChainBuilder();
            FractalChainBuildOptions
chainBuildOptions = new
FractalChainBuildOptions(LSystem,
TaskOptions.TimeSeries.Length);
            string sequence =
chainBuilder.Build(chainBuildOptions);

            FractalSeriesBuilder seriesBuilder =
new FractalSeriesBuilder();
            FractalSeriesBuildOptions
seriesBuildOptions = new
FractalSeriesBuildOptions(sequence,
MathExpectation, Deviation, Dispersion);



            double[] timeSeries =
seriesBuilder.Build(seriesBuildOptions);

            double fitness = 0;
            for (int j = 0; j <
TaskOptions.TimeSeries.Length; j++)
            {
                fitness +=
Math.Pow(TaskOptions.TimeSeries[j] - timeSeries[j],
2);
            }

            Fitness = fitness;
        }

        private LSystem MakeLSystem()
        {
            LSystemBuilder builder = new
LSystemBuilder();
            return
builder.Build(Options.LSystemConstraints);
        }

        public override string ToString()
        {
            return $"{LSystem} {MathExpectation}
{Deviation}";
        }

    }
}
```

ChromosomeOptions.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gmatic;
using GNetic.Lib.gnetic.crossover;
using GNetic.Lib.gnetic.mutators;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gnetic
{
    public class ChromosomeOptions :
IDeepCopyable<ChromosomeOptions>
    {
        public Interval<double> MathExpectation {
get; set; }
        public Interval<double> Deviation { get;
set; }
        public Interval<double> Dispersion { get;
set; }
        public Constraints LSystemConstraints {
get; set; }
```

```
        public MutationType MutationType { get;
set; }
        public CrossoverType CrossoverType { get;
set; }
        public ChromosomeOptions DeepCopy() =>
            new ChromosomeOptions()
            {
                MathExpectation = MathExpectation,
                Deviation = Deviation,
                LSystemConstraints =
LSystemConstraints.DeepCopy(),
                MutationType = MutationType,
                CrossoverType = CrossoverType
            };
    }
}
```

GNeticAlgorithm.cs

```
using GNetic.Lib.common;
using GNetic.Lib.gnetic.crossover;
using GNetic.Lib.gnetic.crossover.managing;
using GNetic.Lib.gnetic.mutators;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using Newtonsoft.Json;
using GNetic.Lib.logger;

namespace GNetic.Lib.gnetic
{
    public class GNeticAlgorithm
    {
        private GNeticOptions Options { get; set; }
        private ChromosomeOptions ChromosomeOptions
{ get; set; }
        private TaskOptions TaskOptions { get; set;
}
        private Thread WorkerThread { get; set; }
        private List<Chromosome> Population { get;
set; }
        private int IterationsCount { get; set; } =
0;
        private bool IsInProcess { get; set; } =
false;
        private DateTime StartedOn { get; set; }
        private TimeSpan Durability { get; set; }
        private MutatorFactory MutatorFactory {
get; set; } = new MutatorFactory();
        private CrossoverFactory CrossoverFactory {
get; set; } = new CrossoverFactory();
        private List<Chromosome>
ExternalChromosomes = new List<Chromosome>();
        private ILogger Logger { get; set; }
        public GNeticAlgorithm(TaskOptions
taskOptions, GNeticOptions options,
ChromosomeOptions chromosomeOptions, ILogger
logger)
        {
            TaskOptions = taskOptions;
            Options = options;
            ChromosomeOptions = chromosomeOptions;
            Logger = logger;
        }

        public void
ApplyExternalChromosomes(IEnumerable<Chromosome>
chromosomes)
        {
            var chrs = chromosomes.Where(c => c !=
null);
            foreach (var chromosome in chrs)
                chromosome.Age = 0;
            Logger.Log($"Applying external
chromosomes: {chrs.Select(c => c.ToString() +
"\n").Join()}");
            ExternalChromosomes.AddRange(chrs);
        }

        public void ApplySettings(GNeticOptions
options, ChromosomeOptions chromosomeOptions)
        {
            Options = options;
            ChromosomeOptions = chromosomeOptions;

            Logger.Log(Options.ToJson(),
ChromosomeOptions.ToJson());
        }
        public void Run()
        {
            StartedOn = DateTime.Now;
            Thread workerThread = new Thread(new
ThreadStart(Processor));
            WorkerThread = workerThread;
            IsInProcess = true;
            WorkerThread.Start();
            Logger.Log(
                Options.ToJson(),
                ChromosomeOptions.ToJson()
            );
        }

        public void Stop()
        {
            IsInProcess = false;
            Durability = DateTime.Now - StartedOn;
            Population = (from c in Population
                          orderby c.Fitness
                          select c).ToList();
            Chromosome bestChromosome =
Population[0];
            Thread.Sleep(100);
            Logger.Log(
                bestChromosome,
                $"Iterations count:
{IterationsCount}",
                $"Calculation time: {Durability}"
            );
        }

        public Chromosome GetBestChromosome()
        {
            Chromosome chromosome = (
                from c in Population
                orderby c.Fitness
                select c
            ).First();

            return chromosome;
        }

        private void Processor()
```

```
        {
            List<Chromosome> chromosomes = new
List<Chromosome>();
            for(int i = 0; i <
Options.InitialPopulationSize; i++)
            {
                Chromosome chromosome = new
Chromosome(ChromosomeOptions, TaskOptions);
                chromosome.MakeBase();
                chromosomes.Add(chromosome);
            }
            Population = chromosomes;
            while (IsInProcess)
                PerformIteration();
        }

        private void PerformIteration()
        {
            RebasePopulationPhase();
            AddExternalChromosomes();
            CrossoverPhase();
            MutationPhase();
            CloneEliminationPhase();
            SelectionPhase();
        }

        private void AddExternalChromosomes()
        {
            if (ExternalChromosomes.Count == 0)
                return;

Population.AddRange(ExternalChromosomes);
            ExternalChromosomes.Clear();
        }

        private void RebasePopulationPhase()
        {
            IEnumerable<Chromosome>
oldestChromosomes = Population.Where(chromosome =>
chromosome.Age > Options.OldestChromosomeAge);
            int oldestCount =
oldestChromosomes.Count();

            if (oldestCount >= Population.Count *
Options.MaxOldestPercent / 100)
            {
                double averageAge = (from c in
Population select c.Age).Sum() / Population.Count;
                Logger.Log($"Average chromosome age
: {averageAge}");
                Population = (from c in Population
                              orderby c.Fitness
                              select c).ToList();
                Chromosome bestChromosome =
Population.First();
                Population.Clear();
                Population.Add(bestChromosome);
                for (int i = 0; i <
Options.InitialPopulationSize; i++)
                {
                    Chromosome chromosome = new
Chromosome(ChromosomeOptions, TaskOptions);
                    chromosome.MakeBase();
                    Population.Add(chromosome);
                }
            }
        }
        private void CrossoverPhase()
        {

            IList<Chromosome> crossovering =
Population.Where(c => Rand.RandInt(0, 100) <=
Options.CrossoveringChance).ToList();
            crossovering.Shuffle();
            if (crossovering.Count % 2 != 0)
            {

crossovering.RemoveAt(crossovering.Count - 1);
            }
            ICrossover crossoverOperator =
CrossoverFactory.GetCrossoverByType(ChromosomeOptio
ns.CrossoverType);
            for(int i = 0; i < crossovering.Count -
1; i++)
            {
                Chromosome parent1 =
crossovering[i];
                Chromosome parent2 = crossovering[i
+ 1];
                Chromosome child1 =
crossoverOperator.Crossover(parent1, parent2);
                Chromosome child2 =
crossoverOperator.Crossover(parent2, parent1);
                child1.CalculateFitness();
                child2.CalculateFitness();
                Population.Add(child1);
                Population.Add(child2);
            }
        }
        private void MutationPhase()
        {
            IList<Chromosome> mutants =
Population.Where(c => Rand.RandInt(0, 100) <=
Options.MutationChance).ToArray();
            IMutator mutationOperator =
MutatorFactory.GetMutatorByType(ChromosomeOptions.M
utationType);
            foreach(Chromosome mutant in mutants)
            {
                Chromosome mutated =
mutationOperator.Mutate(mutant);
                mutated.CalculateFitness();
                Population.Add(mutated);
            }
        }
        private void SelectionPhase()
        {
            foreach(var chromosome in Population)
            {
                chromosome.CalculateFitness();
            }
            Population = (
                from c in Population
                orderby c.Fitness
                select c

).Take(Options.SurviveChromosomeCount).ToList();
            foreach(Chromosome chromosome in
Population)
            {
                chromosome.Age++;
            }
            IterationsCount++;
            var avgFitness = (
```

```
                from c in Population
                select c.Fitness
            ).Sum() / Population.Count;

            Logger.LogAvgFitness(avgFitness);

Logger.LogBestFitness(Population[0].Fitness);

            Logger.Log(
                $"Iteration : {IterationsCount}",
                $"Average population fitness :
{avgFitness}",
                $"Best fitness :
{Population[0].Fitness}",
                $"Chromosome : {Population[0]}"
            );
        }
        private void CloneEliminationPhase()
        {
            var uniqPopulation = (
                from c in Population
                group c by c.ToString()
                into g
                where g.FirstOrDefault() != null
                select g.First()
            ).ToList();

            if (uniqPopulation.Count <
Population.Count)
            {
                int diff = Population.Count -
uniqPopulation.Count;
                for (int i = 0; i < diff; i++)
                {
                    Chromosome chromosome = new
Chromosome(ChromosomeOptions, TaskOptions);
                    chromosome.MakeBase();
                    uniqPopulation.Add(chromosome);
                }
            }
            Population = uniqPopulation;
        }


    }
}
```

GNeticOptions.cs

```
using GNetic.Lib.common;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GNetic.Lib.gnetic
{
    public class GNeticOptions
    {
        public int InitialPopulationSize { get;
set; }
        public int SurviveChromosomeCount { get;
set; }
        public int CrossoveringChance { get; set; }
        public int MutationChance { get; set; }
        public int OldestChromosomeAge { get; set;
}
        public int MaxOldestPercent { get; set; }
    }
}
```

TaskOptions.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.gnetic
{
    public class TaskOptions
    {
        public double[] TimeSeries { get; set; }
    }
}
```

ConsoleLogger.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.logger
{
    public class ConsoleLogger : ILogTransport
    {
        public void Log(string message)
        {
            Console.WriteLine(message);
        }
    }
}
```

FileLogger.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.logger
{
    public class FileLogger : ILogTransport,
IDisposable
    {
        private StreamWriter FileWriter { get; set;
}
        public FileLogger(string filePath)
        {
            FileWriter = new StreamWriter(filePath,
true);
        }
        public void Log(string message)
        {
            FileWriter.WriteLineAsync(message);
        }

        public void Dispose()
        {
            FileWriter.Dispose();
        }
    }
}
```

ILogTransport.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.logger
{
    public interface ILogTransport
    {
        void Log(string message);
    }
}
```

Logger.cs

```
using GNetic.Lib.common;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.logger
{
    public interface ILogger
    {
        void Log(params object[] args);
        void LogAvgFitness(double value);
        void LogBestFitness(double value);
        void Configure(LoggerConfig config);
    }

    public class Logger : ILogger,
IConfigurable<LoggerConfig>
    {
        private IList<LogTool> LogTools = new
List<LogTool>();
        private StreamWriter avgFitnessWriter;
        private StreamWriter bestFitnessWriter;

        public void Configure(LoggerConfig config)
        {
            LogTools.Clear();
            if (config.Loggers == null ||
config.Loggers.Count() == 0)
                throw new Exception("Loggers not
presented");
            foreach(ConfigLogger loggerInstance in
config.Loggers)
            {
                switch (loggerInstance.Type)
                {
                    case "console":
                        ILogTransport
consoleTransport = new ConsoleLogger();
                        LogTool console = new
LogTool(consoleTransport);
                        LogTools.Add(console);
                        break;
                    case "file":
                        string dir =
loggerInstance.OutputDirectory;
                        if
(string.IsNullOrEmpty(dir))
                            throw new
Exception("logger output directory is not valid");
                        if (!Directory.Exists(dir))

Directory.CreateDirectory(dir);
                        ILogTransport fileTransport
= new FileLogger($"{Guid.NewGuid()}.log");
                        LogTool fileLogger = new
LogTool(fileTransport);
                        LogTools.Add(fileLogger);
                        break;
                    default:
                        throw new
Exception("Unknown logger format");
                }
            }

            if
(!string.IsNullOrEmpty(config.AvgFitnessPath))
            {
                avgFitnessWriter = new
StreamWriter(config.AvgFitnessPath);
                avgFitnessWriter.AutoFlush = true;
            }

            if
(!string.IsNullOrEmpty(config.BestFitnessPath))
            {
                bestFitnessWriter = new
StreamWriter(config.BestFitnessPath);
                bestFitnessWriter.AutoFlush = true;
            }


        }

        public void Log(params object[] objs)
        {
            foreach (LogTool tool in LogTools)
            {
                string message = RawToString(objs);
                string formatted =
FormatMessage(message);
                tool.Log(formatted);
            }
        }

        public void LogAvgFitness(double value)
        {
            if (avgFitnessWriter == null)
                return;

            avgFitnessWriter.WriteLine("{" + $"
\"Value\": {value}, " + $" \"Date\":
{DateTimeOffset.Now.ToUnixTimeMilliseconds()}" + "
},");
        }

        public void LogBestFitness(double value)
        {
            if (bestFitnessWriter == null)
                return;
```

```
            bestFitnessWriter.WriteLine("{" + $"
\"Value\": {value}, "+ $"\"Date\":
{DateTimeOffset.Now.ToUnixTimeMilliseconds()}" + "
},");
        }

        private string FormatMessage(string
message)
        {
            return $"[{DateTime.Now}] {message}";
        }
        private string RawToString(params object[]
objs)
        {
            return string.Join(" ", objs.Select(obj
=> $"[{obj}]"));
        }
    }


}
```

LoggerConfig.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.logger
{
    [JsonObject]
    public class LoggerConfig
    {
        [JsonProperty("loggers")]
        public IEnumerable<ConfigLogger> Loggers;
        [JsonProperty("avgFitnessPath")]
        public string AvgFitnessPath { get; set; }
        [JsonProperty("bestFitnessPath")]
        public string BestFitnessPath { get; set; }
    }

    [JsonObject]
    public class ConfigLogger
    {
        [JsonProperty("type")]
        public string Type { get; set; }
        [JsonProperty("outputDirectory")]
        public string OutputDirectory { get; set; }
    }
}
```

LogTool.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GNetic.Lib.logger
{
    public class LogTool
    {
        private ILogTransport LogTransport { get;
set; }
        public LogTool(ILogTransport logTransport)
        {
            LogTransport = logTransport;
        }

        public void Log(string message)
        {
            LogTransport.Log(message);
        }
    }
}
```

IConfiguration.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace GNetic.Lib
{
    public interface IConfigurable<TConfig>
        where TConfig : class
    {
        void Configure(TConfig config)
```

Подано на конференцію: «81 Всеукраїнська науково-технічна конференція «Наука і сталий розвиток транспорту». Інформаційно-телекомунікаційні технології та комп'ютерне моделювання»

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ НА БАЗІ АГЕНТНОЇ МОДЕЛІ

Автор: Галушка О. В. студент групи ПЗ2021
Науковий керівник: доктор технічних наук, професор кафедри «Комп'ютерні інформаційні технології» Шинкаренко В.І.
Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Вирішення багатьох практичних задач зводиться до прогнозування часових рядів. Використовуючи фрактальні властивості деяких рядів кафедрою «КІТ» був розроблений новий підхід для моделювання та прогнозування часових рядів, який базується на відтворенні фрактально-конструктивної моделі заданого часового ряду за допомогою генетичних алгоритмів.

Процес відтворення фрактальної моделі потребує великої кількості обчислень в залежності від складності вхідного часового ряду. Для подальших досліджень та покращень даного підходу було прийнято рішення розробити обчислювальну систему на базі агентного підходу.

Були проведені наступні дослідження:

- визначення ролей та типів агентів в системі;
- вибір оптимального способу мережевої взаємодії між агентами;
- формалізація варіативності конфігурації генетичного алгоритму;
- організація міграції генофондів між агентами;
- способи корегування та оптимізації конфігурацій обчислювальних агентів під час роботи системи методом навчання з підкріпленням.

Таблиця 1- Знання та обов'язки агентів

Назва агенту	**Обов'язки**	**Знання**
Кластерний агент	Взаємодія з агентом користувача, облік доступних агентів в системі, ініціювання команд процесу обчислень, конфігурація обчислювальних агентів.	Інформація про агентів, конфігурації обчислювальних агентів, специфікація задачі (вхідні дані, необхідна точність рішення)
Агент міграцій	Взаємодія з обчислювальними агентами в контексті міграцій, вивід схеми міграцій в залежності від поточного стану обчислень.	Буфер генофонду, поточний стан обчислень кожного агенту.
Обчислювальний агент	Взаємодія з кластерним та міграційним агентами, процес обчислень за допомогою генетичного алгоритму з заданою конфігурацією.	Конфігурація, специфікація задачі.
Дослідницький агент	Спостереження за роботою системи з виводом оптимальних налаштувань системи	Склад, кількість агентів, схема конфігурацій
Агент взаємодії з користувачем	Валідація користувацьких команд з подальшим делегуванням кластерному агенту	Інформація про користувача, інформація про кластерного агента.

Для мережевої взаємодії між агентами використовується асинхронна черга повідомлень RabbitMQ, користувач взаємодіє з системою через Telegram Bot API.

Варіативність конфігурацій обчислювальних агентів забезпечується за рахунок таких параметрів генетичного алгоритму:

- розмір популяції;
- розмір селекції;
- вірогідність мутації;
- вірогідність схрещування;
- тип мутації;
- тип схрещування;
- обмеження складності фрактальної моделі;
- множини пошуку числових параметрів моделі.

Експериментально доведено, що комбінування різних стратегій генетичного алгоритму позитивно впливає на часову ефективність роботи системи.

Міграція генофондів між обчислювальними агентами організовується з забезпеченням достатнього рівня ізольованості популяцій. Для агрегації проміжних результатів використовується буфер хромосом, який являється складовою частиною міграційного агенту. Такий підхід до міграцій має перевагу над безпосередньою попарною міграцією між агентами, так як забезпечується більша різноманітність популяцій в контексті кожного агенту, що сильно зменшує вірогідність виродження популяції з подальшим попаданням в локальний оптимум, що являється однією з основних проблем сімейства генетичних алгоритмів.

В процесі виконання обчислень системою необхідно збирати статистичні дані, які описують процес обчислень, з подальшою їх обробкою. Агрегування та порівняння ефективності роботи різних агентів з різними конфігураціями дає змогу для корегування початкових конфігурацій. Таким чином оптимізація конфігурацій буде позитивно впливати на часову ефективність системи в цілому.

Проект статті

В.І. ШИНКАРЕНКО, О.В. ГАЛУШКА*

[1*]Кафедра комп'ютерних інформаційних технологій, Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010,
ел. пошта shinkarenko_vi@ua.fm
[2*]Кафедра комп'ютерних інформаційних технологій, Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010,
ел. пошта galushka.alex.1998@gmail.com

ВИКОРИСТАННЯ МУЛЬТИАГЕНТНОГО ПІДХОДУ ДЛЯ КОНСТРУКТИВНО-ПРОДУКЦІЙНОГО МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ ЧАСОВИХ РЯДІВ

Мета. Основною метою статті є розробка та програмна реалізація мультиагентної програмної системи для методу моделювання фрактальних часових рядів на основі конструктивно-продукційного моделюваннння. В контексті роботи методу використовується генетичний алгоритм для відтворення конструктивної моделі вхідного фрактального часового ряду побудованої на основі L-систем [1]. **Методика.** Для моделювання часового ряду необхідно відтворити його конструктивну модель яка будується за допомогою породжуючих конструкторів на основі L-систем. В контексті генетичного алгоритму конструктивна модель ряду представляється як хромосома, генами якої є параметри конструктивної моделі. Генетичний алгоритм як метод еволюційних обчислень застосовується в процесі відтворення конструктивної моделі для проведення операцій генерації та видозміни хромосом для знаходження конструктивної моделі яка відтворює заданий часовий ряд. Фітнес функцією виступає обчислення середньоквадратичного відхилення між вхідним та реконструйованим рядами. Мультиагентний підхід в розробці системи використовується для підвищення часової ефективності методу конструктивно-продукційного моделювання фрактальних часових рядів. **Результати.** Розроблена програмна система з використанням мультиагентного підходу яка використовує метод конструктивно-продукційного моделювання часових рядів. На основі порівняння часових характеристик було зроблено висновок про підвищення часової ефективності роботи методу моделювання в порівнянні з попередніми розробками в даній області. Було адаптовано метод моделювання для його використання в задачах прогнозування часових рядів. **Наукова новизна.** Метод конструктивно-продукційного моделювання отримав розвиток для використання в задачах прогнозування. Розроблена система з використанням мультиагентного підходу, що відкриває можливості для горизонтального масштабування обчислювальних ресурсів для більш ефективного використання методу за рахунок зменшення часу для обчислень. **Практична значимість.** Побудована система дозволяє ефективно збільшувати часову ефективність роботи методу конструктивно-продукційного моделювання за рахунок підключення в процес обчислень більшої кількості обчислювальних машин. Це дозволяє витрачати менше часу на обчислення та отримання результатів. Адаптація методу для задач прогнозування дозволяє використовувати його прогнозування майбутніх значень вхідного часового ряду.

Ключові слова: конструктивно-продукційне моделювання, L-система, агент, генетичний алгоритм, породжуючі конструктори.

Вступ

Вирішення багатьох практичних задач зводиться до прогнозування часових рядів. Для їх вирішення розроблено та використовується на практиці ряд підходів та методів. Починаючи з давно відомих методів: авторегресивна інтегрована ковзаюча та ковзаючий медіанний фільтр і закінчуючи новими сучасними підходами на основі універсальних засобів машинного навчання – нейромереж: нейромережа з роздільною здатністю [2], багаторівневі мережі уваги, просторово-часові залишкові мережі та інші. Для задач прогнозування часових рядів відомі такі методи: прогнозування часових рядів на основі нечіткої логіки, інтервальне прогнозування часових рядів, динамічний Байєсівський синтез, стохастичні диференціальні рівняння [3] та інші.

Фрактальні властивості деяких часових рядів дозволили розробити новий підхід для моделювання та прогнозування часових рядів.

Метод конструктивно-продукційного моделювання [5] використовує породжуючі конструктори на основі L-систем та генетичний алгоритм для відтворення конструктивної моделі вхідного часового ряду, на основі якої

можна можна проводити подальший аналіз часового ряду.

Після успішної адаптації методів конструктивізму для моделювання часових рядів та проведення серії випробувань було з'ясовано, що даний підхід до моделювання потребує значного об'єму обчислювальних потужностей для досягнення необхідного результату за адекватний час проведення процесу обчислень.

Виходячи з цього було прийнято рішення провести дослідження щодо підвищення часової ефективності методу конструктивно-продукційного моделювання для часових рядів за рахунок використання мультиагентного підходу в проектуванні та розробці системи. Це дозволить горизонтально масштабувати обчислювальні можливості системи за рахунок підключення в процес обчислень множину робочих машин.

Мета

Основною метою даної роботи є підвищення часової ефективності роботи методу конструктивно-продукційного моделювання фрактальних часових рядів за допомогою горизонтального масштабування обчислювальних ресурсів. Підвищення часової ефективності передбачає використання мультиагентного підходу в проектуванні та розробці системи.

Методика

Базовим поняттям конструктивно-продукційного моделювання є поняття узагальненого конструктора. У загальному випадку конструктор [4] являє собою трійку:

$$C = \langle M, \Sigma, \Lambda \rangle \qquad (1)$$

де M – поновлюваний неоднорідний носій, $\sum$ – сигнатура відносин та пов'язаних з ними операцій, Λ – інформаційне забезпечення процесу конструювання. Метод конструктивно-продукційного моделювання часових рядів використовує 3 типи конструкторів як один композитний конструктор для відтворення конструктивної моделі ряду. Функціональність конструкторів які входять в склад композитного конструктора наступна:

- C_{MS} - по заданій L-системі формує фрактальний мультисимвольний ланцюжок;
- C_{TS} - перетворює заданий мультисимвольний ланцюжок у фрактальний часовий ряд використовуючи такі параметри конструктивної моделі як математичне очікування (M), відхилення математичного очікування (dM) та дисперсію математичного очікування (σ);
- C_{RS} - з використанням генетичного алгоритму [6] знаходить L-систему, математичне очікування та відхилення математичного очікування. Дані параметри являють собою конструктивну модель вхідного часового ряду.
- C_{SB} - використовуючи задані обмеження в конструюванні L-системи, генерує нову.

Хромосома як базова складова концепції генетичного алгоритму розуміється як конструктивна модель з такими генами як L-система, математичне очікування, відхилення математичного очікування. Ступінь пристосованості хромосоми в контексті генетичного алгоритму розуміється як величина середньоквадратичного відхилення між вхідним часовим рядом та реконструйованим.

Операції які проводяться в процесі відтворення конструктивної моделі:

- $B_1\big|_{\{a,b,c,...,+,-\}}^{r,M_x,dM_x}$ – формування правої частини правила заміщення з символів $\{a,b,c,...,+,-\}$, математичне очікування та відхилення математичного очікування часового ряду;
- $B_2\big|_{P,r,M_x,dM_x}^{X,P}$ – формування хромосоми X та додавання її в популяцію P;
- $B_3\big|_{\Rightarrow(C_{MS}),S_L(X_n)}^{\Omega(C_{MS})}$ – ініціює вивід мультисимвольного ланцюга $\Omega(C_{MS})$ конструктором C_{MS}, передавши йому як параметр L-систему S_L хромосомы X_n;
- $B_4\big|_{\Rightarrow(C_{RS}),\Omega(C_{MS})}^{\Omega(C_{RS})}$ – ініціює вивід часового ряду конструктором C_{RS} за мультисимвольним ланцюгом $\Omega(C_{MS})$;

– $B_5\big|_{\Omega(C_{RS}),TS_{In}}^{q(X_n)}$ – обчислює значення фітнес-функції q хромосоми X_n порівнюючи вхідний TS_{In} і реконструйований за L-системою хромосоми X_n часові ряди за середньоквадратичним відхиленням величин рядів;

– $B_6\big|_P^P, B_7\big|_P^P, B_8\big|_P^P$ – сортування хромосом в популяції і видалення найменш пристосованих, додавання хромосом в результаті схрещування та мутації відповідно.

Використання генетичного алгоритму в процесі формування конструктивної моделі вхідного часового ряду ускладнює процес обчислень, так як евристичні методи обчислень потребують значних обчислювальних ресурсів.

Методи підвищення часової ефективності. Підвищення часової ефективності методу можна досягти наступними способами:

– вертикальне масштабування;
– горизонтальне масштабування;
– використання високопродуктивних мов програмування;
– рефакторінг та оптимізація розробленого методу.

Вертикальне масштабування ресурсів здійснюється за рахунок використання потенціалу потужних процесорів, встановлених на робочій машині. Проведення такого роду оптимізації залежить від наявності потужного технічного обладнання.

Горизонтальне масштабування – вид масштабування, при використанні якого обчислювальна потужність системи зростає з додаванням в процес обчислень нових ЕОМ, які працюють паралельно і частково або повністю незалежно одна від одної. Використання такого підходу для роботи системи потребує грамотного проектування системи на предмет розподіленої взаємодії робочих машин (агентів).

Ефективно підвищити часову ефективність роботи програмного забезпечення можна за рахунок вибору при її розробці високопродуктивних мов програмування, які були розроблені для високонавантажених задач. З таких мов програмування можна виділити наступні: Assembly, C, C++, Erlang, Rust, Go, C#/.NET, Java. Попередній список наведений в порядку зменшення продуктивності кінцевих бінарних файлів. При цьому слід відмітити, що швидкодія має свою ціну, яка випливає в час який необхідно витратити на процес розробки програмного продукту.

Рефакторінг та оптимізація передбачає перегляд розробленого ПЗ на предмет можливого впровадження оптимізацій безпосередньо в коді програми.

Попередній аналіз існуючого ПЗ. Для формування плану, кінцевою метою якого є підвищення часової ефективності методу конструктивно-продукційного моделювання для часових рядів, було проведено аналіз існуючого програмного додатку для виявлення його слабких сторін як з боку програмної реалізації, так і з боку формального представлення методу.

В результаті аналізу було виявлено такі слабкі місця, які негативно впливають на часову ефективність роботи методу:

– використання малоефективної мови програмування для проведення важких обчислень з боку генетичного алгоритму та конструювання ряду;
– відсутність паралелізму в процесі виконання обчислень;
– стрімке виродження популяції генетичного алгоритму, що призводить до його застрягання в локальному мінімумі.
– обмеженість стратегій рекомбінації генів в механізмах мутації та схрещування генетичного алгоритму.

Було виявлено, що реалізація роботи методу конструктивно-продукційного моделювання для часових рядів використовується мова програмування Python. Python – інтерпретуєма високорівнева мова програмування загального призначення зі строгою динамічною типізацією і автоматичним управлінням пам’яттю, орієнтована на підвищення продуктивності розробника, читаємості коду та його якості, а також переносимості написаних на ній програм. Основними недоліками даної мови є більш низька швидкість роботи і більше споживання оперативної пам’яті в порівнянні з аналогічним кодом, написаним на компільованих мовах, таких як C, C++, Go. Виходячи з цього, можна зробити однозначний висновок про необхідність вибору більш підходящої мови програмування для проведення обчислень з метою оптимізації процесу і, як наслідок, підвищення часової ефективності роботи методу.

Відсутність паралелізму в процесі виконання обчислень провокує неможливість

масштабування обчислювальних ресурсів як в контексті однієї ЕОМ на якій розташовано додаток, так і в контексті розподілення обчислень на множину робочих машин. Використання мультиагентного підходу в проектуванні системи дозволяє динамічно масштабувати потужність обчислень системи, за рахунок прилучення до процесу роботи методу множину ЕОМ.

Стрімке виродження популяції в процесі роботи генетичного алгоритму є загальною проблемою та слабким місцем для евристичних методів обчислень. Суть проблеми полягає в тому, що евристичні методи на певному етапі своєї роботи не можуть підібрати більш оптимальне рішення виходячи з наявного стану популяції хромосом. Тривале перебування в околицях локального оптимума призводить до стрімкої реплікації схожих генів різних хромосом в нові, що позбавляє всю популяцію різноманітності в контексті генофонду. Універсальних методів для боротьби з даним чинником не існує, але в контексті вирішення певних конкретних задач, деякі механізми оптимізації мають місце бути.

Обмеженість способів рекомбінації генів хромосом генетичного алгоритму, як правило призводить до виродження популяції та неповноцінне охоплення області пошуку більш оптимального рішення генетичним алгоритмом ніж є на певному етапі проведення обчислень. При виродженні популяції генетичний алгоритм довгий час перебуває в околицях локального оптимуму без певного просування, що дуже негативно впливає на часову ефективність алгоритму.

Попереднє проектування системи.

Вибір мови програмування.

Для підвищення швидкодії роботи методу в контексті часової ефективності обчислень, виходячи з попереднього аналізу існуючого ПЗ, актуальним питанням є вибір мови програмування для розробки системи. Основними вимогами для обраної мови є:

- висока швидкодія при виконанні обчислень;
- багата вбудована бібліотека класів для роботи з такими аспектами як обмін повідомленнями через мережу Інтернет, підтримка паралелізму;
- програмна сумісність з операційними система сімейств Windows, Linux;
- достатньо висока продуктивність розробки з боку розробника за рахунок наявності в мові вбудованих засобів управління пам'яттю;
- можлива підтримка контейнеризації агентних додатків для забезпечення більш гнучкого та зручного розгортання ПЗ на робочих машинах;
- компільованість та статична типізація.

Виходячи з даних вимог вибір мови програмування випав на C# на платформі .NET, яка має засоби для всіх з вище перелічених вимог.

Проектування та аналіз структури агентної системи.

Агентна система [7] – це система утворена декількома взаємодіючими інтелектуальними агентами. Агенти в контексті мультиагентної системи мають декілька важливих характеристик:

- автономність: агенти в деякій мірі незалежні та можуть приймати самостійні рішення про свою подальшу діяльність. Рішення агенту засновуються на його стані в певний момент часу та його представлені про середовище в якому він перебуває;
- обмеженість представлення: ні один з агентів не володіє представленням про систему в цілому, або представлення системи занадто складне для того щоб знання про її представлення мало практичне значення для агенту;
- децентралізація: кожний агент виконує свою роль в системі, і при цьому система немає класу агентів який може виконувати всю роботу та керувати усіма аспектами роботи системи.

В агентних системах може проявлятись самоорганізація і складна поведінка навіть якщо стратегія кожного окремо взятого класу агентів достатньо проста. Це особливість лежить в основі ройового інтелекту.

Розглянемо основну функціональність системи моделювання часових рядів та визначимо класи агентів та їх ролі. Для функціонування системи необхідно виділити певних клас агентів, який безпосередньо буде виконувати обчислення за допомогою генетичного алгоритму та методу конструктивно-продукційного моделювання. Далі необхідно впровадити ще один клас агентів, який буде відповідати за управління процесом обчислень в контексті всієї системи, облік активних агентів, контролем та управлінням кожним з обчислювальних агентів. Для виконання цієї ролі введемо клас

агентів які керують кластером обчислювальних агентів. Наступним не менш важливим класом агентів виступає агент взаємодії з користувачем. Користувач буде приводити систему в дію за допомогою текстових команд, відправляючи їх через програмний інтерфейс агента взаємодії з користувачем. Також доцільним буде введення процесу міграції хромосом між генофондами різних агентів. Для цього введемо клас міграційних агентів.

В ході визначення класів агентів та проектування агентної системи було визначено наступні класи агентів, їх ролі та обов'язки в роботі системи, а також знання кожного з класу агентів (табл. 1.1)

Таблиця 1.1 – Класи агентів в системі

Назва агенту	Обов'язки	Знання
Кластерний агент	Взаємодія з агентом користувача, облік доступних агентів в системі, ініціювання команд процесу обчислень, конфігурація обчислювальних агентів.	Конфігурації агентів обчислювальних агентів, специфікація задачі (вхідні дані, необхідна точність рішення)
Агент міграцій	Взаємодія з обчислювальними агентами в контексті міграцій, вивід схеми міграцій в залежності від поточного стану обчислень.	Буфер генофонду, поточний стан обчислень кожного агенту.
Обчислювальний агент	Взаємодія з іншими агентами, процес обчислень за допомогою генетичного алгоритму з заданою конфігурацією.	Конфігурація, специфікація задачі.
Агент взаємодії з користувачем	Валідація користувацьких команд з подальшим делегуванням кластерному агенту	Інформація про користувача, інформація про кластерного агента.

Наступним важливим аспектом при проектуванні та розробці агентної системи є вибір способу мережевої взаємодії агентів в контексті системи.

Вимоги до обміну повідомленнями між агентами наступні:

– необхідність забезпечення надійного каналу зв'язку між кожною парою агентів;
– забезпечення консистентності при пересиланні повідомлення;
– можлива відсутність публічної IP адреси на ЕОМ на якій розгортується агент;

Сформуємо матрицю суміжності необхідних вимог та способів передачі повідомлень (табл. 1.2).

Таблиця 1.2 – матриця суміжності вимог та способів мережевої взаємодії

Спосіб передачі / Вимоги	Надійність каналу зв'язку	Консистентність даних повідомлення	Відсутність публічної IP адреси
HTTP (TCP/IP)	+	+	-
JSON-RPC (TCP/IP)	+	+	-
AMQP (TCP/IP)	+	+	+
Websockets (TCP/IP)	-	+	-
Datagrams (UDP/IP)	-	-	-

На основі матриці можна зробити висновок, що для обміну повідомленнями в системі найбільш прийнятним способом можна вважати асинхронну чергу повідомлень на базі протоколу AMQP [8].

Підвищення часової ефективності роботи генетичного алгоритму.

В процесі аналізу та екпериментальних випробувань для існуючого ПЗ яке використовую метод конструктивно-продукційного моделювання для часових рядів часто виникала ситуація з виродженням популяції [9] генетичного алгоритму за рахунок домінації схожих генів. При попаданні популяції в околиці локального оптимуму [10] прослідковувалось тривала затримка процесу обчислень без видимого прогресу. Для боротьби з наслідками зациклення генетичного

алгоритму був впроваджений механізм перегенерації складу популяції.

Суть механізму полягає в практично повному знищенні генофонду генетичного алгоритму залишаючи в ній тільки одну хромосому з найкращим значенням обчисленої для нього фітнес-функції, всі інші хромосоми перегенеруються. При генерації нових хромосом їхні гени формуються випадковим чином, що додає в популяції різноманітність. Експериментально доведено ефективність даного механізму, часова ефективність при його використанні значно зростає. На рис. 1.1 зображено середнє значення фітнес-функції для популяції в процесі проведення експерименту.

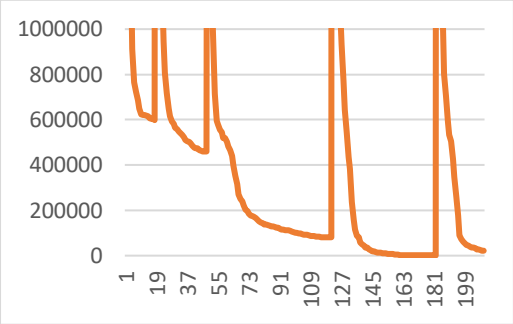


Рисунок 1.1 – середнє значення фітнес-функції для популяції

Різке зростання середнього значення фітнес-функції популяції свідчить про застосування механізму описаного вище. Видно, що після проведення генерації нових хромосом невдовзі якість популяції зростає внаслідок збільшення різноманітності популяції з додаванням нових генів.

Міграція генофондів.

Міграція генофондів між розподіленими генетичними алгоритмами які проводять обчислення паралельно є одним зі способів досягнення більшої часової ефективності роботи методу конструктивно-продукційного моделювання для часових рядів.

Використання різних стратегій та механізмів для проведення операцій мутації та схрещування різними обчислювальними агентами з послідуючим обміном отриманих хромосом між агентами дозволяє значно розширити область пошуку хромосом з більш кращими генами. Управління та контроль за процесом міграції покладається на міграційного агента, який інкапсулює в собі допоміжний банк хромосом та в процесі проведення обчислень системою за допомогою запитів до обчислювальних агентів ініціює обмін представниками генофондів між ними. Впровадження міграції в роботу розподілених генетичних алгоритмів значно прискорило роботу системи в цілому, що свідчить про підвищення часової ефективності методу моделювання.

Результати

У статті запропоновано концепцію проектування та розробки мультиагентної системи моделювання часових рядів за допомогою методу конструктивно-продукційного моделювання.

Розроблено мультиагентну систему для моделювання часових рядів.

В результаті порівняння часових характеристик існуючого ПЗ для моделювання та розробленої системи зроблено висновок про успішне підвищення часової ефективності методу конструктивно-продукційного моделювання для часових рядів. Порівняння зображено на (рис. 1.2).

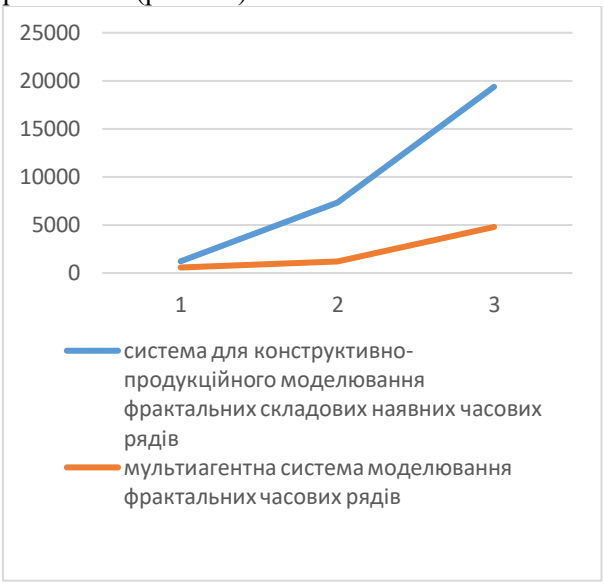


Рисунок 1.2 – порівняння часової ефективності в залежності від складності конструктивної моделі.

Під складність конструктивної моделі в даному контексті розуміється кількість правил заміщення в L-системі, яка є однією зі складових моделі.

Внаслідок підвищення часової ефективності методу, були проведені початкові дослідження на предмет використання методу для прогнозування часових рядів, в результаті яких метод моделювання було адаптовано для прогнозування значень часових рядів.

Наукова новизна та практична значимість

Отримано подальший розвиток методу конструктивно-продукційного моделювання для фрактальних часових рядів. Розроблена агентна система яка використовує розподілені еволюційні обчислення на базі генетичного алгоритму. Підвищення часової ефективності методу моделювання відкриває можливості для його використання в задачах прогнозування часових рядів. Використання агентної концепції для проектування та розробки системи дозволяє ускладнювати її функціонал за рахунок розробки додаткових класів агентів.

Висновки

В роботі запропоновано використання агентної концепції для проектування та розробки системи для моделювання фрактальних часових рядів з метою підвищення методу конструктивно-продукційного моделювання для часових рядів. В результаті розробки системи та проведення експериментів було зроблено висновок про успішне підвищення часової ефективності методу моделювання, що в свою чергу дозволяє використовувати даний підхід не тільки для моделювання часових рядів, а й для прогнозування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shynkarenko V.I., Zhadan A.A. «Constructive-synthesizing modeling of the existing time series fractal components». DNURT 2020, in print.
2. «Нейромережевий аналіз з роздільною здатністю» [Ел. ресурс]. Available: https://nauka-online.com/ua/publications/informatsionnye-tehnologii/2019/6/efektivnist-vikoristannya-nejromerezhevih-modelej-dlya-prognozuvannya-ruhu-tsin-aktsij-kompanij-na-rinku . [Дата звертання: 2.12.2021].
3. «Стохастичні диференціальні рівняння» [Ел. ресурс]. Available: https://uk.freejournal.org/577913/1/stokhastichne-diferentsialne-rivnyannya.html . [Дата звертання: 2.12.2021].
4. Shynkarenko V., Zhadan A., “Modeling of the Deterministic Fractal Time Series by One Rule Constructor”, Computer Science and Information Technologies 2020 (CSIT 2020), vol. 1, pp. 336-340, in print.
5. Shynkarenko V.I. and Ilman V.M., “Constructive-Synthesizing Structures and Their Grammatical Interpretations. Part II. Refining Transformations”. Cybernetics and Systems Analysis, 50(6), 2014, 829 – 841. doi: 10.1007/s10559-014-9674-9; “Part I. Generalized Formal Constructive-Synthesizing Structure”. Cybernetics and Systems Analysis, vol. 50(5), 2014, pp. 665 – 662. doi: 10.1007/s10559-014-9655-z
6. «Генетичні алгоритми» [Ел. ресурс]. Available: https://neurohive.io/ru/osnovy-data-science/chto-takoe-geneticheskie-algoritmy . [Дата звертання: 2.12.2021].
7. «Багатоагентна система» [Ел. ресурс]. Available: https://uk.wikipedia.org/wiki/Багатоагентна_система . [Дата звертання: 2.12.2021].
8. «AMQP specification» [Ел. ресурс]. Available: https://www.rabbitmq.com/protocol.html. [Дата звертання: 2.12.2021].
9. «Виродженість в генетичних алгоритмах» [Ел. ресурс]. Available: http://www.mathnet.ru/links/d44915a3286f0954954d04a583887099/pu146.pdf . [Дата звертання: 2.12.2021].
10. «Генетичні алгоритми. Локальний оптимум» [Ел. ресурс]. Available: https://cyberpedia.su/11x2a08.html . [Дата звертання: 2.12.2021].