Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр Післядипломної освіти
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження та використання алгоритмів еволюційного моделювання в інтелектуальних системах прийняття рішень
(тема)

Виконав:
студент 2 курсу, групи СШІмзд-18-1
Авдусь А.В.
(прізвище, ініціали)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва спеціальності)

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

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва спеціалізації)

Керівник к.т.н., доц. Шевченко О.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Центр Післядипломної освіти
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва)

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

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Авдусь Андрію Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та використання алгоритмів еволюційного моделювання в інтелектуальних системах прийняття рішень
затверджена наказом університету від ___ _____ 20 ___ р. №_____
2. Термін подання студентом роботи до екзаменаційної комісії ____ _____ 20__ р.
3. Вихідні дані до роботи Документація до необхідних у проекті мов програмування, інші Інтернет джерела та література з вказаної теми

4. Перелік питань, що потрібно опрацювати в роботі
1 Огляд алгоритмів еволюційного моделювання та їх застосування в інтелектуальних системах прийняття рішень
2 Дослідження алгоритмів еволюційного моделювання
3 Проектування ІСПР на основі алгоритму мурашиних колоній
4 Програмна реалізація системи на основі алгоритму мурашиних колоній

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____
Рисунок 1.1 – Покоління СППР; Рисунок 1.2 – Класифікація ІСПР за видами відтворених знань людини; Рисунок 3.1 – Опис бізнес-процесу "Оформлення замовлення на доставку"; Рисунок 3.2 – Опис бізнес-процесу "Побудова оптимального маршруту доставки"; Рисунок 3.3 – Функціональні вимоги до ІСПР; Рисунок 3.4 – Діаграма Use Case для ІСПР; Рисунок 3.5 – алгоритм роботи ІСПР; Рисунок 3.6 – Діаграма діяльності "Пошук оптимального маршруту"; Рисунок 3.7 – Діаграма класів ІСПР; Рисунок 4.1 – Взаємозв'язок між основними класами програми; Рисунок 4.2 – Загальна схема архітектури програми; Рисунок 4.3 – Проектування графічного інтерфейсу програми "Мурашиний алгоритм"; Рисунок 4.4 – Створення (ініціалізація) колонії; _____

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	к.т.н., доц. Шевченко О.Ю.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.04.2020	виконано
2	Аналіз предметної галузі та постановка задачі	07.04.2020	виконано
3	Теоретичні дослідження з предметної галузі	08.04.2020	виконано
4	Розробка інформаційної бази	12.05.2020	виконано
5	Розробка програмного забезпечення	13.05.2020	виконано
6	Підготовка пояснювальної записки	21.05.2020	виконано
7	Попередній захист атестаційної роботи	26.05.2020	виконано
8	Захист атестаційної роботи	28.05.2020	виконано

Дата видачі завдання _____ _____ 20 ___ р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 139 с., 28 рис., 3 дод., 29 джерел.

АЛГОРИТМ МУРАШИНИХ КОЛОНІЙ, ЕВОЛЮЦІЙНИЙ АЛГОРИТМ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА ПРИЙНЯТТЯ РІШЕНЬ, ОПТИМАЛЬНИЙ ПОШУК, ПРОЕКТУВАННЯ СИСТЕМИ

Об'єкт дослідження – алгоритми еволюційного моделювання.

Предмет дослідження – застосування алгоритмів еволюційного моделювання в інтелектуальних системах прийняття рішень на прикладі алгоритму мурашиних колоній.

Було проведено вивчення і аналіз існуючої літератури з даної теми, порівняння алгоритмів еволюційного моделювання, та дослідження алгоритму мурашиних колоній, які їх реалізують на основі існуючих даних в літературі, проектування і реалізація інтелектуальних систем прийняття рішень.

В першому розділі здійснено огляд алгоритмів еволюційного моделювання та їх застосування в інтелектуальних системах прийняття рішень (ІСПР).

В другому розділі проведено аналіз найбільш уживаних алгоритмів еволюційного моделювання.

В третьому і четвертому розділах була спроектована і реалізована система для оптимального пошуку маршруту доставки вантажів на основі алгоритму мурашиних колоній.

РЕФЕРАТ

Пояснительная записка: 139 с., 28 рис., 3 прил., 29 источников.

АЛГОРИТМ МУРАВЬИННЫХ КОЛОНИЙ, ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА ПРИНЯТИЯ РЕШЕНИЙ, ОПТИМАЛЬНЫЙ ПОИСК, ПРОЕКТИРОВАНИЕ СИСТЕМЫ, ЭВОЛЮЦИОННЫЙ АЛГОРИТМ

Объект исследования – алгоритмы эволюционного моделирования.

Предмет исследования – применение алгоритмов эволюционного моделирования в интеллектуальных системах принятия решений на примере алгоритма муравьиных колоний.

Были проведены изучение и анализ существующей литературы по данной теме, сравнение алгоритмов эволюционного моделирования, и исследование алгоритмов муравьиных колоний, которые их реализуют на основе существующих данных в литературе, проектирование и реализация интеллектуальных систем принятия решений.

В первом разделе рассмотрены алгоритмы эволюционного моделирования и их применение в интеллектуальных системах принятия решений (ИСПР).

Во втором разделе проведен анализ наиболее употребляемых алгоримов эволюционного моделирования.

В третьем и четвертом разделах спроектирована и реализована система для оптимального поиска маршрута доставки грузов на основе алгорима муравьинных колоний.

ABSTRACT

Explanatory note: 139 p., 28 figures, 3 ann., 29 sources.

ANT COLONY ALGORITHM, DECISION TAKING INTELLECTUAL SYSTEM, EVOLUTIONARY ALGORITHM, MOST EFFICIENT SEARCH, SYSTEM DESIGNING

This degree work is devoted to evolutionary computation algorithms.

The subject of research is applying evolutionary computation algorithms in the decision taking intellectual systems based on the ant colony algorithm.

The studying and analyzing the existing topic related literature, comparing evolutionary computation algorithms and studying the ant colony algorithm, which are implemented based on the existing data in the literature, designing and implementing decision taking intellectual systems have been performed.

The first part covers an overview of the evolutionary computation algorithms and their implementation in the decision taking intellectual systems (DTIS).

The second part analyzes the most frequently used evolutionary computation algorithms.

The third and the fourth parts cover designing and implementing a system for the most efficient cargo delivery route search based on the ant colony algorithm.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

БМ – база моделей;

ІСПР – інтелектуальна система прийняття рішень;

СППР – система підтримки прийняття рішень;

СУБД – система управління базою даних;

СШІ – система штучного інтелекту;

ACO – Ant Colony Optimization – оптимізація колонією мурах;

CHP – Cross-population selection, Heterogenous recombination and Cataclysmic mutation, Ch. L. Eshelman – генетичний алгоритм Эшельмана;

TS algorithm – Tabu Search algorithm – алгоритм пошуку з забороною;

TSP – Traveling Salesman Problem – задача комівояжера;

VRP – Vehicle Routing Problems – задачі маршрутизації транспорту;

VRPTW – VRP with Time Windows – маршрутизація з обмеженням в часі.

ВСТУП

Актуальність. Широкий розвиток і застосування штучного інтелекту в різних сферах діяльності людей зумовлює пошук нових методів і підходів для його реалізацію. Одним із перспективних напрямів дослідження є використання еволюційного моделювання при прийнятті рішень. Методи еволюційного моделювання засновані на еволюційній теорії Дарвіна, в основі моделей яких покладено принципи селекції, схрещування і мутації особин.

Дослідження показали ефективність таких алгоритмів і доцільність їх застосування при розв’язуванні задач як безумовної, так і умовної оптимізації функцій багатьох змінних. Постійно пропонуються нові чи модифіковані алгоритми для поліпшення продуктивності методів даної групи або для розширення сфери їх застосування.

Проте, для зрозумілого застосування кінцевим користувачем таких алгоритмів необхідне спеціальне програмне забезпечення, яке їх реалізовує. Найбільш поширеними різновидами інформаційних систем, в яких реалізують еволюційні алгоритми, є інтелектуальні системи прийняття рішень (ІСПР) та системи штучного інтелекту (СШІ). Вони складають базу моделей і є невід’ємною їх складовою.

Метою атестаційної роботи є дослідження еволюційних алгоритмів та їх застосування в інтелектуальних системах прийняття рішень.

Для реалізації мети було визначено ряд завдань, а саме:

– дослідити теоретичні положення розвитку і використання ІСПР;

– здійснити теоретичні дослідження найпоширеніші еволюційних алгоритмів за їх видами;

– дослідити математичну основу найпоширеніших еволюційних алгоритмів;

– розробити модель пошуку оптимального маршруту доставки вантажів на основі алгоритму мурашиних колоній в умовах часових обмежень;

– розробити проект ІСПР на основі еволюційного алгоритму, а саме на основі алгоритму мурашиних колоній;

– розробити програмний модуль реалізації алгоритму мурашиних колоній.

Робота складається зі вступу, чотирьох розділів, висновків, додатків та переліку джерел посилання.

1 ОГЛЯД АЛГОРИТМІВ ЕВОЛЮЦІЙНОГО МОДЕЛЮВАННЯ ТА ЇХ ЗАСТОСУВАННЯ В ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ПРИЙНЯТТЯ РІШЕНЬ

1.1 Огляд і аналіз інтелектуальних систем прийняття рішень

1.1.1 Загальні поняття щодо систем прийняття рішень

Потреба в прийнятті рішень при вирішенні будь-яких питань як в організаційному управлінні, так і в приватному житті, зумовлює постійний розвиток спеціальних комп’ютерних систем, які забезпечують їх підтримку.

Такі системи пройшли свій розвиток від управлінських систем, які мало чим відрізнялися від звичайних систем обробки даних, до сучасних інтелектуальних систем прийняття рішень на основі хмарних обчислень і потужних методів і алгоритмів.

Системи підтримки прийняття рішень (СППР) виникли на початку 70-х років минулого століття завдяки дальшому розвитку управлінських інформаційних систем і являють собою системи, розроблені для підтримки процесів прийняття рішень менеджерами для вирішення слабоструктурованих проблем. Вже до кінця 70-х років XX століття ряд компаній розробили інтерактивні (діалогові) інформаційні системи, що використовували дані та моделі для допомоги менеджерам при прийнятті рішень [1]. Такі комп’ютерні системи стали називати системами підтримки прийняття рішень, а в зарубіжній літературі ці системи відомі за назвою Decision Support Systems. В літературі існує велика кількість визначень терміну «Система підтримки прийняття рішень», зокрема, в [1] наведені наступні визначення.

Система підтримки прийняття рішень є інтерактивною системою, яка забезпечує користувачеві легкий доступ до моделей і даних для того, щоб

підтримати процес прийняття рішень стосовно слабоструктурованих і неструктурованих завдань.

Відомі й інші означення, зокрема: «СППР — це такі системи, які ґрунтуються на використанні моделей і процедур з оброблення даних та думок, що допомагають керівникові приймати рішення»; «СППР — інтерактивні автоматизовані системи, що допомагають особам, які приймають рішення, використовувати дані і моделі для розв'язання неструктурованих і слабоструктурованих проблем»; «СППР — комп'ютерна інформаційна система, використовувана для підтримки різних видів діяльності під час прийняття рішень у ситуаціях, де неможливо або небажано мати автоматичну систему, яка повністю виконує весь процес створення рішень» [1].

Така велика кількість визначень пов’язане з широким діапазоном функціональних можливостей, типів і розмірів СППР. Проте, виділимо їх основні риси: це комп’ютерні системи, які допомагають при прийнятті рішень в умовах слабоструктрованих та неструктурованих ситуацій.

Найбільш вданим визначенням, на наш погляд, є наступне. СППР це – інтерактивна комп'ютерна система, призначена для підтримки різних видів діяльності під час прийняття рішень стосовно слабоструктурованих і неструктурованих проблем.

1.1.2 Інтелектуальні системи прийняття рішень

Розвитку сучасних СППР притаманна наявність в їх структурі сучасних інформаційних технологій, таких як сховища даних, OLAP-аналіз, Data Mining, Big Data, інтелектуальні програмні агенти, хмарні технології, технології групової взаємодії тощо. Важливою рисою таких систем є їх інтелектуалізація, яка дає підстави СППР називати інтелектуальними системами прийняття рішень.

В [1], [2] описано еволюцію розвитку СППР та їх інтелектуалізації. Відповідно [1], [2], СППР мають три покоління розвитку, етапи їх розвитку на шляху їх інтелектуалізації показані на рисунку 1.1 [2].

Рисунок 1.1 – Покоління СППР [2]

До основних відмінностей інтелектуальних систем прийняття рішень від інших СППР належать наступні [2]:

– використання для підтримки рішень знань спеціалістів (експертів);

– відтворення методами штучного інтелекту усвідомлених розумових зусиль людини;

– забезпечення розв’язання проблеми прийняття рішень в управлінні, коли максимально ефективно використовуються можливості як людей-експертів, осіб що приймають рушення (ОПР), так і програмно-технічних засобів.

Для визначення поняття «Інтелектуальна система прийняття рішень» (ІСПР) необхідно визначити тлумачення терміну «інтелект» [3]. Відповідно [4], термін «інтелект (intelligence)» походить від латинського слова intellectus, що означає розум, розсудливість, розуміння, вміння розмірковувати раціонально.

Термін штучний інтелект (artificial intelligence, AI) вперше був запропонований в 1956 році Джоном Мак-Карті на семінарі з аналогічною

назвою, присвяченому розробці методів розв’язання логічних задач, що пройшов в Дартмутському коледжі США [2].

Штучний інтелект (ШІ) зазвичай тлумачать як властивість автоматизованих (комп’ютерних) систем, створених на підґрунті відповідних математичних методів і моделей, брати на себе окремі функції інтелекту людини, зокрема, генерувати й обґрунтовувати рішення на основі формалізованого у вигляді бази знань та моделей раніше отриманого досвіду, а також аналізу та врахування зовнішніх збурюючих впливів і невизначеності. Отже, інтелектом можна називати здатність вирішувати задачі шляхом набуття, запам’ятовування та цілеспрямованого поглиблення (уточнення) знань у процесі навчання на підґрунті набутого досвіду та здатності до адаптації відповідно до зміни обставин.

Відповідно, характерними рисами інтелекту, які мають проявлятись у процесі розв’язування задач, є здатність до навчання, узагальнення, накопичення досвіду (знань, навичок) й адаптація до зміни умов.

Тоді, під поняттям інтелектуальної системи прийняття рішень (ІСПР) будемо розуміти людино-машинні інтерактивні системи, що допомагають відповідальній та компетентній особі обґрунтовувати і приймати раціональні управлінські рішення, в процесі вироблення яких задіяні штучні підсилювачі інтелекту [3].

За визначенням [2], інтелектуальна система прийняття рішень (ІСПР) – інтерактивна комп’ютерна система, призначена для підтримки прийняття рішень у різних сферах діяльності стосовно слабоструктурованих і неструктурованих проблем, яка ґрунтується на використанні моделей і процедур з обробки даних та знань на основі технологій штучного інтелекту.

Сфери застосування ІСПР різноманітні [5]. Зокрема, в [6] автори пропонують поділити задачі, для яких розробляють ІСПР на два підкласи з точки зору їх функціонування. Це клас задач, для яких суттєве значення понять (властивостей). Сюди відносяться задачі діагностики захворювань,

розпізнавання образів, класифікація явищ на основі збору даних тощо. До іншого класу відносять задачі, для яких не є суттєвим значення понять, а скоріше, їх семантика або частотність використання термінів в тексті тощо. Сюди можна віднести кластеризацію інформаційних ресурсів, класифікацію текстів, інтелектуальні пошукові системи, реферування та анотування текстових документів.

Також ІСПР доцільно розглядати в залежності від рівня інтелектуалізації, зокрема, в [2] автори пропонують поділити на два класи:

− системи, що відтворюють усвідомлені розумові зусилля людини (статичні детерміновані або стохастичні системи);

− системи, що відтворюють підсвідомі розумові дії людини (еволюційні системи з нейротехнологіями і генетичними алгоритмами) (рис. 1.2).

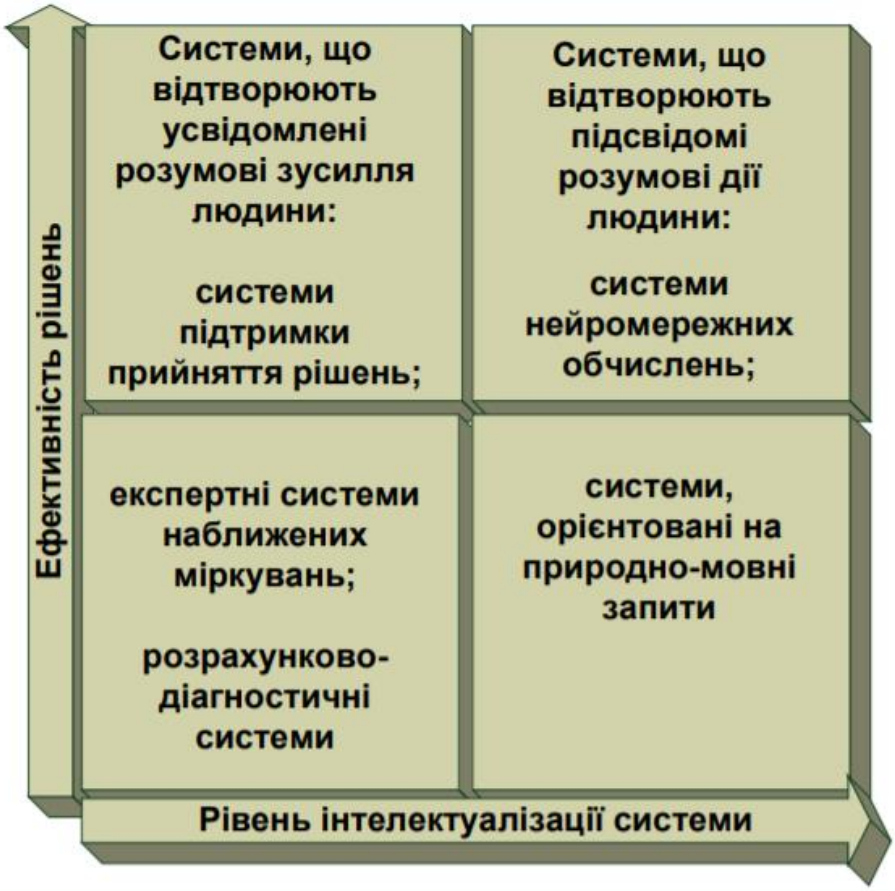

Рисунок 1.2 – Класифікація ІСПР за видами відтворених знань людини [2]

До першого класу відносять СППР, експертні системи наближених міркувань та розрахунково-діагностичні системи. Другий клас: системи нейромережевих обчислень та системи, орієнтовані на природно-мовні запити. На сьогодні використання систем другого класу не достатньо поширена у зв’язку із складністю реалізації алгоритмів. Саме тому питання застосування алгоритмів еволюційних моделювання є досить актуальним.

1.1.3 Структура ІСПР

Не дивлячись на те, що на ринку існує велика кількість СППР, які можна класифікувати за різними ознаками, усі вони мають спільні компоненти [1]: підсистема інтерфейсу користувача, підсистема керування базою даних і підсистемою базою моделей [1]. Ці підсистеми утворюють основу класичної структури СППР, запропонованої ще класиками розвитку СППР (Robert H. Bonczek, Clyde W. Holsapple, Andrew B. Whinston), яку вони визначили в своїй праці [6] щодо теоретичних основ проектування СППР.

В [2] до основних підсистем СППР віднесені підсистема розв’язання задач (Problem-Processing System), підсистема представлення (Presentation Sysytem), мовна підсистема (Language Sysytem), підсистема знань (Knowledge System). Даний підхід до аналізу компонентів СППР свідчить про наявність інтелектуальних компонент, які присутні в сучасних ІСПР.

В [3] Вітлінський В.В. пропонує розглядати структуру ІСПР за наступним підходом.

По-перше, традиційні для інтелектуальних систем модулі:

- бази даних і знань;
- бази моделей;
- механізми висновків;
- система накопичення та актуалізація знань;
- блок пояснень;

− організації діалогової взаємодії з користувачем.

По-друге, такі підсистеми:

− аналізатор проблемної ситуації;

− імітаційні моделі проблемної ситуації;

− математичні методи та моделі аналізу та прогнозування;

− різні види інтерфейсу (образного, текстового, мовного, у вигляді графіків і діаграм) з ОПР.

По-третє, підсистему інтелектуальних технологій, яка дозволяє на підставі використання математичних моделей і методів створювати гібридні інтелектуальні системи і має такі складові:

− експертну систему, що ґрунтується на символьному представленні знань у вигляді правил і фактів;

− систему, що ґрунтується на нечіткій логіці;

− систему штучних нейронних мереж.

По-четверте, модуль генетичних алгоритмів — тобто підсистему пошуку раціональних рішень, що містить як теоретико-аналітичні методи, так і евристичні методи розв’язування задач, зокрема, оптимізаційних.

Об’єднання цих компонент дозволяє створювати такі інтелектуальні системи прийняття рішень, які здатні вирішувати значний відсоток складних проблем, які потребують застосування складних методів і алгоритмів.

1.2 Теоретичні основи еволюційних алгоритмів

1.2.1 Поняття еволюційних обчислень і алгоритмів

Перш ніж розглядати поняття еволюційних алгоритмів, розглянемо термін «алгоритм» як загальне поняття. В [3], [8] автори пропонують під алгоритмом розуміти точну (чітку) інструкцію для виконання у визначеному порядку низки операцій для розв’язування певної задачі з

деякого класу (множини) задач. У математиці та кібернетиці клас задач певного типу вважається вирішеним (розв’язаним), коли для їх розв’язування встановлений (побудований) алгоритм. Побудова алгоритму для задач певного типу (класу) пов’язана в низці випадків з витонченими, складними розмірковуваннями, що вимагають винахідливості та високої кваліфікації. Такого роду креативна діяльність потребує участі людини. Задачі, що пов’язані зі знаходженням (побудовою) алгоритму розв’язування певного класу задач будемо називати інтелектуальними задачами.

Еволюційні обчислення – термін, який зазвичай використовується для загального опису алгоритмів пошуку, оптимізації або навчання, що базується на формалізованих принципах природнього еволюційного процесу [9].

Еволюційні методи призначені для пошуку кращих рішень й ґрунтуються на статистичному підході при дослідженні ситуацій та ітераційному приближенні до шуканих станів систем. На відміну від точних методів математичного програмування, еволюційні методи дозволяють знаходити рішення, що близькі до оптимальних, за прийнятний час, а також, на відміну від відомих евристичних методів оптимізації, характеризуються набагато меншою залежністю від особливостей області застосування й у багатьох випадках забезпечують кращий ступінь наближення до оптимального розв’язку.

Основною перевагою еволюційних методів оптимізації є можливість вирішення багатомодальних (таких, які мають декілька локальних екстремумів) задач з великою розмірністю за рахунок поєднання елементів випадковості та детермінованості так само, як це відбувається в природному середовищі [9].

Вважається, що наукове направлення «еволюційні обчислення» є основним підходом до моделювання механізмів живої природи. Воно включає до себе три основні напрямки фундаментальних досліджень:

генетичні алгоритми, еволюційне моделювання (іноді зустрічається назва «еволюційні стратегії») та еволюційне програмування [10], [11].

Наразі під назвою еволюційні алгоритми мають на увазі усі обчислювальні і оптимізаційні методи, в основу яких покладена еволюційний підхід: еволюційні стратегії, еволюційне програмування, генетичні алгоритми, генетичне програмування. Ці методи відрізняються здебільшого представленнями можливих розв’язків задачі. Так, в еволюційних стратегіях хромосоми представлені векторами дійсних чисел, в генетичних алгоритмах – векторами двійкових чисел, в еволюційному програмуванні – скінченними автоматами, а в генетичному програмуванні – деревами. Оператори селекції еволюційних алгоритмів не залежать від представлення розв’язків, адже вони використовують лише значення функції пристосованості. Тому різниця між цими видами алгоритмів невелика, все залежить від найкращої стратегії для конкретної поставленої задачі. Під конкретну задачу підбирають відповідні представлення розв’язків задачі, а також для оперування ними – оператори мутації та схрещування [12].

Останніми роками активно розвивається такий напрям наукових досліджень як природні обчислення (Natural Computing), який об’єднує математичні методи на основі принципів природних механізмів прийняття рішень. До таких методів належать наступні еволюційні алгоритми: імітація самоорганізації мурашиної колонії, оптимізаційні алгоритми «бджолиногорою» тощо [12].

1.2.2 Види еволюційних алгоритмів

1.2.2.1 Еволюційне програмування

Еволюційне програмування винайдене Лоуренсом Фогелем у 1960-1965 роках [13]. Він помітив можливість ще одного, альтернативного

підходу до проблем штучного інтелекту – моделювання не кінцевого результату еволюції, а моделювання самого процесу еволюції як засобу «відпрацювання розумної поведінки» і можливостей передбачення різних явищ у середовищі. Л. Фогель виконав низку експериментів, у яких скінченні автомати представляли собою особин у популяції розв’язків задачі. У цих скінченних автоматах було передбачено використання символів у цифрових послідовностях, які, еволюціонуючи, ставали все придатнішими до розв’язування поставленої задачі.

В еволюційному програмуванні популяцію скінченних автоматів розглядають в експериментальному середовищі та отримують певну послідовність символів. Для кожного автоматабатька виконується процедура звірення кожного наступного символу з відповідним йому прогнозованим вихідним символом з автомата і оцінюється значення функції втрат для цього виходу. Після останнього прогнозу обчислюється «життєздатність» такого автомату чи програми. Автоматинащадки утворюють випадковою мутацією автоматів-батьків і теж оцінюють. Найкращі автомати 247 відбираються для наступного покоління, і процес повторюється. У разі необхідності прогнозу нових символів, нове спостереження додається до старих.

Таким чином, в еволюційному програмуванні, на відміну від генетичних алгоритмів, моделюють еволюцію більш як процес пристосувальної поведінки особин популяції або виду, аніж процес адаптації генів [13].

1.2.2.2 Основи генетичних алгоритмів

Для роботи генетичного алгоритму (ГА) використовують віртуальну популяцію, де гени кожної окремої особини є частним рішенням поставленого завдання. Число генів у особини залежить від числа параметрів завдання. В результаті оцінювання популяції кожної особини

ставиться у відповідність деяка величина, яка називається пристосованістю і показує, наскільки успішно дана особина вирішує це завдання, тобто наскільки її гени відповідають поставленим умовам. Більш пристосовані особини схрещуються, з їх нащадків формується нова популяція, члени якої оцінюються, потім схрещуються і т.д. В ході схрещування двох особин за рахунок застосування генетичних операторів відбувається обмін генетичною інформацією, і отримані нащадки мають як властивостями першого батька, так і властивостями другого.

В результаті роботи ГА виходить популяція, яка містить особину, гени якої краще генів інших особин відповідають необхідним умовам. Дана особина і буде знайденим за допомогою ГА рішенням. Слід зазначити, що знайдене рішення може і не бути найкращим, однак воно може бути близько до оптимального. Відмінною особливістю ГА є те, що обчислювальна складність алгоритму мало залежить від складності завдання. Значення мають: вид цільової функції, кількість параметрів і, якщо є, область обмежень.

Основною областю застосування ГА є завдання оптимізації. До теперішнього часу генетичні алгоритми були використані для вирішення наступних проблем:

- екстремальні завдання;
- NP-повні проблеми ("Завдання комівояжера" і SAT-проблеми);
- складання розкладів;
- завдання про розміщення;
- апроксимація функцій;
- побудова мінімальних діагностичних тестів для задач розпізнавання;
- налаштування і навчання штучних нейронних мереж;
- ігрові стратегії;
- моделювання штучного життя.

Найчастіше ГА використовують у машинному навчанні [14].

Ідея генетичних алгоритмів дістала подальшого розвитку та стала підґрунтям такого напрямку в теорії штучного інтелекту, яким є еволюційне програмування. На даний час створена відповідна математична теорія, запропонована низка модифікацій алгоритмів. Генетичні алгоритми дозволяють ефективно здійснити вибір кращих варіантів рішень у тих випадках, коли складним чи неможливим є використання класичних методів оптимізації [3].

1.2.2.3 Основи алгоритмів Еволюційних стратегій

Еволюційні стратегії [13] у багатьох аспектах подібні і до генетичних алгоритмів, і до еволюційного програмування, бо в них теж імітують процеси природної еволюції. Однак, вони мають суттєву відмінність на прикладному рівні. У той час як генетичні алгоритми створені для оптимізації дискретних або цілочислових розв’язків задач, еволюційні стратегії застосовують для неперервних значень, які є типовішими для експериментальних задач.

Основні відмінності еволюційних стратегій від інших оптимізаційних методів і еволюційних алгоритмів:

– пошук від однієї популяції до іншої на противагу від однієї особини до іншої;

– використання даних про саму функцію, а не її похідних;

– використання ймовірнісних, а не детермінованих, методів за переходу від популяції до популяції;

– кодування розв’язків векторами дійсних чисел;

– зосередження уваги на вплив генетичних операторів на зміни фенотипу;

– адаптивний крок мутації – крок мутації еволюціонує разом з розв'язком, оскільки цей параметр пов’язаний з хромосомами;

– невелика різниця між батьками та нащадками в процесі схрещування, що обумовлено сильною взаємопов'язаністю – невеликі зміни у перших відображаються невеликими змінами в останніх.

Технологія еволюційних стратегій була винайдена І. Рехенбергом, а згодом розвинута Г.- П. Швефелем та іншими вченими [13].

1.2.2.4 Алгоритми мурашиних колоній

Мурашина колонія – це один із нових перспективних метаевристичних алгоритмів (різновид евристики рою), який імітує колективну поведінку мурашок для досягнення певної цілі, спільної для всього мурашника. Ідея алгоритму мурашиної колонії, натхнена статтею про колективну поведінку аргентинських мурах, вперше висловлена А.Колорні, М.Доріго і В.Маньєццо у 1991-1992 роках при розв’язанні задачі про оптимальні шляхи у графах відповідно до того, як мураха відшукує найкращий шлях між мурашником та деяким харчовим ресурсом. Так само вона застосовна до задачі комівояжера, задач оптимізації маршрутів вантажних перевезень, задачі розфарбування графів, оптимізації мережених графіків, задач календарного планування тощо.

Один-єдиний мураха має обмежені когнітивні можливості, але весь мурашник, діючи спільно в певному напрямку, може досить швидко і ефективно знайти оптимальний шлях. Відтоді ідея мурашиної колонії поширилася на значно більше коло числових задач, а також всередині цієї ідеї з’явилися численні відгалуження, які застосовують різні аспекти поведінки мурах. Колонію мурах можна розглядати як багатоагентну систему, у якій кожен агент «мурашка» функціонує автономно за досить простими правилами. На противагу примітивній поведінці окремих агентів, поведінка всієї системи виявляється досить розумною.

Перший мураха (припустимо, що він бігає навколо мурашника випадковим чином), який знаходить ресурс харчування яким-небудь

шляхом, повертається до мурашника, залишаючи на цьому шляху відповідний слід феромону, який приваблює інших мурах, що проходять повз, слідувати саме цим шляхом. Оптимальніший шлях скоро набуває більшої концентрації «запаху» феромонів, ніж інші, чим виявляється ще привабливішим. Поки цей шлях залишається найкращим, легкі сліди феромонів на інших шляхах поступово вивітрюються.

Розподіл феромонів по середовищу пересування мурах є своєрідною динамічно змінюваною глобальною пам'яттю всього мурашника. «Мурашка» на деякому вузлі графа має прийняти рішення, яким шляхом далі слідувати, при цьому він користується параметрами «привабливості» та «сліду феромонів» на тій чи іншій ланці. Будь-який мурашка у фіксований момент часу може зчитувати або вносити зміни лише до єдиної комірки глобальної пам'яті [12], [15], [16].

1.2.2.5 Алгоритми бджолиного рою

Алгоритм «бджолиного рою» – це один різновид евристик рою, відносно молодий метод знаходження глобального екстремуму (максимуму або мінімуму) складної багатовимірної функції (у термінах алгоритму – цільової функції). Як видно з назви, цей алгоритм використовує особливості поведінки рою бджіл-розвідників під час пошуку нектару.

Припустимо, що бджоли живуть у багатовимірному просторі, де кожна координата означає один параметр функції, яку необхідно оптимізувати. Нехай глобальний екстремум функції – це місце, де зосереджено найбільше нектару, причому таке місце у довкіллі вулика – єдине, а на інших галявинах нектару однозначно менше. Знайдена кількість нектару – це значення цільової функції у даній точці простору. Деякий розвіданий розв'язок представлений бджолою, яка зберігає параметри місця, де можна добути нектар. Нові бджоли знаходять ділянки

простору, де значення цільової функції оптимальні, причому окіл випадкового пошуку наступними бджолами від вже розвіданих точок з найкращими показниками цільової функції динамічно зменшується (тільки не можна зменшувати окіл занадто швидко, бо є шанси застрягнути на локальному екстремумі). Найуспішніша бджола запам'ятовується як проміжний розв'язок, і процес повторюється, поки не буде досягнутий відповідний критерій зупинки.

Цей тип алгоритмів вже зарекомендував себе у численних застосуваннях, наприклад, навчання нейронних мереж розпізнавання образів, оптимізація ваг у перцептронах, формування розкладу робіт на виробництві, кластеризація даних, багатоцільова оптимізація тощо [12], [15], [16].

1.2.2.6 Алгоритми інтелектуального пошуку (пошук табу)

Методи інтелектуального пошуку, зазвичай називають пошуком табу. Вони набули широкого поширення і визнання в рішенні практичних оптимізаційних задач. Додатки цих методів стрімко розширюються в таких областях, як управління ресурсами, проектування процесів, логістика, планування і загальна комбінаторна оптимізація. Поєднання з іншими методами, як евристичними, так і алгоритмічними, також дає продуктивні результати.

Метод пошуку табу (TS) є метаевристікой, яка управляє процедурою локального евристичного пошуку з метою глобального дослідження простору рішень. В останні роки стрімко розширюється область додатків методу пошуку табу в оптимізації. Процедури TS, що включають основні концепції та гібридні схеми, що поєднують ці концепції з іншими евристичними і алгоритмічними методами, успішно застосовувалися в різних практичних завданнях.

Пошук табу заснований на припущенні, що процес вирішення завдань, що претендує на звання «інтелектуального», повинен включати адаптивну пам'ять і ретельне дослідження ситуації.

Пошук табу призначений для знаходження нових і більш ефективних шляхів отримання поліпшених рішень і для визначення відповідних принципів, які можуть розширити основи інтелектуального пошуку. Як це зазвичай трапляється, з'являються нові поєднання базових ідей, що призводить до поліпшеним рішенням і найкращим практичних результатів. Це робить TS привабливою областю для досліджень і використання в практичних задачах.В останнім часом з'явилися гібридні схеми, що поєднують пошук табу не тільки з локальним пошуком, але і з еволюційними алгоритмами і імітацією відпалу, які і самі по собі є глобальними пошуковими процедурами. Застосування в практичних завданнях показує, що ці підходи мають велике майбутнє [12].

1.2.2.7 Роєві алгоритми

Останнім часом все більш популярними стають алгоритми оптимізації, які мають назву роєвих алгоримів [15], [17], [18]. Ідея даних методів запозичена з соціальної поведінки деяких видів тварин, наприклад, зграї птахів, косяка риб або стада копитних. Дослідження показали ефективність ройових алгоритмів і доцільність їх застосування при розв’язуванні задач як безумовної, так і умовної оптимізації функцій багатьох змінних. Постійно пропонуються нові чи модифіковані алгоритми для поліпшення продуктивності методів даної групи або для розширення сфери їх застосування.

Роєві алгоритми, засновані на використанні популяції і працюють з набором потенційних розв’язків. Кожен розв’язок поступово поліпшується і оцінюється, таким чином, кожен потенційний розв’язок впливає на те, як будуть покращені інші розв’язки. Більшість популяційних методів

запозичили таку концепцію з біології: процес пошуку найкращого розв’язку імітує деякий природний процес або поведінку певних видів тварин, причому враховуються їх видові особливості.

Найбільш відомими серед біоінспірованих алгоритмів є наступні [17]:

- метод пошуку зозулі (Cuckoo Search Algorithm, CSA);
- метод кажанів (Bat Algorithm, BA);
- метод світлячків (Firefly Algorithm, FFA);
- метод зграї вовків (Wolf Pack Search, WPS).

Перераховані методи спочатку були розроблені для розв’язування задач однокритеріальної безумовної оптимізації з дійсними змінними, але можуть бути модифіковані і для розв’язування задач безумовної та умовної оптимізації, а також для задач дискретної оптимізації. Кожен із наведених алгоритмів імітує характерну поведінку певного виду тварин: CSA – спосіб відкладання яєць зозулями, BA – ехолокацію кажанів, FFA – випромінювання від світлячків, WPS – процес полювання зграї вовків. Початкова популяція потенційних розв’язків генерується випадковим чином і далі шукається оптимальний розв’язок в процесі свого розвитку.

На відміну від еволюційних алгоритмів, тут не використовуються генетичні оператори. У ройових алгоритмах особини (частинки) переміщуються в гіперпросторі в процесі пошуку розв’язків і враховують успіхи своїх сусідів. Якщо одна частинка бачить хороший (перспективний) шлях (в пошуках їжі або захисту від хижаків), то інші частинки здатні швидко піти за нею, навіть якщо вони перебували в іншому кінці рою. З іншого боку, в рої для збереження досить великого простору пошуку повинні бути частинки з долею випадковості в своїй поведінці [15], [17], [18].

1.3 Огляд напрямів застосування еволюційних алгоритмів в програмних продуктах

Об’єднання описаних вище алгоритмів (інструментарію) дозволяє створити інтелектуальні системи, які здатні вирішувати значний відсоток складних проблем (слабко структурованих і неструктурованих), які зазвичай потребують застосування різнорідних видів інструментальних засобів (як традиційних, так і інтелектуальних систем).

Зокрема, генетичні алгоритми можна застосовувати для підбору вагових коефіцієнтів і топології нейронної мережі, а також для формування бази правил і функцій належності систем на нечіткій логіці. Нейронні мережі дозволяють обирати відповідні параметри для генетичних алгоритмів (параметри схрещування та мутації). Окрім того, генетичні алгоритми дозволяють налаштовувати параметри нечітких множин та відповідні коефіцієнти, що визначають раціональну швидкість навчання нейронних мереж.

Ведуться роботи щодо синтезу штучних нейронних мереж, експертних систем і генетичних алгоритмів.

Дослідження показали, що в теорії інтелектуальних систем прийняття рішень на даний час відносно мало уваги приділяється урахуванню системних характеристик аналізованих варіантів рішень, зокрема, таких як маневреність, стійкість, гнучкість, адаптивність тощо, що теж відкриває простір для подальших наукових досліджень. Математичні моделі слабко структурованих процесів і об’єктів, що функціонують в умовах невизначеності, завжди містять у собі неповністю визначені поведінкові характеристики людей.

На сьогоднішній день існує ряд прикладних програмних продуктів, в яких реалізовано інструментарій еволюційних алгоритмів. Залежно від сфери використання, ступеня автономності та призначення, їх можна класифікувати, зокрема, наступним чином:

− спеціалізоване програмне забезпечення – створені для розв’язування вузького кола прикладних задач. Нескладні для освоєння та користування, але обмежені для використання широким колом користувачів через свою вузьконаправленість;

− додатки до математичних та аналітичних пакетів – через них надаються можливості розв'язувати широке коло задач, але більшість математичних пакетів не є вільно поширюваними та їх використання потребує певного рівня знань та навичок від користувачів;

− фреймворки – є безкоштовним програмним забезпеченням. За їх допомогою користувач може будувати власні програмні засоби на основі існуючого коду, але це також потребує певних знань і навичок у програмуванні.

Розглянемо приклади окремих програмних продуктів, які відносяться до кожного з наведених класів.

Спеціалізоване програмне забезпечення (ПЗ):

1. NeuroShell Trader − ПЗ для створення торгової системи. Це інструмент, за допомогою якого користувач створює торгові моделі, поєднуючи штучний інтелект і традиційні методи. Можна будувати моделі для акцій, товарів, FOREX, індексів, та ін. Можна створювати моделі для фондових бірж у всьому світі, таких як NYSE (New York Stock Exchange Index), FTSE (Financial Times Stock Exchange Index), DAX (Deutscher Aktienindex (German stock index)) та ін. Створені моделі автоматично тестуються та надають сигнали-прогнози з надходженням нових даних;

2. StrategyQuant − потужна платформа для розробки торгових систем для будь-яких ринків та часових періодів. Не потребує безпосереднього програмування та автоматично тестує згенеровані стратегії;

3. Genetic System Search for Technical Analysis – програмний засіб, використання якого допомагає користувачеві визначати правила роботи та розробляти власні інвестиційні торгові системи.

Додатки та надбудови до математичних та аналітичних пакетів:

1. XL BIT – додаток до MS Excel, універсальний інструмент для оптимізації та прогнозування. До задач, які можна розв’язування за його допомогою відносяться опрацювання зображень, складання розкладів, вибір акцій, розпізнавання підозрюваних, управління задачами, торгівля на Forex. Особливостями даного програмного засобу є те, що у роботі з еволюційним алгоритмом, зокрема генетичними, можна використовувати до 100 популяцій, три методи схрещування на вибір та два види масштабування функції пристосовуваності, доступна можливість налаштування рівнів схрещування та мутацій під час роботи з програмою та відстежування значення змінних, а кількість оптимізованих змінних залежить тільки від швидкодії та об'єму запам’ятовуючих пристроїв комп'ютера; є можливість налаштування випадкових вихідних даних та генерації детального звіту і графіка пристосовуваності;

2. Excel Solver – надбудова Microsoft Excel. За допомогою засобу пошуку рішення знаходять розв’язку оптимізаційних задач, зокрема за еволюційним методом.

3. Global Optimization Toolbox є додатком для програмного пакету MATLAB. За його допомогою знаходять глобальні розв'язки многокритеріальних задач. Цей засіб можна використовувати для розв’язування оптимізаційних задач, в яких цільова функція є скінченною, нескінченною, стохастичною, не містить похідних, або включає симуляції чи закриті функції з неявно заданими значеннями для параметрів у налаштуваннях. До особливостей даного інструменту можна віднести вибір оптимального компоненту шляхом використання змішано-цілочислового генетичного алгоритму; обмежену мінімізацію та многокритеріальну оптимізацію, можливість налаштування генетичного алгоритму та застосування гібридних схем.

Фреймворки. Фреймворк Pyevolve – розроблений з метою побудови цілісного фреймворку для генетичних алгоритмів. Його особливостями є мультиплатформне використання, просте для використання API, надання

можливості користувачеві створювати нові представлення та генетичні оператори і використовувати еволюційну статистику; висока швидкодія; багатий набір стандартних функцій та параметрів за замовчуванням; відкритий код.

Перелік зазначених вище програмних засобів не є вичерпним, його можна доповнити такими програмними засобами: Evolver, Excel Genetic Algorithm Tool, GeneHunter.

Крім того, в Інтернеті також можна знайти багато корисних сайтів, присвячених реалізації еволюційних алгоритмів. Зокрема, на сайті www.basegroup.ru запропоновано бібліотеку компонентів «Delphi GeneBase», на сайтах www.generation5.org та www.sourceforge.net є достатня кількість прикладів реалізації генетичного алгоритму різними мовами програмування [19].

1.4 Постановка задачі

Метою атестаційної роботи є дослідження алгоритмів еволюційного моделювання та їх використання в інтелектуальних системах прийняття рішень.

- здійснити аналіз теоретичних відомостей щодо інтелектуальних систем прийняття рішень;
- здійснити аналіз відомих алгоритмів еволюційного моделювання;
- провести огляд напрямів застосування еволюційних алгоритмів в програмних продуктах;
- на підставі проведеного аналізу сформулювати вимоги до інтелектуальної системи прийняття рішень;
- розробити структуру ІСПР;
- розробити комплекс моделей на основі еволюційних алгоритмів для інтелектуальної системи прийняття рішень;

– здійснити практичну реалізацію системи.

Висновки до розділу 1.

В розділі 1 було досліджено теоретичні відмінності щодо СППР та СППР та визначено, що система підтримки прийняття рішень є інтерактивною системою, яка забезпечує користувачеві легкий доступ до моделей і даних для того, щоб підтримати процес прийняття рішень стосовно слабоструктурованих і неструктурованих завдань. ІСПР – інтерактивна комп'ютерна система, призначена для підтримки прийняття рішень у різних сферах діяльності стосовно слабоструктурованих і неструктурованих проблем, яка ґрунтується на використанні моделей і процедур з обробки даних та знань на основі технологій штучного інтелекту.

Було досліджено, що в склад ІСПР входять наступні модулі: бази даних і знань, бази моделей, механізми висновків, система накопичення та актуалізація знань, блок пояснень, організації діалогової взаємодії з користувачем.

Здійснений огляд теоретичних відомостей щодо еволюційних обчислень, суть яких полягає в тому, що їх реалізація заснована принципах природнього еволюційного процесу. Здійснено огляд наступних видів алгоритмів: еволюційне програмування, еволюційні стратегії, генетичні алгоритми, алгоритми мурашиних колоній, алгоритми бджолиного рою, алгоритми інтелектуального пошуку (пошук табу), роєві алгоритми.

Проведений аналіз застосування еволюційних алгоритмів, яке залежно від сфери використання, ступеня автономності та призначення який розглядають в наступних напрямках: спеціалізоване програмне забезпечення, додатки до математичних та аналітичних пакетів, фреймворки.

Розроблено постановку завдання, яке включає такі завдання: здійснити аналіз теоретичних відомостей щодо інтелектуальних систем прийняття рішень, здійснити аналіз відомих алгоритмів еволюційного

моделювання, провести огляд напрямів застосування еволюційних алгоритмів в програмних продуктах, на підставі проведеного аналізу сформулювати вимоги до інтелектуальної системи прийняття рішень, розробити структуру ІСПР, розробити комплекс моделей на основі еволюційних алгоритмів для інтелектуальної системи прийняття рішень, здійснити практичну реалізацію системи.

2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ЕВОЛЮЦІЙНОГО МОДЕЛЮВАННЯ

2.1 Основні положення еволюційних алгоритмів

Існує велика кількість алгоритмів, які відносяться до класу еволюційних, а саме [15], [16],[17],[20]:

- генетичні алгоритми (genetic algorithm);
- генетичне програмування (genetic programming);
- еволюційна стратегія (evolution strategy);
- еволюційне програмування (evolutionary programming);
- алгоритми диференціальної еволюції (differential evolution);
- алгоритми інтелектуального пошуку (tabu search);
- популяційні алгоритми на основі живої природи;
- алгоритми на основі неживої природи;
- алгоритми, інспіровані людським суспільством тощо.

В математичну основу цих алгоритмів покладено використання базових принципів теорії біологічної еволюції, а саме відбору, мутації і відтворення особин. Еволюційні алгоритми є частиною більш широкої технології м'яких обчислень (soft computing), вони включають в себе нечітку логіку Заде (L. A. Zadeh), нейронні мережі, імовірнісні судження тощо. Вони можуть використовуватися у різних комбінаціях або самостійно, доповнювати один одного і призначені для створення інтелекту.

Дослідження літератури показали, що найбільш уживаними є генетичні алгоритми, в той час, як для вирішення різних задач можна застосовувати й інші перелічені вище.

Розглянемо основні положення еволюційних алгоритмів [15]. Не дивлячись на те, що кожен із них має свої особливості, є певні риси, притаманні цьому класу алгоритмів.

Операцію одноразового обчислення значень цільової функції *f(X)*, яка має обмеження у вигляді функцій *E (X), G (X)*, та їх можливих похідних зазначених функцій в певній точці, де $X \in D$ називають випробуванням.

Оскільки кожне окреме випробування вимагає значних витрат комп'ютерного часу, алгоритми оптимізації повинні бути побудовані таким чином, щоб була здійснена мінімальна кількість випробувань.

Агентом називають наближення до вирішення і позначають s_i, $\in [1:|S|], |S| \geq 1$, де S – позначає множину агентів (популяцію агентів), використовується безліч агентів (популяція агентів), $|S|$ – загальну кількість агентів.

Розглянемо найбільш поширений підхід до ітераційного алгоритму оптимізації. Алгоритм має наступні кроки.

Крок 1. Ініціалізація алгоритму, тобто задаємо початкове значення лічильника числа ітерацій $t = 0$, агенти в початковому положенні мають вигляд: початкові положення агентів $X_i(0), i \in [1:|S|]$. Також на цьому кроці задаються значення вільних параметрів алгоритму.

Крок 2. Застосування пошукових (міграційних) операторів алгоритму до поточних станів агентів $X_i(t) = X_i^t = X_i$. X; (t) = x: = X ;, в результаті чого вони набувають нові положення $X_i(t+1) = X_i^{t+1} = X_i^{'}; i \in [1:|S|]$.

Крок 3. Здійснюємо перевірку виконання умов завершення ітерацій. Якщо ці умови не виконані, вважаємо, що $t = t + 1$ повертаємося до кроку 2. В іншому випадку отриманий результат, тобто знайдений стан агентів, розглядаємо як наближене рішення задачі.

Загалом, міграційні оператори обчислюють нові стани агентів відповідно деякої функції $\Phi(\cdot)$:

$$X_i^{t+1} = \Phi\left(X_k^\tau, f\left(X_k^\tau\right), E\left(X_k^\tau\right), G\left(X_k^\tau\right), k \in [1:|S|], \tau \in [0:t]\right),$$
$$i \in [1:|S|], t+1 \le \hat{t}. \quad (2.1)$$

В даній функції (2.1) $\hat{t}$ розглядають як кількість ітерацій, які використали в процесі пошуку. Тлумачення функції полягає в тому, що новий стан агенту s_i усіма попередніми положеннями всіх агентів популяції, а також відповідними значеннями цільової функції і обмежень.

В якості наближеного рішення задачі приймається вектор $\tilde{X}^*$, який знаходиться з умови $f^* = f\left(X^*\right) = min_{t\in\left[0:\hat{t}\right], i\in[1:|S|]} f(X_i^t)$.

В ітераційних алгоритмах функція $\Phi(\cdot)$ є однією і тією ж на всіх ітераціях, виключаючи, можливо, лише деякий невеликий число початкових «розгінних» ітерацій.

У популяційних алгоритмах (population algorithms), кількість агентів більше одиниці і на кожній з ітерацій розміщуються усі агенти, виключаючи, можливо, лише деяких з них (наприклад, «кращих» агентів).

2.2 Генетичні алгоритми

Існує велика кількість способів кодування особин, а також операторів мутації, кросовера, відбору, селекції і, нарешті, методів нішування і врахування обмежень. Це свідчить, що можна запропонувати надзвичайно велике різноманіття генетичних алгоритмів, що відрізняються способами кодування, структурою, використовуваними генетичними операторами і їх параметрами тощо. Деякі орієнтири в різноманітті можливих генетичних алгоритмів дають типові алгоритми.

Розглянемо деякі з них [15], [20].

Канонічний (canonical) генетичний алгоритм Холланда має наступні характеристики:

– бінарне кодування особин;
– фіксована розрядність генів;
– постійний розмір популяції;
– відсутність проміжної популяції;
– управління популяцією методом рулеточного відбору;
– селекція особин для схрещування методом панмиксии;
– використання одноточечного кросовера і мутатора.

Простий (simple) генетический алгоритм Голдберга (D. E. Goldberg) відрізняється від канонічного тільки використання замість рулеточного відбору методу турнірного відбору.

Генетичний алгоритм Genitor Вітлі (D. Whitley) відрізняється від канонічного генетичного алгоритму своїми наступними властивостями:

– на кожній ітерації алгоритм створює одну особину-нащадка від однієї випадкової батьківської пари особин;
– особина-нащадок замінює не батьківську особину, а одну з найгірших особин популяції (у початковому варіанті алгоритму - найгіршу);
– відбір особини для проведення вказаної заміни проводиться методом рангового відбору.

Таким чином, генетичний алгоритм Genitor є прикладом реалізації підходу, в якому популяція оновлюється частинами, а не вся відразу.

Однією з переваг алгоритму Genitor є те, що він вимагає в два рази менше пам'яті комп'ютера для зберігання інформації про популяції порівняно з будь-яким генетичним алгоритмом, що використовує проміжну популяцію. З іншого боку, алгоритм Genitor має порівняно з канонічним генетичним алгоритмом значно гірші диверсифікаційні властивості, що обумовлено потенційно швидким збіднінням різноманіття

популяції за рахунок можливого тривалого збереження в ній кращих особин і швидкого виключення з неї найменш пристосованих особин.

Генетичний алгоритм CHP Эшельмана (Cross-population selection, Heterogenous recomblnation and Cataclysmic mutation, Ch. L. Eshelman) визначають його наступні основні властивості:

– серед особин-батьків та їх особин-нащадків вибирають деяке число кращих різних особин;

– для схрещування вибирають випадкову пару особин-батьків таку, що хеммингову відстань між ними не менше заданого і відстань між крайніми різними бітами також не менш заданого;

– для схрещування особин-батьків використовують поводнорідний кросовер HUX (Half Uniform Crossover), коли особини-нащадку переходить рівно половина бітів кожної з осіб-батьків;

– популяція містить всього близько 50 осіб;

– у стані стагнації застосовують катастрофічну мутацію (cataclysmic mutation), коли всі особини популяції, крім самої пристосованої, піддають сильної мутації, змінює близько третини бітів кожної особини.

Підкреслимо, що алгоритм CHP використовує мутацію тільки для подолання стагнації. В основному циклі алгоритм використовує тільки кросовер.

Гібридні (hybrid) генетичні алгоритми найчастіше засновані на комбінації генетичного алгоритма і будь-якого локального алгоритму пошуку (наприклад, градієнтного, якщо фітнес-функція диференційована). На кожній ітерації гібридного алгоритму вектори варійованих параметрів, які відповідають кожній особини-нащадку, використовують в якості стартових точок для зазначеного алгоритму локального пошуку. Знайдені в результаті цієї процедури особини замінюють в популяції стартових особин. Отриману популяцію обробляють у відповідності з цим генетичним алгоритмом. Тобто, в гібридному алгоритмі даного класу

кожна з особин досягає локального мінімуму, в околиці якого вона знаходиться, і тільки після цього піддається дії звичайних генетичних операторів. Можна вважати, що гібридний алгоритм реалізує комбінацію моделей дарвінівської і (помилкової) ламарковської еволюції, відповідно з якої особина здатна навчатися і записувати отримані навички в свій генотип, щоб потім передати їх особинам-нащадкам.

Часто в обчислювальній практиці в якості алгоритму локальної оптимізації використовують так званий алгоритм сходження на гору (hill climblng). У випадку бінарного кодування особин ідея алгоритму полягає у почерговому інвертуванні кожного біта хромосоми, обчислення пристосованості отриманих особин-нащадків і виборі найбільш пристосованою з них.

На перший погляд, гібридні алгоритми, засновані на комбінації генетичного алгоритму та алгоритму локального пошуку, зрушують баланс між інтенсивністю і широтою пошуку в бік інтенсивності і не можуть у цьому зв'язку забезпечити високу ймовірність локалізації глобального екстремуму багатоекстримальної функції. Однак на практиці в багатьох випадках такі алгоритми виявляються досить ефективними, оскільки при не дуже великому числі локальних екстремумів фітнес-функції вірогідність потрапляння хоча б однієї особини в околицю глобального екстремуму цієї функції досить велика.

Шляхом зміни ставлення числа ітерацій, що відводиться на розв'язання кожної з локальних задач оптимізації, до максимально допустимого числа поколінь генетичного алгоритму можна регулювати баланс між інтенсифікацією і диверсифікацією пошуку.

Зазначимо, що алгоритми, в яких тим чи іншим чином комбінують один з алгоритмів глобальної оптимізації з деяким алгоритмом локальної оптимізації, відносять до класу меметичних (memetic) алгоритмів. Ці алгоритми іноді називають ламарковскими (Lamarckian) або алгоритмами з

ефектом Болдуіна (Baldwin effect). Останній термін пов'язаний з одним з найбільш правдоподібних варіантів ламаркианизма.

Гібридизацію пошукових алгоритмів можна реалізувати також шляхом їх перемикання за деякими правилами в процесі пошуку. Приклад такого підходу - модель навчальної еволюції (learnable evolution), в якій здійснюється перемикання між еволюційним алгоритмом і методом машинного навчання.

Багатопопуляційні генетичні алгоритми зазвичай використовують для організації паралельних обчислень, однак ефективним може бути використання таких алгоритмів також на традиційних однопроцесорних ЕОМ. Відомими варіантами багатопопуляційного генетичного алгоритму є острівний (island) генетичний алгоритм і клітинний (cellular) генетичний алгоритм.

Адаптивні генетичні алгоритми. Як було показано вище, багато генетичних операторів мають вільні параметри. Ці параметри можуть змінювати свої значення в процесі обчислень, тобто бути нестаціонарними.

Зробити параметр b нестаціонарним можна, замінивши його статичне значення деякої детермінованою або стохастичною функцією $b(t)$, де t – лічильник поколінь генетичного алгоритму. Можливий також альтернативний спосіб, заснований на використанні зворотного зв'язку значень цього параметра з ефективністю оператора. По відношенню до оператора мутації відомим прикладом такого підходу є «правило 1/5» (onefifth rule), згідно з яким частка успішних мутацій повинна бути рівною 1/5. Якщо ця частка менше 1/5, то значення відповідних вільних параметрів мутатора змінюють таким чином, щоб ймовірність мутації збільшилася, а якщо більше 1/5 - зменшилася. Тут під успішною розуміють мутацію, яка виробляє особина-нащадка, більш пристосовану порівняно зі своєю особиною-батьком. Такого роду параметри називають адаптивними. Генетичний алгоритм, що використовує адаптивні параметри своїх генетичних операторів, називають адаптивним.

Генетичний мікроалгоритм являє собою модифікацію канонічного генетичного алгоритму, призначену для вирішення завдань, які не потребують великих популяцій і довгих хромосом. Такий алгоритм доцільно використовувати у разі жорсткого ліміту часу, коли рішення, нехай і квазиоптимальне, необхідно знайти швидко. Можна сказати, що при синтезі генетичного мікроалгоритму за критеріями забезпечуваного якості рішення (його близькості до глобального екстремуму) і необхідним обчислювальних витрат перевага віддана другому критерію. Схема одного з варіантів генетичного мікроалгоритма має наступний вигляд.

1) Формуємо популяцію $S = (s_i, i \in [1{:}\,|S|])$,де число особин $|S| = 5$.

2) Обчислюємо пристосованість $\varphi(s_i)$ кожної з особин, знаходимо кращу особина, позначаємо її s_5' і переносимо в наступне покоління (елітарний відбір).

3) Методом турнірної селекції з числа особин $s_i, i \in [1; 5]$ вибираємо для репродукції чотири батьківські особини.

4) Виконуємо схрещування відібраних особин з імовірністю, рівної одиниці, так щоб отримати чотири особини-нащадка. Мутацію особин не виробляємо.

5) За допомогою того чи іншого критерію перевіряємо збіжність алгоритму. У випадку збіжності переходимо до кроку 1, в іншому випадку – до кроку 2.

Особливостями генетичного мікроалгоритма є: невеликий і фіксований розмір популяції; елітарний відбір особин; детермінована селекція; схрещування з імовірністю, рівній одиниці; невикористання мутації; багаторазове повторення процедури рестарту алгоритму.

2.3 Еволюційні стратегії

В 1964 році Рехенбергом (I. Rechenberg) був запропонований алгоритм еволюційної стратегії для вирішення задачі глобальної умовної мінімізації цільової та фітнес-функцій з обмеженнями типу нерівностей [12], [13], [15], [21].

Фітнес-функція у популяційних алгоритмах дозволяє оцінювати «якість» агентів популяції. Цю функцію також називають функцією приналежності, функцією корисності, функцією пристосованості тощо. У процесі пошуку оптимального рішення, агенти рухаються таким чином, щоб наблизитися до глобального екстремуму фітнес-функції [15].

Розглянемо цю задачу глобальної умовної оптимізації:

$$\max_{X \in D \subset R^x} F(X) = f(X^*) = f^*. \tag{2.2}$$

Область допустимих значень D визначаємо обмеженням типу: $D = \{X | G(X) \geq 0\}$.

Вважаємо, що більшим значенням цільової функції *f(X)* відповідає більше значення фітнес-функції $\varphi(X)$.

Суть алгоритму еволюційної стратегії передає наступна послідовність його основних кроків.

1) Здійснюємо мутації і схрещування особин поточної популяції.

2) Батьківських особин і отриманих таким чином особин-нащадків включаємо в проміжну популяцію.

3) На основі цієї популяції формуємо наступну популяцію шляхом детермінованого відбору без повторень найбільш пристосованих особин.

Перелічимо основні відмінності алгоритму еволюційної стратегії від класичного генетичного алгоритму. Перш за все, якщо генетичний алгоритм моделює еволюцію на рівні геномів особин, то алгоритм

еволюційної стратегії орієнтований на еволюцію їх фенотипів. Тому даний алгоритм використовує тільки речове кодування особин, на відміну від класичного генетичного алгоритму, який передбачає бінарне кодування.

У класичному генетичному алгоритмі процес відбору особин з батьківської популяції для включення їх в нову популяцію здійснюють випадковим чином пропорційно їх пристосованості. В алгоритмі еволюційної стратегії відбір особин в нову популяцію виробляють без повторень на основі детермінованої процедури.

Алгоритм еволюційної стратегії передбачає, що на кожній ітерації спочатку проводиться модифікація особин (мутація), а потім селекція. Класичний генетичний алгоритм виконує ці дії в зворотному порядку. Крім того, цей алгоритм використовує стаціонарні значення вільних параметрів своїх операторів (наприклад, ймовірності схрещування і мутації особин), а алгоритм еволюційної стратегії, навпаки, безперервно змінює значення цих параметрів в режимі самоадаптації.

Відзначимо також відмінності в способах заліку обмежень на компоненти вектори змінних параметрів. В алгоритмі еволюційної стратегії особин-нащадків, які не задовольняють хоча б одному з обмежень, що визначає область допустимих значень *D*, відкидають, тобто не включають в нову популяцію. Якщо число таких відкинутих особин перевищує деяку заздалегідь задану величину, то запускається процес адаптації параметрів алгоритму (наприклад, збільшується ймовірність мутації), В класичному генетичному алгоритмі значення вільних параметрів генетичних операторів є стаціонарними, і зазначені обмеження враховують за допомогою штрафних функцій.

Сучасні генетичні алгоритми значно відрізняються від класичних, так що, як зазначалося вище, відмінності між ними і алгоритмом еволюційної стратегії незначні.

Основою канонічного алгоритму еволюційної стратегії є комбінування генетичних операторів мутації і селекції за таким правилом.

1) Вибираємо з поточної популяції S особину s_i і шляхом застосування до неї оператора мутації отримуємо особину s_i'.

2) Обчислюємо пристосованості $\varphi(s_i)$, $\varphi(s_i')$ зазначених особин і поміщаємо в наступну популяцію *S(t+1)* найбільш пристосовану з них.

Тут індекс i послідовно приймає значення i = *1,2,..,,*$|S|$, Алгоритм селекції, який використовується на останньому кроці, являє собою так званий алгоритм селекції урізанням (truncation).

Для генерації випадкової початкової популяції *S(0)* необхідно, щоб для кожної компоненти x_j, $j \in [1{:}|X|]$ вектора змінних параметрів X було встановлено інтервал $[x_j^{min}; x_j^{max}]$ (цей інтервал використовується тільки на етапі ініціалізації). Для особини s_i, $i \in [1{:}\ |X|]$ початкові значення компонент $x_{i,j}$, $j \in [1{:}\ |X|]$ вектору X_i вважають рівномірно розподіленими в зазначеному інтервалі.

Подібно генетичному алгоритму, алгоритм еволюційної стратегії в якості умови зупинки може використовувати такі події:

– досягнення заданого числа поколінь $\hat{t}$;

– досягнення заданого стану популяції, коли, наприклад, приспособленності всіх особин популяції опустилися нижче деякого порога (нагадаємо, що мова йде про завдання мінімізації фітнес-функції);

– досягнення заданого рівня збіжності, коли в деякій метриці відстані між особинами популяції менше певного порогового значення.

Позначимо $\alpha = |S|$ число особин поточної популяції (батьківських особин), а $\beta = |S'| > \alpha$ – число особин-нащадків. Відомі так звані $(\alpha + \beta)$ і $\alpha\beta$ – алгоритми еволюційної стратегії.

$(\alpha + \beta)$ – алгоритм передбачає, що в проміжну популяцію S' входять всі $(\alpha + \beta)$ особини і з їх числа виробляють селекцію і включення в популяцію $S(t{+}1)$ кращих особин числом α. З визначення $(\alpha + \beta)$ алгоритму виходить, що він забезпечує відмінну від нуля ймовірність попадання в популяцію $S(t{+}1)$ деяких батьківських особин. Зауважимо, що

у введених позначеннях канонічний генетичний алгоритм, очевидно, відноситься до класу *(1 + 1)* -алгоритмів еволюційної стратегії.

У разі αβ -алгоритми в проміжну популяцію S' включають тільки β особин-нащадків і з їх числа проводять відбір і включення в популяцію S' кращих особин числом α . Очевидно, що αβ -алгоритм виключає можливість попадання в популяцію $S(t+1)$ батьківських особин.

Вільним параметром $(\alpha + \beta)$ і αβ -алгоритмів є ставлення $r = \frac{\beta}{\square} \geq 7$.

Зі збільшенням значень цього параметра зростає ймовірність того, що кожна особина-батько зробить щонайменше одну особину-нащадка, більш пристосовану, ніж вона сама. При зростанні значень параметра r відмінності між $(\alpha + \beta)$ і αβ - алгоритмами стають менш суттєвими.

Для $(\alpha + \beta)$ - алгоритму доведена його збіжність за ймовірністю, а для αβ – алгоритми питання про збіжність залишається відкритим.

У порівнянні з αβ -алгоритми, $(\alpha + \beta)$ - алгоритми, взагалі кажучи, більш «економно» використовують знайдені високопристосовані особини, оскільки залишає їх для конкуренції з нащадками. Ця якість $(\alpha + \beta)$ - алгоритмів підвищує інтенсивність пошуку, але зменшує його широту і може привести до передчасної збіжності алгоритму до деякого локального мінімуму фітнес-функції.

Оператор мутації в канонічному алгоритмі еволюційної стратегії є єдиним еволюційним оператором. Оператор використовує алгоритм, близький до алгоритму гаусової мутації, відповідно до якого значення гена $x_j^{'}$ хромосоми $X^{'}$ визначають на основі значення гена x_j - хромосоми X за формулою

$$x_j^i = x_j + N_i(m, \sigma), j[1:|X|], \qquad (2.3)$$

де середнє квадратичне відхилення σ інтерпретують як ймовірність мутації. Зазвичай використовують нульове математичне сподівання $m = 0$, а значення величини σ змінюють в процесі функціонування алгоритму в режимі самоадаптації, тому оператор мутації виявляється самоадаптаційним (self adaptive). При цьому параметр σ може включатися в розширений вектор змінних параметрів завдання, який еволюціонує за загальними правилами зміни особин алгоритму еволюційної стратегії.

У найпростішому випадку зміна середнього квадратичного відхилення σ може бути підпорядковане правилу однієї п'ятої (див. п.2.2). Відповідно до цього правила через кожне фіксоване число ітерацій перевіряють число успішних мутацій (породили більш пристосованих особин). Якщо їх відносна кількість перевищує 1/5 (популяція зосереджена в околиці деякого оптимуму), то величину σ по деякому закону збільшують, а в іншому випадку зменшують.

2.4 Алгоритми пошуку з заборонами

Алгоритм пошуку з заборонами (табу) (Tabu Search algorithm, TS algorithm) відноситься до класу метаевристичних алгоритмів інтелектуального пошуку. Основоположником алгоритму TS є Гловер (F. Glover). Спочатку алгоритм TS знайшов своє застосування в задачах теорії графів і цілочисельного програмування. Для вирішення безперервних завдань оптимізації алгоритм почали використовувати в кінці 80-х років минулого століття. В даний час алгоритм широко застосовують також в задачах планування та кластерного аналізу.

Основним механізмом, який дозволяє алгоритму TS долати локальні екстремуми, є список табу T (t) = T^t (tabu list). Список будується на основі деякої частини передісторії пошуку, тобто на основі рішень

$$X^t, X^{t-1}, \dots, (t-k) \geq 0, \tag{2.4}$$

і забороняє частину околиць d^t поточного рішення X^t, так що нова точка X^{t+1} є рішенням допоміжного завдання локальної умовної оптимізації

$$f(X^{t+1}) = minf(x), X \in d^t \backslash T^t \tag{2.5}$$

Очевидно, що алгоритм TS трансформується в звичайний алгоритм локального пошуку в разі, коли список T^t порожній, тобто коли k = 0.

Елементи списку T^t називають табу-переміщеннями (tabu moves). Взагалі, список заборон може мати змінну довжину, коли $k = k\ (t) = k^t$ Величина k^t має зміст числа ітерацій, протягом яких дане рішення знаходиться в списку T^t, і називається табу-тривалістю (tabu tenure). Відома рекомендація, по якій $k^t = O(|X|)$

Загальна схема алгоритму TS має наступний вигляд:

1) тим чи іншим способом обираємо початкову точку X^0 і вважаємо $\tilde{X}^* = X^0, \tilde{\varphi}^* = \varphi\left(X^0\right), T^0 = Null, t = 0$;

2) якщо $d^t \backslash T^t$= Null, то переходимо до кроку 4. В іншому випадку, за формулою (2.5) знаходимо рішення X^{t+1}і вважаємо *t = t + 1*;

3) якщо $\varphi(X^t) < \tilde{\varphi}^*$, то виконуємо присвоювання $\tilde{X}^* = X^t$, $\tilde{\varphi}^* = \varphi\left(X^t\right)$;

4) якщо умови закінчення пошуку виконані, завершуємо обчислення. В іншому випадку оновлюємо список заборон T^tі повертаємося до кроку 2.

Особливістю умов закінчення пошуку в алгоритмі TS є те, що ці умови включають в себе поржню множину $d^t \backslash T^t$.

Занадто суворі табу можуть призвести до передчасної стагнації обчислювального процесу. Тому в алгоритмі TS епізодично дозволяється

порушувати заборони. З цією метою використовують механізм так званих перспективних рішень (aspirant decisions) або еквівалентний механізм функції аспірації (aspirant-function) *a(X)*, яка скасовує табу-статус переміщення *X*. Вважають, що рішення $X \in T^t$ досягло рівня аспірації і може бути оголошено можливим, якщо $\varphi(X) \leq a(X)$.

2.5 Алгоритм оптимізації колонією мурах

Мурашина оптимізація була запропонована Дориго (M Dorigo), Маньеццо CV: Maniezzo) і Колорні (A. Colorni) в 1992 р як алгоритм розв'язання задачі комівояжера (Traveling Salesman Problem, TSP). Це завдання передбачає вибір найкоротшого шляху в графі, так що є пряма аналогія із завданням, яке щодня вирішують мурахи, оптимізуючи свої шляхи до їжі. В даний час відомо велика кількість дискретних і безперервних алгоритмів мурашиної оптимізації, які мають спільну назву алгоритми оптимізації колонією мурах (Ant Colony Optimization, ACO) [22], [23], [24].

Першим варіантом алгоритму ACO, розробленим для вирішення завдання неперервної оптимізації, став алгоритм безперервної оптимізації колонією мурах CACO (Continuous Ant Colony Optimization,), В даний час відомо безліч модифікацій алгоритму CACO, наприклад: алгоритм безперервно взаємодіє мурашиної колонії (Continuous Interaction Ant Colony, CIAC); розширений алгоритм оптимізації колонією мурах для безперервних областей (extended Ant Colony Optimization to continuous domains, ACOR); алгоритм безперервної системи мурашиної колонії (Continuous Ant Colony System, CACS); алгоритм бінарної мурашиної системи (Binary Ant System, BAS); алгоритм прямого призначення (Direct Application of Ant Colony Optimization, DACO).

Основними особливостями алгоритму CACO є комбінування глобального і локального пошуку, використання для глобального пошуку

генетичного алгоритму, використання для локального пошуку тільки каналу стігмергії.

Розглядаємо задачу багатовимірної глобальної умовної максимізації в гіперпаралелепіпеді П. В кожній точці $X_i(\tau)$ в якій побував мураха, $i \in [1:|S|]$, він залишає феромонну точку $\phi_i(t,\tau)$, інтенсивність якої (кількість феромону) на даній ітерації t дорівнює $\theta_i(t,\tau)$, де $\tau \in [0:t]$ – номер ітерації, на якій залишена ця точка. Кількість феромону в феромонних точках з ростом числа ітерацій зменшується (феромон випаровується) відповідно з рекуррентной формулою

$$\theta_i(t,\tau) = \begin{cases} b_\theta \theta_i(t-1,\tau), \theta_i(t-1,\tau) \geq \theta_{min}, \\ 0, \text{інакше}, \end{cases} \quad (2.6)$$

де b_θ – коефіцієнт стійкості феромонних точок. Формула (2.6) означає, що феромонна точка зникає, якщо кількість феромону в ній стає меншим за величину θ_{min}.

Сукупність феромонних точок

$$\Phi_i(t) = \{\phi_i(t,\tau) | \tau \in [0:t], \theta_i(t,\tau) > \theta_{min}\}, i \in [1:|S|] \quad (2.7)$$

утворює феромонний слід мурашки S_i. Число феромонних точок в цьому сліді позначаємо $|\Phi_i(t)|$.

На поточній ітерації t активними є феромонні точки, що належать всім феромонним слідам $\Phi_i(t)$, $i \in [1:|S|]$. Сукупність цих точок позначимо $\Phi(t) = \{\Phi_i(t), i \in [1:|S|]\}$. Тоді $|\Phi(t)| = \sum_{i=1}^{|S|} |\Phi_i(t)|$ – загальна кількість активних феромонних точок на поточній ітерації t. Введемо ще позначення: $|S^g| = \{S_i^g, i \in [1:|S^g|]\}, S^l = \{S_i^l, i \in [(|S^g|+1):|S|]\}$

– безлічі мурах, призначених для глобального і локального пошуку відповідно; $|S^g| + |S^l| = |S|$.

У введених позначеннях схема алгоритму CACO має такий вигляд.

Етап ініціалізації. Створюємо $|S|$ регіонів пошуку. Ініціалізуємо в кожному з регіонів $X_i(0), i \in [1:|S|]$ феромонну точку. Приймаємо лічильник числа ітерацій як $t = 1$.

Етап оптимізації:

1) обчислюємо в кожному з регіонів пошуку значення фітнес-функції $\varphi(X_i) = \varphi_i$;

2) сортуємо регіони в порядку зростання величини φ_i і представляємо результат сортування у вигляді лінійного списку;

3) з ймовірністю мутації $\aleph_m$ виповнюємо генетичну операцію мутації для перших 0,9 $|S^g|$ (найменш пристосованих) мурах зазначеного списку;

4) аналогічно виконуємо генетичну операцію кросовера для наступних 0,1 $|S^g|$ мурах того ж списку;

5) ініціалізуємо в кожному з $|S^g|$ регіонів пошуку, отриманих в результаті попередніх операцій, феромонних точку;

6) на основі каналу стігмергії для кожного з $|S^l|$ мурах виконуємо локальний пошук кращого регіону:

– якщо регіон з кращим значенням фітнес-функції знайдений, то ініціализируем відповідну феромонних точку і переміщаємо одного з мурах безлічі S^l в цей регіон;

– в іншому випадку випадковим чином робимо спробу знайти новий кращий регіон.

7) за формулою (2.5) моделюємо випаровування феромону.

Етап завершення обчислень. Перевіряємо умову закінчення ітерацій. Якщо ця умова виконана, завершуємо обчислення, в іншому випадку повертаємося до етапу оптимізації.

Кожен регіон пошуку $X_i(0)$, що створюється на етапі ініціалізації, являє є випадковою точку в області $П$, координати якої визначаємо за правилом $X_{i,j}(0) = U_1\left(x_i^-; x_i^+\right), i \in [1:|S|], j \in [1:|X|]$. Регіону $X_i(0)$ ставимо у відповідність феромонних точку $\phi_i(0,0)$ з інтенсивністю $\theta_i(0,0) = \theta_{init}$, де θ_{init} - початкова кількість феромону. Для мутації використовуємо гаусів мутатор

$$x_{i,j}^{'} = x_{i,j} + N_1(0,\sigma), j \in [1:|X|],, \tag{2.7}$$

де середнє квадратичне відхилення σ називаємо кроком мутації (mutation step size). З ростом числа ітерацій зменшуємо (редукуємо) цю величину за законом

$$\sigma(t) = \sigma_{max}(1 - (1 - \left(U_1(0;1)\right)^{\left(1-\frac{t}{t}\right)^{b_m}}), \tag{2.8}$$

де σ_{max} - максимальна величина кроку мутації;

b_m – натуральний параметр нелінійності. Нелінійна редукція кроку мутації дозволяє на початкових ітераціях пошуку забезпечити широту пошуку, а на завершальних ітераціях – точність локалізації максимуму.

В якості оператора кросовера використовуємо арифметичний кросовер

$$x_{i,j}^{'} = ux_{i,j} + (1-u)x_{i,j}, i,j \in \left[1:\left|S^g\right|\right], k \in [1:|X|], i \neq j, \tag{2.9}$$

де $u = U_1(0;1)$.

Батьківські особини S_i, S_j для схрещування вибираємо з числа 0,1 $|S^g|$ мурах безлічі S^g з однаковою ймовірністю.

Локальні пошук кращого регіону виконуємо за наступною схемою:

1) випадковим чином обираємо мурашок S_i з числа останніх $|S^l|$ мурах зазначеного вище лінійного списку;

2) з урахуванням інтересу даного мурашки обчислюємо координати центру ваги поточних активних феромонних точок

$$W_i(t) = \sum_{j,k} \omega_{i,j,k}(t) X_{j,k}, j \in [1:|S|], k \in [1:|\Phi_j(t)|], \quad (2.10)$$

де $X_{j,k}$ – k-та точка феромонного сліду $\Phi_j(t)$;

$$\omega_{i,j,k}(t) = \frac{\omega_{i,j,k}(t)}{\sum_{l=1}^{|S|} \sum_{m=1}^{|\Phi_j(t)|} \omega_{i,l,m}(t)}, \quad (2.11)$$

Формула (2.11) – нормований інтерес i-го мурашки до k-ї точки феромонного сліду $\Phi_j(t)$;

3) обчислюємо координати нового можливого положення мурашки S_i; за формулою

$$X_i^{'} = X_i + V_i. \quad (2.12)$$

Тут V_i – крок в напрямку $X_i - W_i(t))$, величина якого дорівнює $abs(b_{beg} - b_{inc} a_i)$, де b_{beg}, b_{inc} – позитивні визначені користувачем константи, що мають сенс початкової величини кроку і його інкремента відповідно; a_i – слабкість або вік регіону X_i (в оригіналі – weakness і age відповідно).

Початкове значення величина a_i отримує при створенні відповідного регіону пошуку. Якщо значення фітнес-функції в регіоні X_i'вище значення тієї ж функції в регіоні X_i , то значення величини a_i збільшуємо, а в іншому випадку – зменшуємо. Даний механізм дозволяє зменшувати крок локального пошуку з наближенням мурашки s; до положення максимуму фітнес-функції. Інтерес мурашки S_i до k-ї точки феромонного сліду $\Phi_j(t)$ визначає формула

$$\omega_{i,j,k}(t) = \frac{\rho(t)}{2}\theta_{j,k}(t)\, exp\left(-\rho_{i,j,k}(t)\right), i,j \in [1:|S|], k \in [1:|\Phi_j(t)|],, \qquad (2.13)$$

в якій $\rho(t)$ – середнє поточне відстань між двома мурашками популяції;

$\theta_{j,k}(t)$ – кількість феромону у зазначеній феромонной точці;

$\rho_{i,j,k}(t) = |\left|X_i(t) - X_{j,k}\right||_E$ – евклідова відстань між регіоном $X_i(t)$ і тієї ж феромонної точки.

Припустимо, що на даній ітерації t активними є всі регіони, в яких побував кожен з мурах популяції. Тоді з формули (2.11) виходить, що для мурашки S_i, $i \in [1:|S|]$ координати центру ваги всіх активних феромонних точок можна обчислити за формулою

$$W_i(t) = \sum_{j,k}^{\square} \omega_{i,j,k}(t) X_{j,k}, j \in [1:|S|], k \in [0:t], \qquad (2.14)$$

Як умова закінчення ітерацій використовують типові для популяціонному алгоритмів оптимізації умови. Алгоритм CACO містить наступні основні вільні параметри:

– θ_{init} – початкова кількість феромону (рекомендоване значення θ_{init} = 1,0);

– θ_{min} – кількість феромону, нижче якого феромонна точка стає неактивна (рекомендоване значення θ_{min} = 0,00000001);

– $b_\theta \in (0;\infty)$ – коефіцієнт стійкості феромонних точок (рекомендоване значення b_θ=0,1);

– $\aleph_m, \sigma_m, \sigma_{max}, b_m$ – ймовірність мутації, крок мутації, максимальна величина кроку мутації і параметр нелінійності мутації відповідно;

– b_{beg}, b_{inc} – початкова величина кроку локального пошуку і його інкремент відповідно;

– $a_i(0), i \in [1:|S|]$ - початкове значення віку регіонів.

За загальним властивості популяційних алгоритмів пошукової оптимізації зі збільшенням розміру популяції |S| алгоритм CACO підвищує ймовірність локалізації глобального максимуму фітнес-функції (ціною підвищення обчислювальної складності кожної ітерації). Для функцій, що мають просту топологію, хороші результати може показати популяція з усього приблизно 10 мурах. Для складних багатовимірних багатоекстремальних функцій число мурах має бути близько 100. Автори алгоритму рекомендують використовувати число особин в популяції, рівне $|S| = |S|_{max}(1 - exp(-0{,}1|X|)) + 5$, де $|S|_{max}$ = 1000 - максимально допустиме число мурах в популяції.

У термінах алгоритму роя частинок зменшення кількості феромону θ_{min} і збільшення коефіцієнта стійкості феромонних точок b_θ підвищує вплив соціальної компоненти на еволюцію мурах. В цьому випадк феромонні сліди $\Phi_j(t), j \in [1:|S|]$ "подовжуються" (зростають величини $|\Phi_j(t)|$) і вплив інших мурах на кожного даного мурашки збільшується. В результаті підвищується ймовірність "застрявання" мурах в локальних максимумах цільової функції [22], [23], [24].

2.6 Висновки до розділу 2

В другому розділі розглянуто найбільш поширені алгоритми еволюційного моделювання. Визначено, що в математичну основу цих алгоритмів покладено використання базових принципів теорії біологічної еволюції, а саме відбору, мутації і відтворення особин.

Найбільш уживаними є генетичні алгоритми, в той час, як для вирішення різних задач можна застосовувати й інші.

Розглянуто основні певні риси, притаманні цьому класу алгоритмів, які включають такі кроки: ініціалізація алгоритму та задання значень вільних параметрів алгоритму, застосування пошукових (міграційних) операторів алгоритму до поточних станів агентів, в результаті чого вони набувають нові положення, здійснення перевірки виконання умов завершення ітерацій. Якщо ці умови не виконані, повернення до виконання алгоритму, в іншому випадку вважаємо, що рішення знайдено.

Проведено аналіз математичного опису наступних еволюційних алгоритмів: генетичні алгоритми, еволюційна стратегія, алгоритми інтелектуального пошуку (табу-пошук), алгоритми оптимізації колонією мурах. Відповідно аналізу літератури визначено, що алгоритм еволюційної стратегії, на відміну від генетичних алгоритмів, передбачає, що на кожній ітерації спочатку проводиться модифікація особин (мутація), а потім селекція. Класичний генетичний алгоритм виконує ці дії в зворотному порядку. Основою канонічного алгоритму еволюційної стратегії є комбінування генетичних операторів мутації і селекції.

Для вирішення задач планування та кластерного аналізу доречно застосувувати алгоритми інтелектуального пошуку, основним механізмом, який дозволяє алгоритму TS долати локальні екстремуми, є список табу.

В роботі розглянуто алгоритм для вирішення задач безперервної, оптимізаціï алгоритм оптимізації колонією мурах CACO. Основними особливостями алгоритму CACO є комбінування глобального і локального

пошуку, використання для глобального пошуку генетичного алгоритму, використання для локального пошуку тільки каналу стігмергії.

Алгоритми мурахи мають багато різновидів і широко застосовуються у вирішенні різноманітних задач.

Таким чином, еволюційні алгоритми зазвичай використовуються для загального опису алгоритмів пошуку, оптимізації або навчання, що базується на формалізованих принципах природнього еволюційного процесу

3 ПРОЕКТУВАННЯ ІСПР НА ОСНОВІ АЛГОРИТМУ МУРАШИНИХ КОЛОНІЙ

3.1 Проектування компонентів ІСПР

Застосування будь-яких математичних алгоритмів та методів вимагає розроблення спеціальних інформаційних систем, які б забезпечили просте і зручне їх використання для кінцевого користувача, який не завжди є фахівцем у цій сфері.

Як правило, еволюційні алгоритми застосовують в системах штучного інтелекту та інтелектуальних системах прийняття рішень і являються складовими бази моделей таких систем. ІСПР та СШІ – це такі системи, які в інтерактивному режимі забезпечують легкий доступ до даних і моделей з метою підтримки прийняття рішень в певній сфері, для якої і вони розробляються.

Загалом, за класичною структурою в склад ІСПР входять наступні компоненти [1]:

– моделі і система керування базою моделей (БМ);
– база даних і система керування базою даних;
– інтерфейс користувача.

Ця структура відрізняє ІСПР від інших видів програмних застосунків та інформаційних систем.

З розвитком інформаційних систем і технологій вищеописана структура може бути розширена різними компонентами. Наприклад, системою підтримки комунікацій, системою генерування звітів, базою знань тощо.

Коротко розглянемо основні положення проектування вищеописаних основних компонентів.

Інтерфейс користувача. При проектування користувацького інтерфейсу розробляють схеми та макети екрану, приділяючи увагу його

ергономіці, візуальній привабливості, простоті у використанні кінцевим користувачем.

База даних і СУБД. Загалом, під базою даних розуміють поіменовану, структуровану сукупність логічно взаємопов'язаних даних, які характеризують окрему предметну область і перебувають під управлінням СУБД [25]. Тобто в БД зберігаються структуровані дані. Сучасні технології передбачають також збереження неструктурованих даних в СУБД нового покоління, які мають назву NoSQL. Проте, в даній роботі розглядаються лише структуровані дані, які зберігаються в ІСПР. Як видно з визначення, БД неможлива без організації в певній СУБД. При проектуванні ІСПР якщо передбачено використання даних з БД, цей компонент є обов'язковим.

База моделей і система керування базою моделей. Невід'ємною складовою більшості ІСПР є математичні моделі й алгоритми. Реалізація використання моделей ІСПР може бути здійснена наступним чином. В ІСПР може бути організований доступ до моделей кінцевим користувачам шляхом реалізації компоненту системи керування базою моделей. З іншого боку, моделі можуть бути вбудовані в ІСПР і недоступні для кінцевого користувача.

Взаємодія основних складових ІСПР розглядається при проектуванні її архітектури. Тип архітектури залежить від розміру, призначення і виду ІСПР. Наприклад, невелика ІСПР, розроблена для потреб невеликого підприємства, не потребує складних архітектурних рішень. В той час, як широкомасштабні СППР, розраховані на вирішення складних завдань чи використання на великих підприємствах, вимагають ретельного проектування архітектури з детальним описом її компонентів.

Отже, архітектура ІСПР, що проектується, буде включати наступні компоненти:

– база даних;
– система керування базою даних;

– база моделей;
– інтерфейс користувача.

3.2 Розробка моделі пошуку оптимального маршруту доставки вантажів на основі алгоритму мурашиних колоній в умовах часових обмежень

Відповідно до класичної структури будь-якої СППР, важливим її компонентом є база моделей, яка включає в себе моделі, на основі яких приймаються рішення. Розглянемо застосування алгоритмів еволюційного моделювання в ІСПР на прикладі алгоритму мурашиних колоній.

Основні положення алгоритму мурашиних колоній розглядалися в розділі 2.5. Як правило, дані алгоритми використовують для вирішення наступних задач [22][23] [24]:

– задача комівояжера;
– пошук оптимального маршруту доставки вантажів;
– задачі календарного планування;
– квадратична задача про призначення;
– формування розкладів;
– сегментація медичних зображень [26].

Розглянемо використання цього алгоритму для вирішення задачі пошуку оптимального маршруту доставки вантажів на основі задачі комівояжера.

Задача комівояжера є однією з основних проблем комбінаторної оптимізації на графових моделях, на пошук ефективного вирішення якої витрачається багато зусиль [27]. Цільовою функцією (ЦФ) даної задачі є мінімальна довжина Гамільтона циклу, що проходить через безліч вершин графа. В МА пошук оптимуму завдання комівояжера здійснюється шляхом використання агентів (мурах), які залишають феромон на пройденому

маршруті, а саме, чим коротше шлях, тим більше концентрація феромону і, відповідно, вище ймовірність використання цього шляху наступними агентами. Оптимальним маршрутом буде той, де концентрація феромону найвища. МА вважається ефективним методом рішення задачі комівояжера. Проте, даний метод має один недолік - існує ймовірність досягнення локального оптимуму рішення [27].

Розглянемо використання алгоритму в наступних умовах.

Першочерговою задачею є обслуговування усіх клієнтів мінімальною кількістю маршрутів, а для знайдених маршрутів необхідно мінімізувати загальну тривалість маршруту.

Відповідно поставленій задачі вхідні дані наступні:

– існує одне центральне депо, з якого виїжджають автомобілі;

– однорідні транспортні засоби;

– наявна інформація про клієнтів: їх потреби в товарах і часі обслуговування.

Умови задачі наступні:

– загальна кількість товару, який поставляється, не може перевищувати вантажомісткість транспортного засобу;

– для кожного клієнта задається часове вікно;

– у випадку жорсткого часового вікна, транспортний засіб не може обслужити клієнта, якщо настала верхня межа часового вікна;

– в різні проміжки часу вантаж може бути доставлений в точку доставки повторно.

Метою розв’язання задачі є пошук множини маршрутів, які починаються і закінчуються в депо, задовольняють потреби клієнтів, несуть мінімальні витрати.

Особливістю даної задачі є потреба в різних проміжках часу повертатися повторно в пункт доставки, я кому транспортний засіб вже був.

Задача, яка розглядає оптимізацію маршруту з умовою часового інтервалу відноситься до класу задач маршрутизації транспорту (Vehicle Routing Problems, VRP), а саме задача маршрутизації з часовим вікном (VRP with Time Windows, VRPTW) [28].

Відповідно положень щодо задачі VRPTW, вона подібна VRP з основним додатковою умовою: для виконання запиту кожного клієнта v_i існує відомий проміжок часу, визначений як інтервал $[e_i, l_i]$ - намічений горизонт (scheduling horizon).

Метою задачі є мінімізація кількості транспортних засобів, загальні часи шляху і очікування, необхідні для обробки запитів клієнтів в призначені інтервали часу.

У порівнянні з VRP, в задачах даного типу додаються такі обмеження:

– рішення є неприйнятним, якщо клієнт обслуговується після верхньої часової межі;

– машина, яка прибула раніше нижньою часової межі, очікує її настання;

– як варіант, запізнення не впливає на придатність рішення, але додає деякий штрафне значення до цільової функції.

Отримавши рішення VRPTW, крім усього іншого, є можливість точніше підібрати час виїзду транспорту з депо і тим самим уникнути непотрібних простоїв.

На основі раніше розроблених моделей [29] побудуємо модифіковану модель пошуку оптимального маршруту доставки вантажів на основі алгоритму мурашиних колоній в умовах часових обмежень. Дана модель була розроблена на основі моделі, описаної в [29]. Відповідно положень, описаних в [29], маємо наступний комплекс моделей (3.1 – 3.10)

Мінімізація сумарної тривалості переїздів подана у вигляді цільової функції:

$$\text{Min} \sum_{i=0}^{N} \sum_{j=0, j\neq i}^{N} \sum_{k=1}^{K} C_{ij} X_{ijk} . \quad (3.1)$$

За умови обмежень:

(3.2) – максимальна кількість маршрутів;

(3.3) – обмеження на переїзд;

(3.4) – (3.5) – обмеження на обслуговування;

(3.6) – обмеження на місткість;

(3.7) – обмеження на час переїзду;

(3.8) – (3.10) – обмеження по тимчасовим вікнам:

$$\sum_{k=1}^{K} \sum_{j=1}^{N} X_{ijk} \leq K, \text{ для } i = 0, \quad (3.2)$$

$$\sum_{j=1}^{N} X_{ijk} = \sum_{j=1}^{N} X_{jik} \leq 1, \text{для i = 0 k} \in \{1, \dots, K\}, \quad (3.3)$$

$$\sum_{k=1}^{K} \sum_{j=0, j\neq i}^{N} X_{ijk} = 1, \text{ для i} \in \{1, \dots, N\}, \quad (3.4)$$

$$\sum_{k=1}^{K} \sum_{i=0, i\neq j}^{N} X_{ijk} = 1, \text{ для j} \in \{1, \dots, N\}, \quad (3.5)$$

$$\sum_{i=1}^{N} q_i \sum_{j=0, j\neq i}^{N} X_{ijk} \leq Q, \text{ для k} \in \{1, \dots, K\}, \quad (3.6)$$

$$\sum_{i=1}^{N}\sum_{j=0,j\neq i}^{N} X_{ijk}\left(t_{ij}+s_i+w_i\right)\leq T_k, \text{ для k} \in \{1,\dots,K\}, \quad (3.7)$$

$$t_0 = w_0 = s_0 = 0, \quad (3.8)$$

$$\sum_{k=1}^{K}\sum_{i=0,i\neq j}^{N} X_{ijk} = X_{ijk}\left(t_i+t_{ij}+s_i+w_i\right)\leq t_j \text{ для j} \in \{1,\dots,N\}, \quad (3.9)$$

$$e_i \leq (t_i+w_i) \leq l_i, \text{для} i \in \{1,\dots,N\}, \quad (3.10)$$

де $X_{ijk} = \begin{cases} 1-\text{звязокміжзамовникоміdo}j \\ \text{відвіднимтранспортнимзасобом}k; \\ 0-\text{інше} \end{cases}$

t_i – час прибуття в пункт i;

t_{ij} – тривалість переїзду із i в j;

w_i – час очікування в пункті i;

q_i – потреби в товарі пункту i;

e_i – початок тимчасового вінка в пункті i;

e_i – початок тимчасового вінка в пункті i;

l_i – кінець тимчасового вікна в пункті i;

s_i – тривалість обслуговування i;

K – кількість транспортних засобів;

T_k – максимальна триваілсть (час) маршруту для k;

Q – місткість ТС;

N – множина пунктів, депо включно.

Оскільки особливістю нашої задачі є вимога повернення транспортного засобу в той же пункт доставки в інший час, вищеописану задачу будемо розглядати як ітераційну в певних проміжках часу.

Розіб'ємо інтервал часу D на множину рівних часових періодів і позначимо як d номер кожного періоду інтервалі часу, де $d \in \{1, \dots, D\}$.

Тоді модель (3.1) – (3.10) набуває наступного вигляду (3.11) – (3.20).

$$\text{Min} \sum_{d=1}^{D} \sum_{i=0}^{N} \sum_{j=0, j\neq i}^{N} \sum_{k=1}^{K} C_{ij} X_{ijkd}. \quad (3.11)$$

За умови обмежень:

$$\sum_{k=1}^{K} \sum_{j=1}^{N} X_{ijkd} \leq K, \text{ для } i = 0, \quad (3.12)$$

$$\sum_{j=1}^{N} X_{ijkd} = \sum_{j=1}^{N} X_{jikd} \leq 1, \text{для i = 0 k} \in \{1, \dots, K\}, \quad (3.13)$$

$$\sum_{k=1}^{K} \sum_{j=0, j\neq i}^{N} X_{ijkd} = 1, \text{ для i} \in \{1, \dots, N\}, \quad (3.14)$$

$$\sum_{k=1}^{K} \sum_{i=0, i\neq j}^{N} X_{ijkd} = 1, \text{ для j} \in \{1, \dots, N\}, \quad (3.15)$$

$$\sum_{i=1}^{N} q_i \sum_{j=0, j\neq i}^{N} X_{ijkd} \leq Q, \text{ для k} \in \{1, \dots, K\}, \quad (3.16)$$

$$\sum_{i=1}^{N} \sum_{j=0, j\neq i}^{N} X_{ijkd} \left(t_{ij} + s_i + w_i\right) \leq T_k, \text{ для k} \in \{1, \dots, K\}, \quad (3.17)$$

$$t_0 = w_0 = s_0 = 0, \quad (3.18)$$

$$\sum_{k=1}^{K}\sum_{i=0,i\neq j}^{N} X_{ijkd} = X_{ijkd}\left(t_i + t_{ij} + s_i + w_i\right) \leq t_j \text{ для } j \in \{1, \dots, N\}, \quad (3.19)$$

$$e_{id} \leq (t_i + w_i) \leq l_{id}, \text{для} i \in \{1, \dots, N\}, \quad (3.20)$$

Реалізація заданої моделі (3.11) – (3.20) на основі алгоритму мурашиних колоній здійснюється наступним чином.

Крок 1. Ініціалізація початкового інтервалу d.

Крок 2. Формування списку точок доставки вантажу у заданий інтервал часу.

Крок 3. Ініціалізація ACO. Установка параметрів ACO і ініціалізація феромонів мережі.

Крок 4. Формування маршруту. Кожна мураха починає будувати маршрут з депо відповідно до правил переходу.

Крок 5. Локальний пошук. Після того, як кожен мураха побудував рішення, використовується Пошук в околиці для вивчення околиць рішень. 2-opt обмін використовується для поліпшення сусідніх рішень. Якщо сусіднє рішення краще, ніж поточне рішення, то сусіднє рішення стане поточним.

Крок 6. Оновлення феромона. Приріст феромону може бути обчислений в залежності від якості рішення і рівень феромону на ребрах може бути оновлений.

Крок 7. Статус табу. За певний проміжок часу Табу пошук дозволено використовувати один раз. Коли ACO отримує 4 послідовних однакових рішення, то розглядаються рішення поблизу оптимуму або локальних оптимумів. Якщо Табу пошук було змарновано, то перейти на Крок 8, інакше – Крок 10.

Крок 8. Табу пошук. Поточне краще рішення ACO буде перетворено Табу пошуком. Потім відбувається пошук сусідніх рішень, перехід на краще сусіднього рішенням і оновлення феромону в ACO.

Крок 9. Завершення Табу пошуку. Перевіряється умови завершення Табу пошуку. Якщо воно задовольняє умовам припинення, то перейти на Крок 4 і продовжити ACO, інакше - Крок 8 і продовжити Табу пошук.

Крок 10. Завершення ACO. Перевіряється умови завершення ACO. Якщо воно задовольняє умовам припинення, ACO завершується, інакше - Крок 8 і продовжити ACO пошук.

Крок 11. Зміна інтервалу часу. Якщо верхній інтервал досягнуто часу завершення маршруту, закінчення алгоритму і перехідна крок 14 Інакше – перехід на крок 12.

Крок 12. Ініціалізація початкової точки наступної ітерації. Приймаємо за депо останню знайдену точку. Формуємо новий список точок доставок в *f*-ти проміжок часу.

Крок 13. Повернення до кроку 3.

Крок 14. Завершення алгоритму.

Запропонований алгоритм реалізації розробленої моделі дозволить обчислити модель на основі алгоритму мурашиних колоній.

3.3 Опис бізнес-процесів транспортної компанії

З метою розуміння діяльності транспортної компанії, пов’язаної з доставкою вантажів клієнтам, розглянемо основні бізнес-процеси для вирішення даної задачі.

Моделювання бізнес-процесів було здійснено в середовищі проектування Enterprise Architect v14.

Розглянемо наступні бізнес-процеси:

– бізнес-процес «Оформлення замовлення на доставку» (рисунок 3.1);

– бізнес-процес «Побудова оптимального маршруту доставки» (рисунок 3.2).

Бізнес-процес "Оформлення замовлення на доставку" виконується наступним чином. Працівник call-центру отримує змовлення від клієнта та необхідні дані про нього, а саме: адресу доставки, вміст вантажу (замовлені товари), допустимий проміжок часу доставки. Отрмана інформація вноситься в базу даних замовлень, уточнюється у клієнта правильність внесених даних, таким чином здійснюється підтвердження замовлення. Далі замовлення передається у відділ доставки.

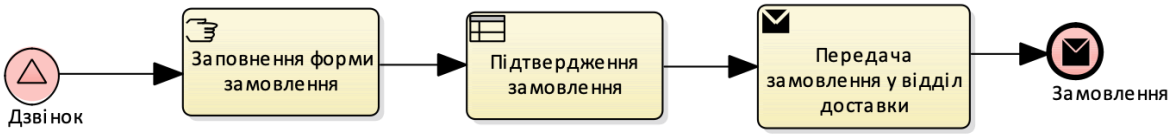

Рисунок 3.1 – Опис бізнес-процесу «Оформлення замовлення на доставку»

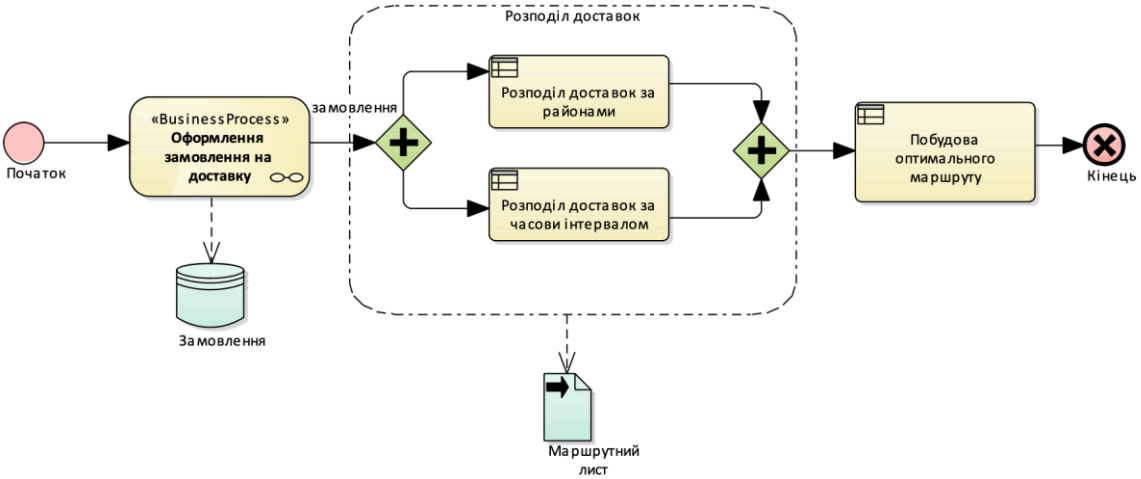

Рисунок 3.2 – Опис бізнес-процесу «Побудова оптимального маршруту доставки»

Даний бізнес-процес виконується після бізнес-процесу оформлення замовлення. Вхідними даними для виконання цього процесу являються оформлені замовлення. Система отримує з БД інформацію про нові замовлення та розподіляє їх за районами, закріплює за транспортним засобом. Далі здійснюється розрахунок оптимальної моделі маршруту на основі моделей, описаних в (3.11) – (3.20).

3.4 Моделювання ІСПР

Проектування будь-якої системи передбачає послідовність певних кроків з метою виявлення висог до системи та проектування її компонетів.

З метою моделювання ІСПР в роботі визначені функціональні вимоги до ІСПР, розроблена діаграма використання (Use Case), алгоритм роботи системи, представлений у вигляді діаграм діяльності (Activity diagram), побудована діаграма класів (Class diagram).

Моделювання ІСПР здійснювалось в середовищі проектування Enterprise Architect v14.

На рисунку 3.3 показаний перелік функціональних вимог до ІСПР. Був визначений наступний перелік функціональних вимог:

– система повинна здійснювати облік адрес доставок;

– система повинна розраховувати відстані між пунктами доставки;

– система повинна будувати оптимальні маршрути руху доставки, а саме:

  – створення графів заданої розмірності,

  – побудова матриці відстаней,

  – розрахунок довжини маршруту,

  – розрахунок часу виконання маршруту,

  – побудова графу.

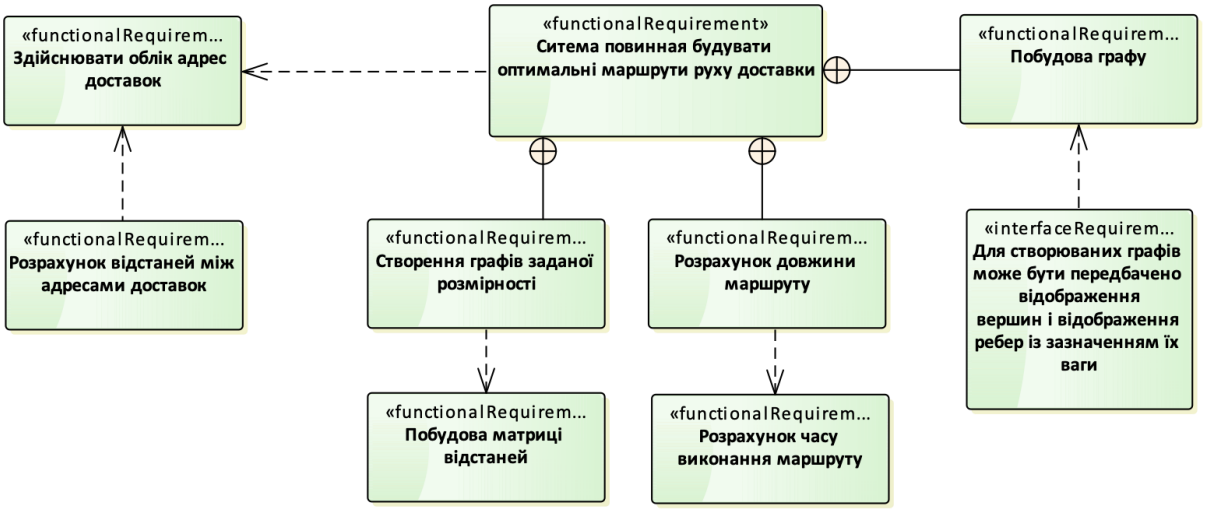

Рисунок 3.3 – Функціональні вимоги до ІСПР

На рисунку 3.4 показана діаграма діяльності у вигляді діаграми (Use Case).

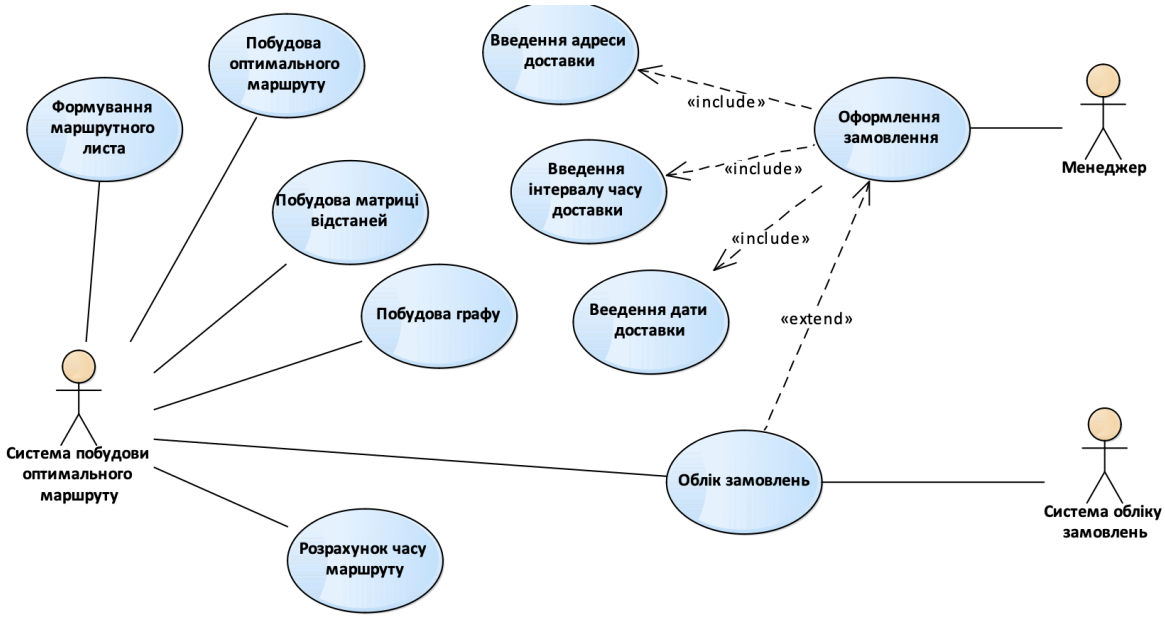

Рисунок 3.4 – Діаграма Use Case для ІСПР

На рисунку 3.5 показаний алгоритм роботи ІСПР, який передбачає наступні кроки:

– отримання і аналіз замовлення;
– розподіл доставок за районами;
– розподіл доставок за датами;
– формування маршрутного листа;
– розподіл доставок за часовими інтервалами;
– побудова оптимального маршруту.

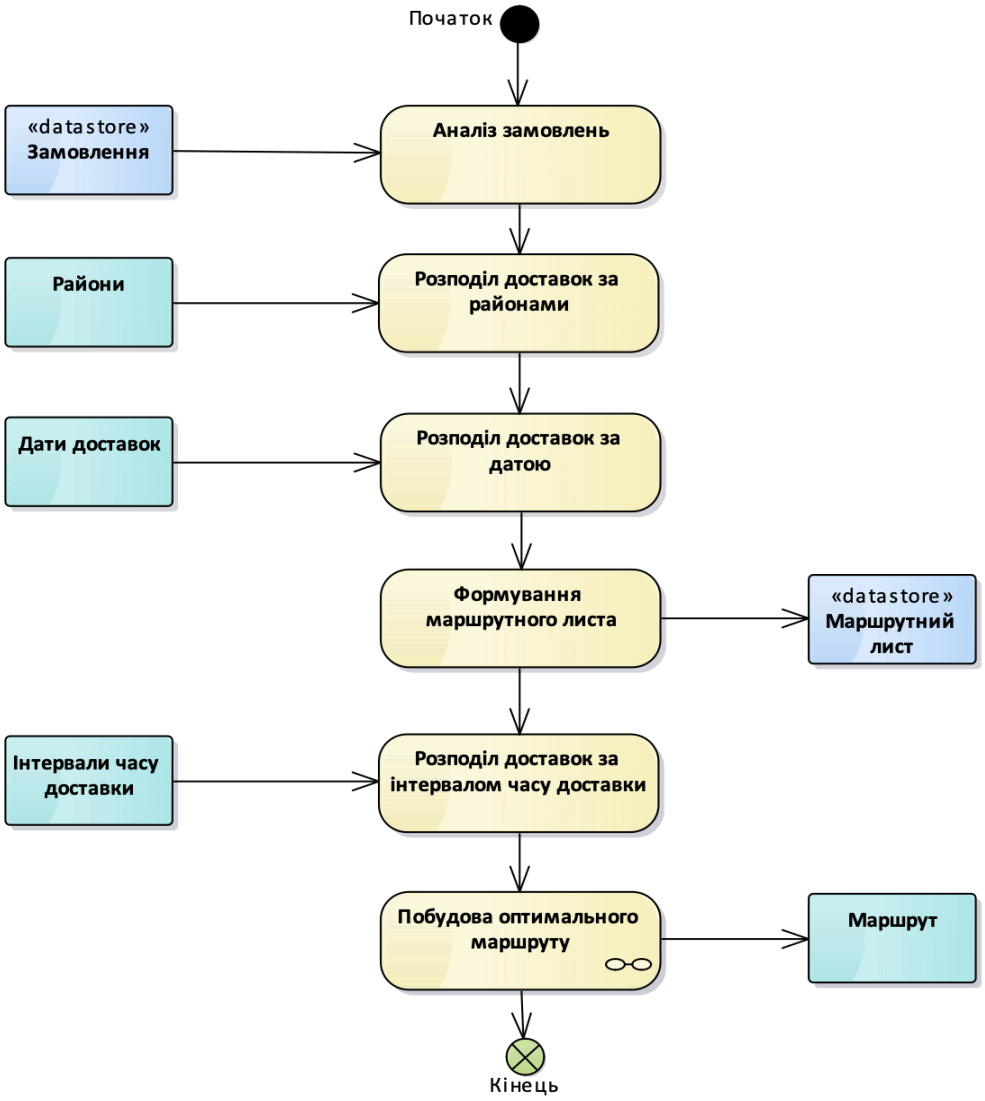

Рисунок 3.5 – алгоритм роботи ІСПР

На рисунку 3.6 – деталізація процесу пошук оптимального маршруту.

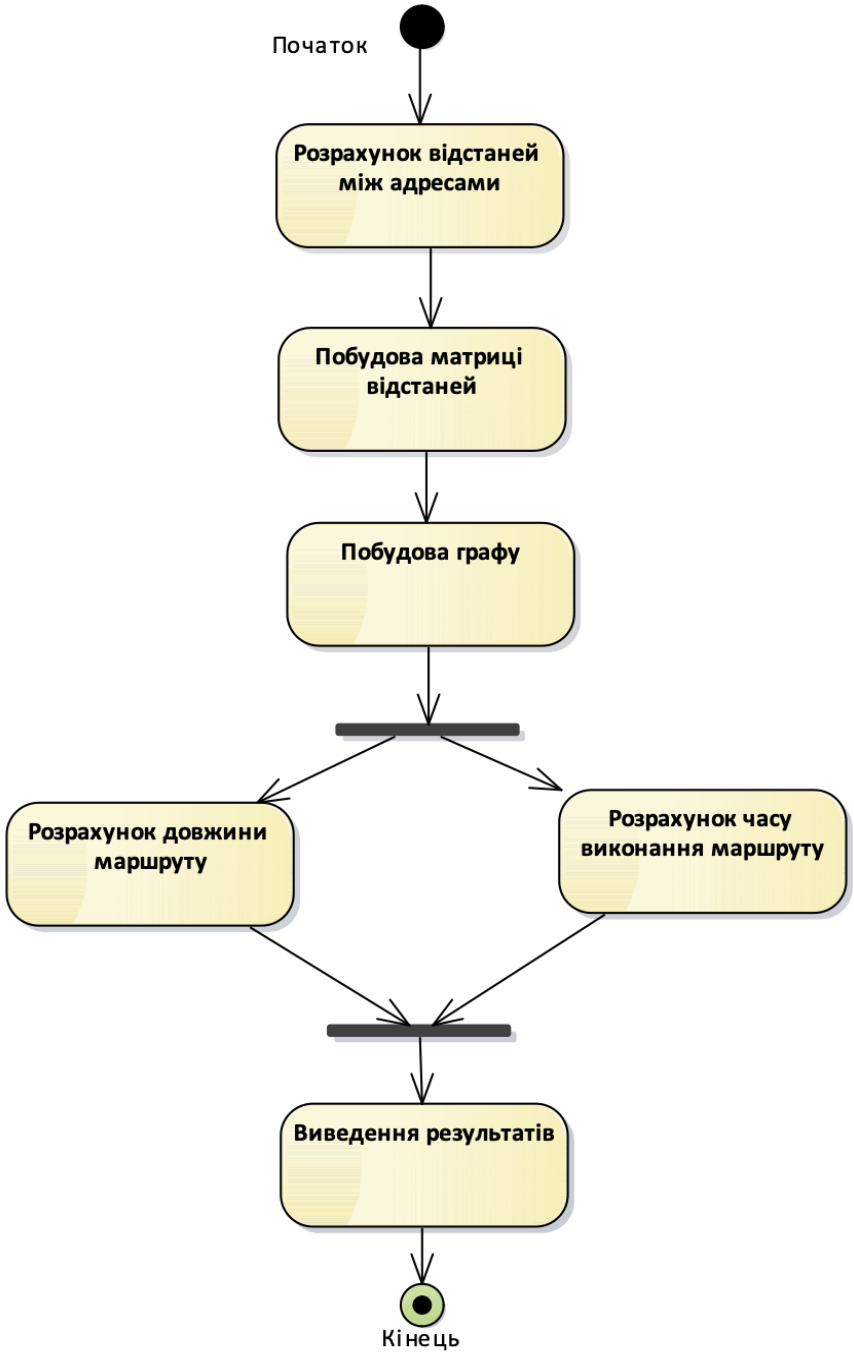

Рисунок 3.6 – Даграма діяльності "Пошук оптимального маршруту"

Діаграма класів представлена на рисунку 3.7. Більш детальний опис даної діаграми представлений в наступному розділі при описі реалізації програми.

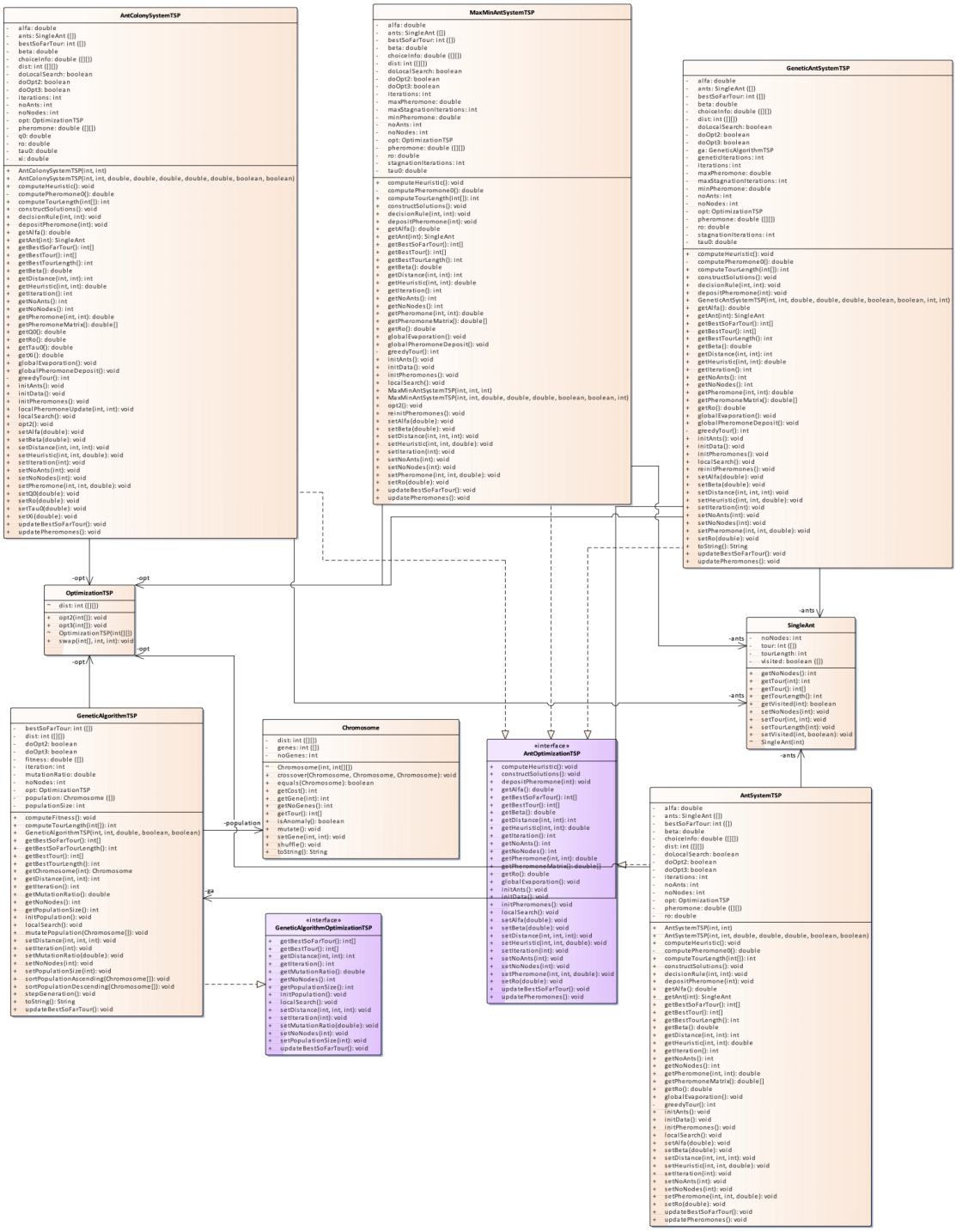

Рисунок 3.7 – Діаграма класів ІСПР

Висновки до розділу 3.

В третьому розділі отримані наступні результати.

При проектування ІСПР важливо визначити її структуру. В даній роботі запропоновано розробити ІСПР, яка містить наступні компоненти: база даних, система керування базою даних, база моделей, інтерфейс користувача.

Вдосконалено модель пошуку оптимального маршрутку доставки вантажу, передбачивши в моделі можливість повернення в ту ж точку призначення в різний часовий інтервал. В запропонована модель представлена як динамічна, яка розглядає оптимізацію маршруту за певні проміжки часу.

В роботі описані моделі бізнес-процесів підприємства, а саме бізнес-процеси «Оформлення замовлення на доставку» та «Побудова оптимального маршруту доставки». Моделі даних бізнес-процесів виконані відповідно нотацій BPMN.

Визначені функціональні вимоги до системи, а саме: система повинна вести облік адрес доставок, розраховувати відстані між пунктами доставки, будувати оптимальні маршрути руху доставки.

В середовищі Enterprise Architect v14 побудовані моделі діяльності системи, а саме: Use Case, діаграми діяльності та діаграма класів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ НА ОСНОВІ АЛГОРИТМУ МУРАШИНИХ КОЛОНІЙ

4.1. Розробка комп’ютерної програми «Мурашиний алгоритм»

Основними етапами розробки програми для розрахунку оптимального маршруту відстаней між досліджуваними містами згідно моделі «Мурашиного алгоритму» за Марком Дорігом є:

1. проектування структури класів та інтерфейсів;
2. перенесення математичного алгоритму на мову програмування;
3. розробка графічного інтерфейсу.

Було запропоновано використання мови програмування Java, бо вона є об’єктно-орієнтованою; може бути використана для створення основних класів програми; дозволяє запускати виконувані файли на будь-який ОС і виводити на екран складні інтерфейси.

Основними технологіями є:

1. JDK – платформа розробника Java.
2. JavaFX – графічна бібліотека.

Для розробки програми використовується середовище IDE – IntellijIdea Ultimate, яка має ряд переваг над існуючими IDE.

4.2 Проектування структури класів та інтерфейсів.

Серед найбільш важливих java-класів та java-інтерфейсів можна виділити такі:

– клас SingleAnt – який описує поведінку окремої мурахи;

– інтерфейс AntOptimization – який абстрактно описує алгоритм для вирішення завдання комівояжера за допомогою Мурашиного алгоритму;

– клас AntColony – який описує формування мурашиної колонії для вирішення задач оптимізації;

– клас GraphicalSolverFrame – який реалізує графічний інтерфейс програмного рішення;

– клас PointCoords – який реалізує відображення руху мурах за побудованим маршрутом;

– клас Main – клас запуску програми.

Взаємозв’язок між перерахованими класами програми представлений на наступному рисунку.

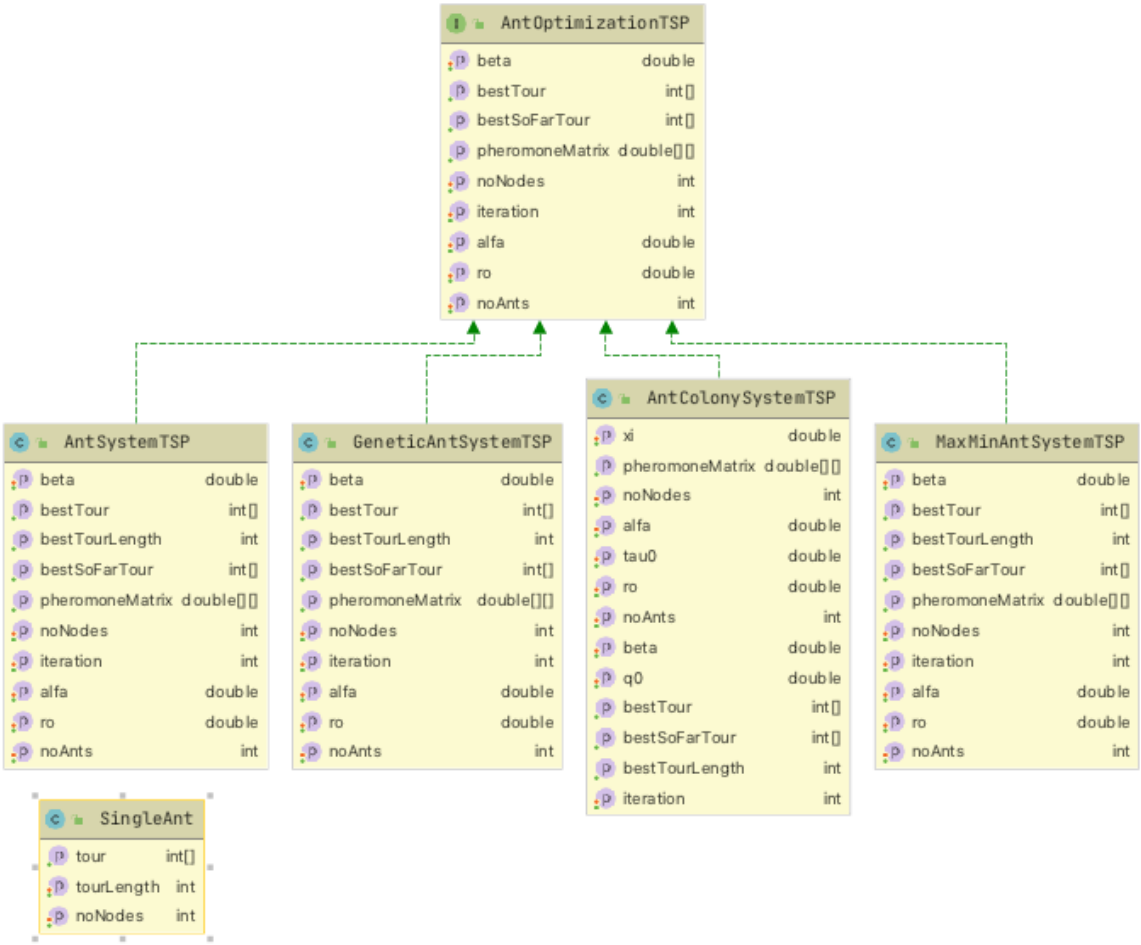

Рисунок 4.1 – Взаємозв’язок між основними класами програми

В програмі використаний алгоритм фабрики, який передбачає формування загального інтерфейсу, реалізує цілу низку методів згідно

традиційного мурашиного алгоритму і генетичного алгоритму, які завжди вважались лідерами у пошуку оптимальних рішень.

Основні відмінності у реалізації алгоритмів мурашиних колоній пов'язані з методом додавання феромонів, зокрема, в системі ACS (AntColonySystem) це реалізовано в процесі одержання рішень, тоді як в інших системах – після одержання рішень. Також реалізація ACS передбачає застосування традиційних методів локальної оптимізації. Цей метод мурашиних систем реалізується шляхом використання інформації, отриманої від попередніх агентів.

Це досягається за допомогою двох механізмів. По-перше, використовується сувора елітна стратегія при відновленні феромонів на ребрах. По-друге, агенти обирають наступний вузол для переміщення, використовуючи, так зване, псевдовипадкове пропорційне правило. За кращого агента може вважатися агент, який одержав краще рішення на даній ітерації, або глобально – кращий агент той, який одержав краще рішення на всіх ітераціях від початку роботи методу. Останньою відмінністю методу системи мурашиних колоній є те, що агенти оновлюють кількість феромонів у процесі складання рішення (подібно до щілястого й кількісного методів мурашиних систем). Такий підхід приводить до зменшення імовірності вибору однакових шляхів усіма агентами. За рахунок цього знижується імовірність зациклення в локальному оптимумі.

Як вже зазначалось клас SingleAnt – описує основні параметри окремої мурахи, варіанти руху та виділення феромонів. Інтерфейс AntOptimization – абстрактно описує моделі алгоритму для вирішення завдання комівояжера за допомогою Мурашиного алгоритму. На основі класу SingleAnt створюється мурашина колонія AntColony, яка реалізує різноманітні варіанти пошуку оптимального маршруту. Для цього використовується вже окремий клас, який реалізує різноманітні методи оптимізації.

Була розроблена високорівнева архітектура додатку, яка зображена на рисунку 4.2.

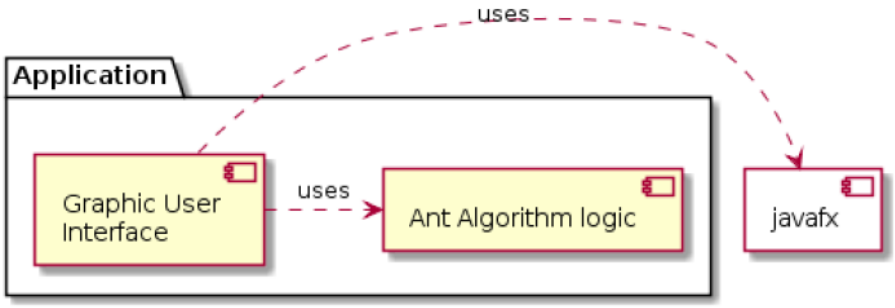

Рисунок 4.2 – Загальна схема архітектури програми

У програмі «Мурашиного алгоритму» реалізована трирівнева архітектура :

– найнижчий рівень – інтерфейси (абстрактний опис методів програми);

– класи (параметри та конкретна реалізація методів);

– графічні вікна (візуальне представлення).

Для реалізації користувацького інтерфейсу скористались графічною бібліотекою JavaFX, яка дозволяє проектувати і кодувати складні елементи і будувати взаємодію з ними, саме тому вона була обрана в якості базової бібліотеки.

Програмна реалізація абстрактних методів здійснена в межах головних класів.

Графічний інтерфейс (рисунок 4.3) реалізований у форматі xml, який легко інтегрується з бізнес-логікою мурашиного алгоритму.

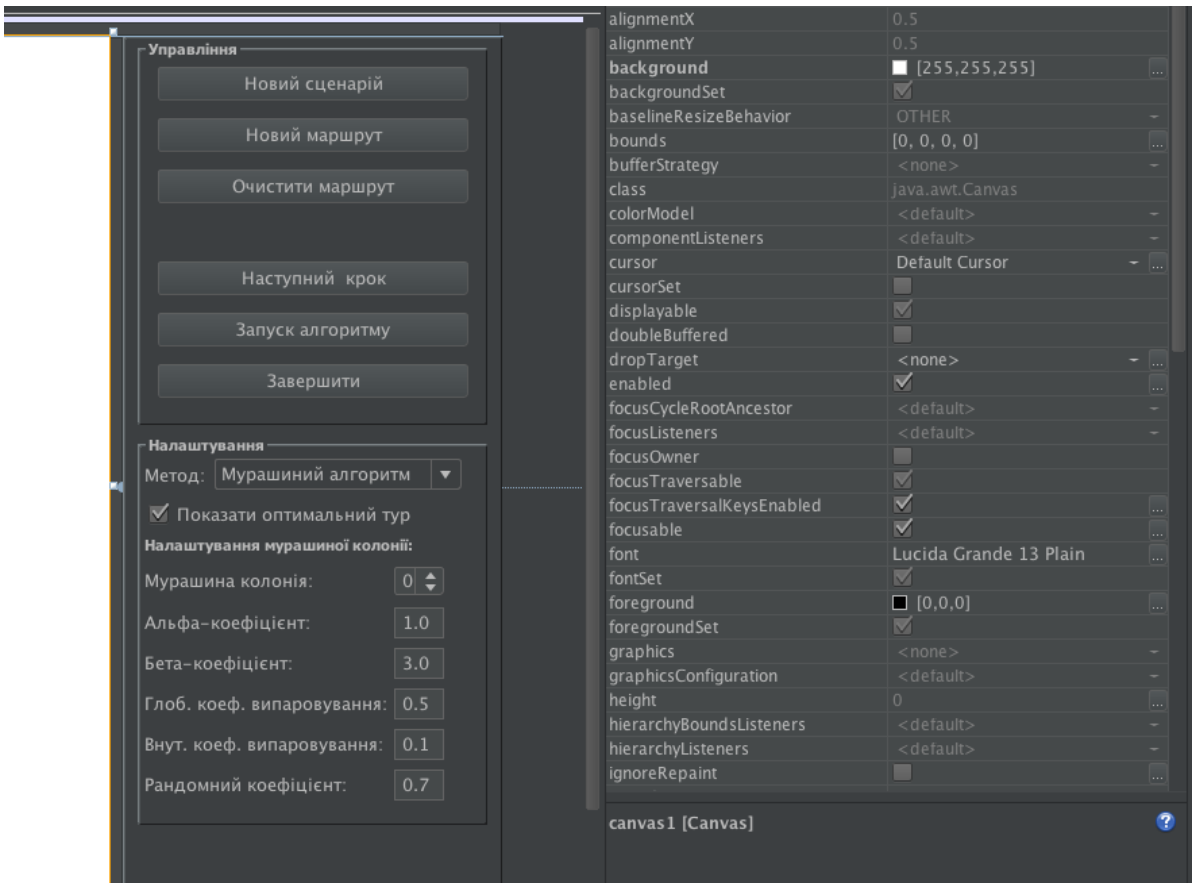

Рисунок 4.3 – Проектування графічного інтерфейсу програми «Мурашиний алгоритм»

4.3 Проектування математичного забезпечення

Механізм головного алгоритму програми реалізує математичну модель, описану у другому розділі роботи, а саме поведінки окремих мурах, які можуть ходити у будь-якому порядку (випадкових значень) через міста, відкладаючи феромони, і передавати інформацію щодо руху у загальну колонію мурах. Після проходу через обраний маршрут мурахи залишають феромон, і там, де слідів від феромону більше, мурахи починають ходити частіше. З кожною ітерацією мурашиного алгоритму, шляхи на яких залишено меншу кількість феромонів втрачають популярність серед мурах і значення феромонів у програмі скорочується, а то й обнуляється у випадку, якщо мурахи зовсім перестають ходити за

даними маршрутами. Вирішення поставленого завдання вимагає наявності інформації щодо кількості вершин і відстаней між ними.

Для отримання інформації було обрано спосіб, який дозволяє створити граф безпосередньо у програмі. До основних вихідних параметрів моделі відносяться: кількість мурах, альфа- та бета-коефіцієнти, значення вершин міст (x,y), коефіцієнти (глобального і локального) випаровування та випадковий коефіцієнт вибору мурахами міст. Результатом програми є визначення оптимального маршруту руху мурах.

Головний клас AntColonySystem імплементує інтерфейс AntOptimization і реалізує низку методів, описаних у цьому інтерфейсі. До головних методів програми відносяться такі:

– initData() – метод ініціалізації початкових даних;
– initPheromones() – ініціалізація феромонів;
– computeHeuristic() – метод евристичного розрахунку;
– initAnts() – ініціалізація мурашиної колонії
– globalEvaporation() – метод випаровування;
– updatePheromones() – метод оновлення феромонів;
– localSearch() – метод локального пошуку;
– int[] getBestSoFarTour() – метод отримання масиву найкращих маршрутів;
– updateBestSoFarTour() - метод оновлення найкращого рішення;
– double[][] getPheromoneMatrix() – отримання матриці феромонів;
– constructSolutions() –конструктор рішень.

Нижче наведені окремі приклади реалізації коду (рисунок 4.4 – 4.9). Зокрема, початкова умова по ініціалізації колонії.

```
public void initAnts(){
   int i,j;
   for(i = 0; i < noAnts; i++){
      ants[i].setTourLength(0);

      for(j = 0; j < noNodes; j++)
         ants[i].setVisited(j, false);
      for(j = 0; j <= noNodes; j++)
         ants[i].setTour(j, 0);
   }
}
```

Рисунок 4.4 – Створення (ініціалізація) колонії

Механізм оновлення феромонів показаний за правилом, яке реалізовано у наступному методі.

```
public void localPheromoneUpdate(int ant, int step){

  int idx1 = ants[ant].getTour(step);
   int idx2 = ants[ant].getTour(step-1);

   double currentValue = pheromone[idx1][idx2];
   pheromone[idx1][idx2] = (1-xi)*currentValue+xi*tau0;
   pheromone[idx2][idx1] = pheromone[idx1][idx2];

   double niu = 0;
   if(dist[idx1][idx2] > 0)
      niu = 1.0/dist[idx1][idx2];
   else
      niu = 1.0/0.0001;

   choiceInfo[idx1][idx2] = Math.pow(pheromone[idx1][idx2],alfa)*Math.pow(niu,beta);
   choiceInfo[idx1][idx2] = choiceInfo[idx2][idx1];
}
```

Рисунок 4.5 – Метод реалізації локального оновлення феромонів

Також використовувались математичні бібліотеки для визначення відстані між двома містами.

```
public int getIntegerDistance(PointCoords p){
   return (int)Math.round(Math.sqrt(Math.pow(this.x-p.x, 2) + Math.pow(this.y-p.y,2)));
}
```

Рисунок 4.6 – Розрахунок відстані між двома містами

І як варіант, приклад реалізації підрахунку загальної довжини маршруту для окремої мурахи.

```
public int computeTourLength(int tour[]){
   int len = 0;
   for(int i = 0; i < noNodes; i++){
      len+=dist[tour[i]][tour[i+1]];
   }
   return len;
}
```

Рисунок 4.7 – Розрахунок довжини маршруту

Програма зчитує вихідні параметри роботи алгоритму з форми. Увідповідні параметри Nodes, alfa, beta, globalEvapRate, localEvapRate, pseudoRandCoef, mutationRatio заносяться дані з графічного інтерфесу, а саме відповідниї текстових полів:

– alfa = Double.parseDouble(jTextField1.getText());
– beta = Double.parseDouble(jTextField2.getText());
– globalEvapRate = Double.parseDouble(jTextField3.getText());
– localEvapRate = Double.parseDouble(jTextField4.getText());
– pseudoRandCoef = Double.parseDouble(jTextField5.getText());
– mutationRatio = Double.parseDouble(jTextField6.getText());

Далі створюється об’єкт системи колонії мурах.

as = new AntSystemTSP(noNodes, noAnts, alfa, beta, globalEvapRate, doOpt2, doOpt3);

Ініціалізує вихідні дані щодо вершин, здійснюється побудова графів і відбувається ініціалізація феромонів та мурашиної колонії:

1) as.initData();
2) as.initPheromones();
3) as.computeHeuristic();
4) as.initAnts();
5) drawCanvas();

Після написку на кнопку «Запуск алгоритму», запуск у циклі головного потоку run здійснюється запуск циклу while в якому виконуються рішення щодо основного алгоритму.

```
java.awt.EventQueue.invokeLater(new Runnable() {
      public void run() {
         new GraphicalSolverFrame().setVisible(true);
      }
   });
}
class RunThread extends Thread{
   @Override
   public void run(){
      while(currentState == RUNNING){
         step();
         try{
            sleep(50);
         }
         catch(Exception e){
            System.err.println("Thread sleep error.");    }    }    }
```

Рисунок 4.8 – Код запуску програми

У випадку запуску основного алгоритму «*RUNNING*» програма проводить розрахунки у методах, що відповідаютьза зміни у систему руху мурах і переносить оновлені дані на графічну форму.

```
private void step(){
   if(((String)jComboBox1.getSelectedItem()).equals("Мурашитий алгоритм")){
      as.constructSolutions();
      as.localSearch();
      as.updatePheromones();
      as.setIteration(as.getIteration()+1);
   }
drawCanvas();   }
```

Рисунок 4.9 – Запуск основних методів мурашиного алгоритму в циклі while запущеного потоку

Розрахунок основного алгоритму здійснюється у методі step(), що налаштований на зміну після 50 мс: у випадку зміни currentState на FINISH програма виходить із циклу while, але робота потоку не припиняється. Якщо внести зміни у початкові дані, то можна здійснювати наступні розрахунки.

Тимчасова складність алгоритму оцінена як:

$$Q(t*m*n^2), \tag{4.1}$$

де t – число ітерацій,

m – кількість мурах,

n – кількість вершин у стовпці.

При випаровуванні феромона слід враховувати, що рівень феромона на ребрах не повинен досягати нуля, інакше перехід за такими ребрами унеможливлюється і отримане значення буде субоптимальним.

4.4 Розробка графічного інтерфейсу

Розробка графічного інтерфейсу (рисунок 4.10) GUI Designer Form, що дозволяє у мінімальні строки за допомогою компонентного підходу розробити зручний інтерфейс для кінцевих користувачів даної програми. Для цього існує велика кількість форм, діалогових вікон та різноманітних контейнерів та компонентів (елементів).

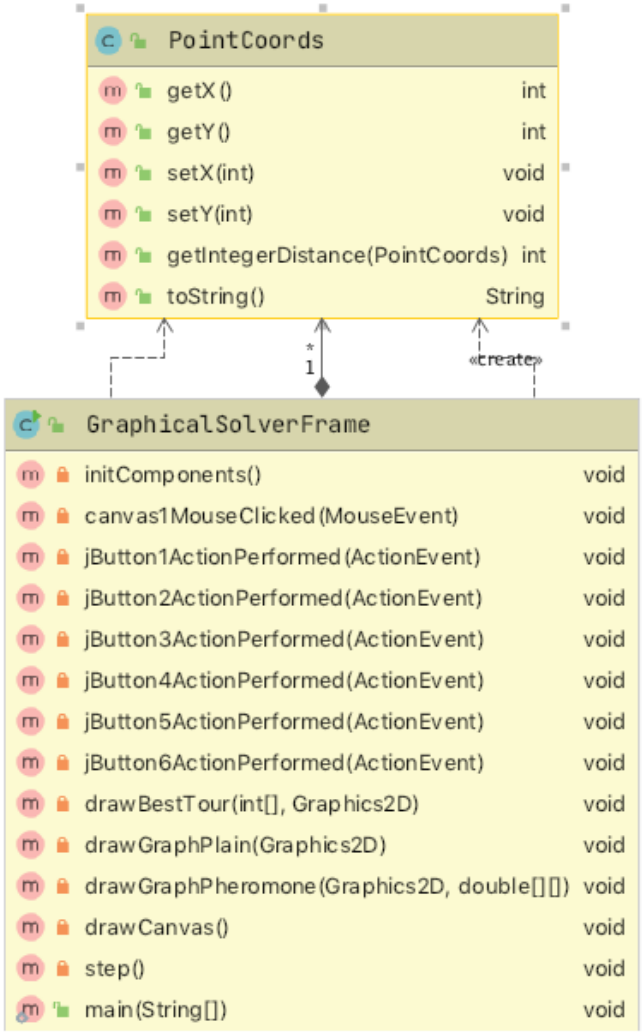

Рисунок 4.10 – Реалізація графічного інтерфейсу програми

До основних панелей програми можна віднести такі, як:

1) блок управління алгоритмом програми (рисунок 4.11);

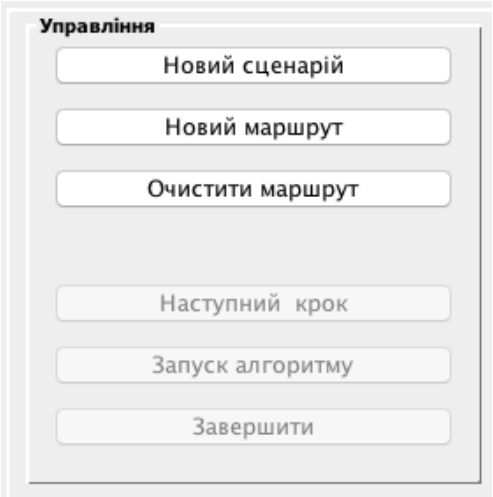

Рисунок 4.11 – Блок управління головним алгоритмом програми

2) блок налаштування (рисунок 4.12);

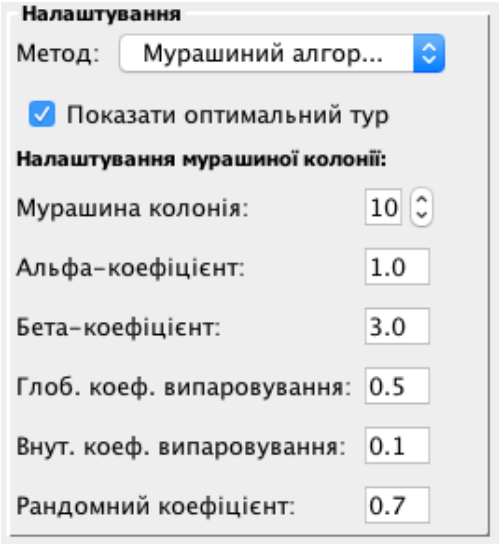

Рисунок 4.12 – Блок налаштування параметрів алгоритму програми

3) графічне вікно, в якому відображається рух мурах за визначеним графом користувача.

Окрім відображення графу, в інтерфейсі додатку також передбачені компоненти для відображення маршруту та інформаційні елементи для відображення кількості ітерацій запуску програми та загальної відстані, пройденої мурахами.

Для побудови блоку налаштувань обрано використання звичайних кнопок бібліотеки SWING (JButton). Користувач може запускати програму, зупиняти, задавати та очищувати початкову умову, задавати та очищувати розв’язки програми, покроково переглядати розв’язок програми, перезапускати програму.

Блок налаштування повинен містити параметри створення мурах, мурашиної колонії, альфа- та бета-коефіцієнти, коефіцієнти випаровування феромонів та коефіцієнт випадкового вибору руху мурах.

4.5. Опис роботи програми «Мурашиного алгоритму»

Результатом розробки програми «Мурашиного алгоритму» є комп’ютерна програма, інтерфейс якої представлений на рисунку 4.13.

Користувач у вікні програми самостійно розміщує вершини в необхідних місцях (рисунок 4.14). Для цього спочатку необхідно натиснути кнопку «Новий сценарій» і перейти до побудови вершин. У відповідності до заданого прикладу розмістити вершини у графічному вікні. Побудова графу здійснюється автоматичним зв’язуванням його вершин. Приклад задання вершин та побудови графу показаний на рисунку.

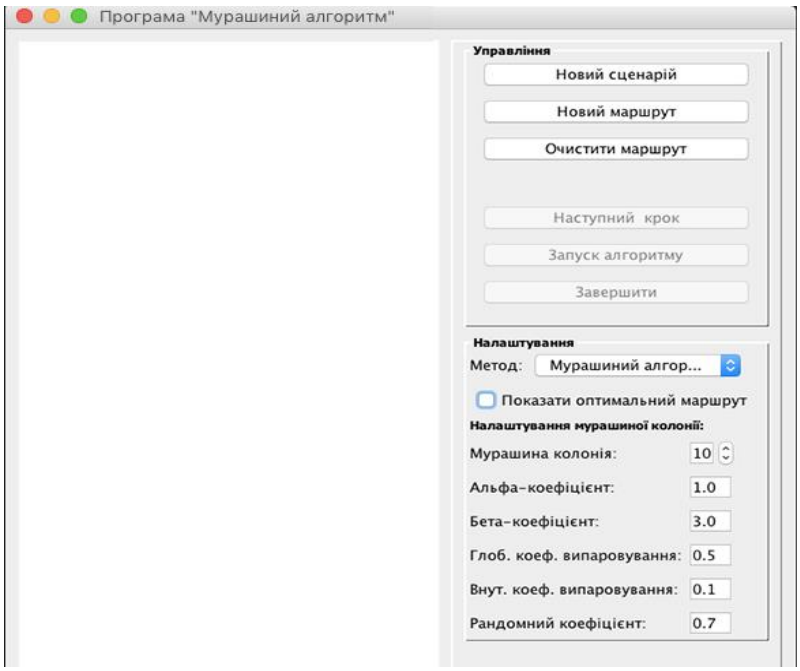

Рисунок 4.13 – Головний інтерфейс програми «Мурашиний алгоритм»

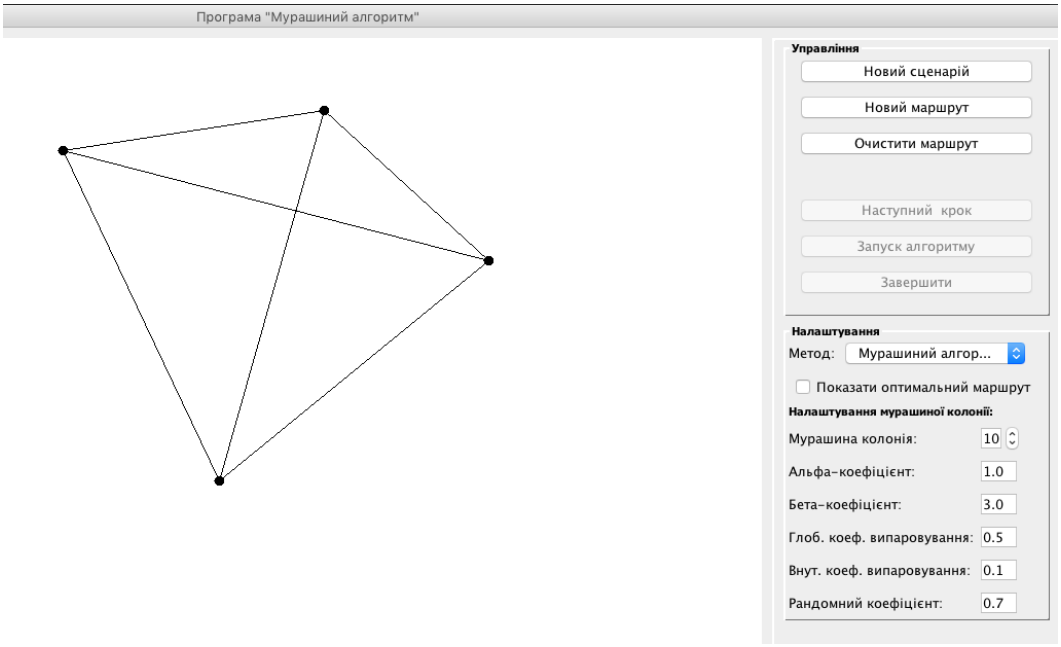

Рисунок 4.14 – Приклади автоматичного зв’язування вершин (побудова ребер) графу

Після побудови графу у додатку програми необхідно задати початкові параметри. Для цього необхідно перейти до вкладки «Налаштування», яка була представлена на наступному рисунку.

Програма дозволяє змінювати основні параметри моделі:

– популяцію мурашиної колонії,

– альфа- та бета-коефіцієнти,

– коефіцієнти випаровування феромонів,

– коефіцієнт випадкового вибору руху мурах.

Після задання початкового етапу (рисунок 4.15) користувач на власний вибір може здійснити ручний або автоматичний пошук рішень.

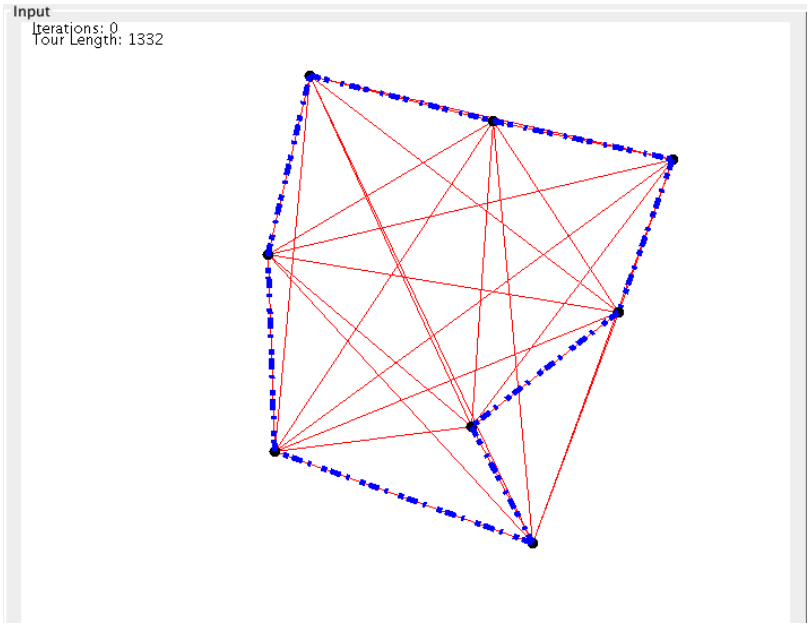

Рисунок 4.15 – Приклад задання початкового стану системи

Для цього передбачені такі кнопки як «Запуск алгоритму» (автоматичний режим роботи) або перейти на «Наступний крок» (ручний режим роботи). Операції пошуку оптимальних рішень можуть бути перезапущені або зупинені. Користувач також може очистити сценарій, задати нові вихідні дані та знову розрахувати оптимальний маршрут.

Застосування ручного режиму дозволяє переглядати зміни, що відбуваються після кожної ітерації (рисунок 4.16 – 4.17).

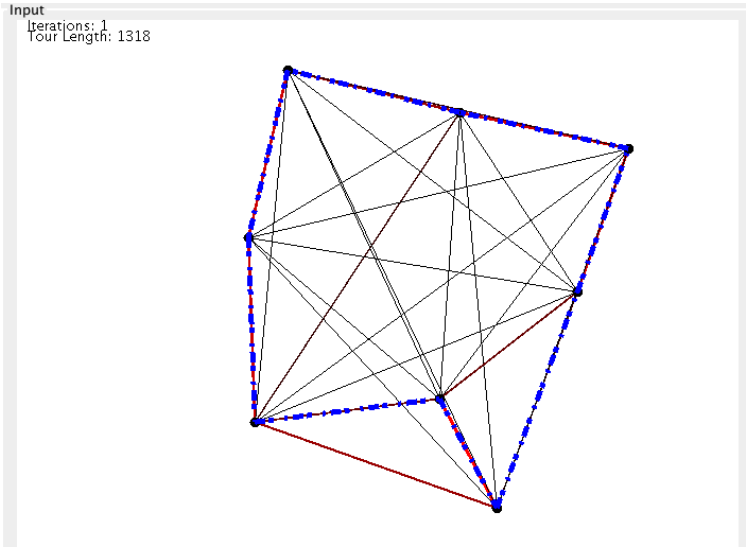

Рисунок 4.16 – Зміна роботи програми після першої ітерації

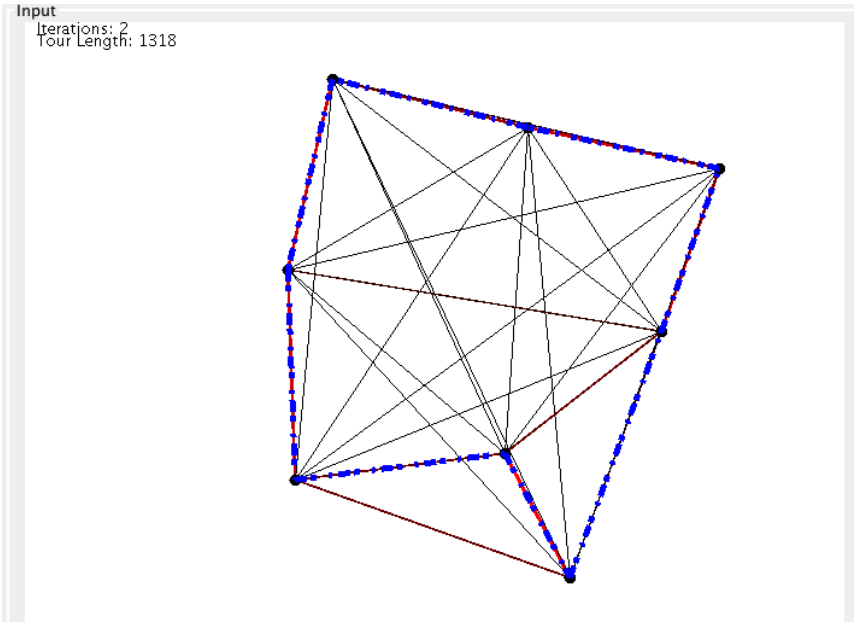

Рисунок 4.17 – Зміна роботи програми після другої ітерації

Для того, щоб користувачеві перейти до автоматичного визначення оптимального маршруту за моделлю комівояжера, потрібно натиснути

кнопку «Запуск алгоритму», тоді програма автоматично почне розраховувати та відображати зміни оптимальних відстаней на графіку. Оптимальний шлях відображається у верхній лівій частині екрану і містить інформацію щодо пройденого шляху (Tour Length) та кількості ітерацій.

Зупинка програми може буде здійснена у будь-який момент після натискання на кнопку «Завершити»: надається вибір між таки функціями як перезапуск або формування нового сценарію, задання початкового маршруту або новий запуск програми.

У програмі «Мурашиного алгоритму» присутня функція, що дозволяє виділяти найоптимальніший шлях руху мурах. Для цього потрібно ввімкнути певну опцію – перемикнути галочку у нижньому меню «Показати оптимальний маршрут», що показано на рисунку 4.18.

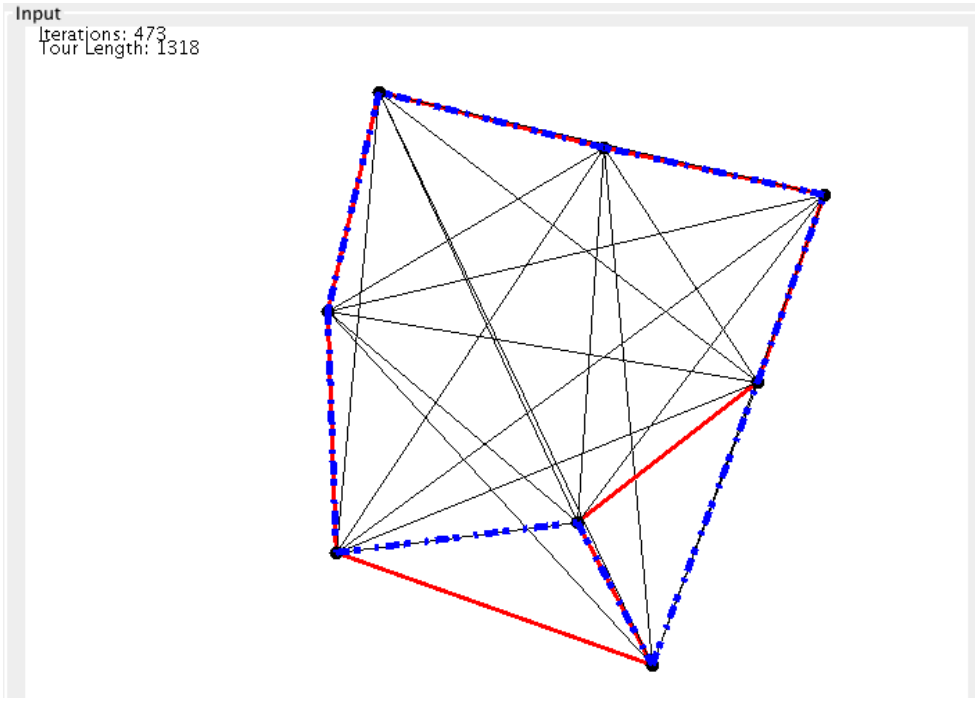

Рисунок 4.18 – Результат роботи (початковий та оптимальний маршрути) у програмі «Мурашиний алгоритм»

Результатом роботи програми є створення нового графу з індивідуальними параметрами. Після здійснення розрахунків графік може бути опрацьований та проаналізований аналітиком (користувачем). Для користувача доступними є такі показники, як: масив точок між містами [xOld, yOld, xNew, yNew]; кількість ітерацій та загальна відстань, пройдена мурахами (рисунок 4.19).

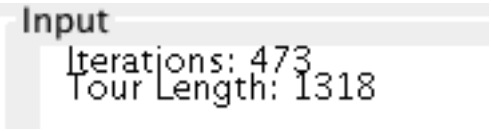

Рисунок 4.19 – Представлення загальної кількості ітерацій та довжини маршруту

Рисунок демонструє залежність часу пошуку і знайденого оптимального маршруту від значень регульованих параметрів.

Висновки до розділу 4.

Обґрунтовано необхідність побудови трирівневої архітектури програми «Мурашиного алгоритму» з виділенням таких рівнів, як інтерфейси (абстрактні описи методів програми), класи (параметри та конкретна реалізація методів) та графічні вікна.

Визначено, що для реалізації обраного шаблону програми «Мурашиний алгоритм» доцільно скористатись: на рівні математичної реалізації – базовими бібліотеками стандартного пакету JDK, а на рівні графічного інтерфейсу – технологіями JavaFX, бо ці технології дозволять запускати виконувані файли на будь-який ОС і виводити на екран складні інтерфейси.

Головний клас програми «Мурашиний алгоритм» є AntColonySystem, що імплементує інтерфейс AntOptimization, реалізує низку методів: ініціалізації початкових даних; ініціалізації мурашиної

колонії; ініціалізації феромонів; випаровування та оновлення феромонів; евристичного розрахунку; отримання масиву найкращих маршрутів; оновлення найкращого рішення та конструктор рішень.

Для користувача програми передбачений наступний функціонал: задання початкової умови (вершин і графів), блок управління та блок налаштування алгоритмів розрахунків (початкової популяції мурашиної колонії, альфа- та бета-коефіцієнтів, коефіцієнтів випаровування феромонів та коефіцієнту випадкового вибору руху мурах) та графічне відображення результатів побудови найкращого рішення.

Проведене тестування в ручному та автоматичному режимі дозволяє стверджувати, що основні алгоритми працюють коректно і програма є ефективною для вирішення задач пошуку оптимальних маршрутів між містами.

ВИСНОВКИ

Робота призначена дослідженню алгоритмів еволюційного моделювання та їх застосування в ІСПР.

В роботі було досліджено теоретичні положення розвитку ІСПР та СППР, визначено, що ІСПР має три покоління. ІСПР, на відміну від СППР, ґрунтується на використанні моделей і процедур з обробки даних та знань на основі технологій штучного інтелекту. Було досліджено, що ІСПР має наступну структуру: бази даних і знань, бази моделей, механізми висновків, система накопичення та актуалізація знань, блок пояснень, організації діалогової взаємодії з користувачем.

Здійснено огляд наступних видів алгоритмів: еволюційне програмування, еволюційні стратегії, генетичні алгоритми, алгоритми мурашиних колоній, алгоритми бджолиного рою, алгоритми інтелектуального пошуку (пошук табу), роєві алгоритми. Суть цих алгоритмів полягає в тому, що їх реалізація заснована принципах природнього еволюційного процесу.

Проведений аналіз застосування еволюційних алгоритмів, яке залежно від сфери використання, ступеня автономності та призначення який розглядають в наступних напрямках: спеціалізоване програмне забезпечення, додатки до математичних та аналітичних пакетів, фреймворки.

Розглянуто найбільш поширені алгоритми еволюційного моделювання. Визначено, що в математичну основу цих алгоритмів покладено використання базових принципів теорії біологічної еволюції, а саме відбору, мутації і відтворення особин.

Спільні риси, притаманні цим алгоритмам, включають такі кроки: ініціалізація алгоритму та задання значень вільних параметрів алгоритму, застосування пошукових (міграційних) операторів алгоритму до поточних станів агентів, в результаті чого вони набувають нові положення,

здійснення перевірки виконання умов завершення ітерацій. Якщо ці умови не виконані, повернення до виконання алгоритму, в іншому випадку вважаємо, що рішення знайдено.

Проведено аналіз математичного опису наступних еволюційних алгоритмів: генетичні алгоритми, еволюційна стратегія, алгоритми інтелектуального пошуку (табу-пошук), алгоритми оптимізації колонією мурах.

Відповідно аналізу літератури визначено, що алгоритм еволюційної стратегії, на відміну від генетичних алгоритмів, передбачає, що на кожній ітерації спочатку проводиться модифікація особин (мутація), а потім селекція. Класичний генетичний алгоритм виконує ці дії в зворотному порядку.

Основою канонічного алгоритму еволюційної стратегії є комбінування генетичних операторів мутації і селекції. Для вирішення задач планування та кластерного аналізу доречно застосувувати алгоритми інтелектуального пошуку, основним механізмом, який дозволяє алгоритму TS долати локальні екстремуми, є список табу.

В роботі розглянуто алгоритм для вирішення задач безперервної, оптимізаціїї алгоритм оптимізації колонією мурах CACO. Основними особливостями алгоритму CACO є комбінування глобального і локального пошуку, використання для глобального пошуку генетичного алгоритму, використання для локального пошуку тільки каналу стігмергії.

Алгоритми мурахи мають багато різновидів і широко застосувуються у вирішенні різноманітних задач.

Таким чином, еволюційні алгоритми зазвичай використовуються для загального опису алгоритмів пошуку, оптимізації або навчання, що базується на формалізованих принципах природнього еволюційного процесу

Визначена структура ІСПР, яка містить наступні компоненти: база даних, система керування базою даних, база моделей, інтерфейс

користувача.

Вдосконалено модель пошуку оптимального маршрутку доставки вантажу, передбачивши в моделі можливість повернення в ту ж точку призначення в різний часовий інтервал. В запропонована модель представлена як ітераційна лінійна модель, яка розглядає оптимізацію маршруту за певні проміжки часу.

В роботі описані моделі бізнес-процесів підприємства, а саме бізнес-процеси «Оформлення замовлення на доставку» та «Побудова оптимального маршруту доставки». Моделі даних бізнес-процесів виконані відповідно нотацій BPMN.

Визначені функціональні вимоги до системи, а саме: система повинна вести облік адрес доставок, розраховувати відстані між пунктами доставки, будувати оптимальні маршрути руху доставки.

В середовищі Enterprise Architect v14 побудовані моделі діяльності системи, а саме: Use Case, діаграми діяльності та діаграма класів.

Продемонстровано реалізацію алгоритму мурашиних колоній на прикладі розробленої програми «Мурашиний алгоритм». В програмі запропоновано трирівневу архітектуру з такими рівнями: інтерфейси (абстрактні описи методів програми), класи (параметри та конкретна реалізація методів) та графічні вікна.

Визначено, що для реалізації обраного шаблону програми «Мурашиний алгоритм» доцільно скористатись: на рівні математичної реалізації – базовими бібліотеками стандартного пакету JDK, а на рівні графічного інтерфейсу – технологіями JavaFX, бо ці технології дозволять запускати виконувані файли на будь-який ОС і виводити на екран складні інтерфейси.

Для користувача програми передбачений наступний функціонал: задання початкової умови (вершин і графів), блок управління та блок налаштування алгоритмів розрахунків (початкової популяції мурашиної колонії, альфа- та бета-коефіцієнтів, коефіцієнтів випаровування

феромонів та коефіцієнту випадкового вибору руху мурах) та графічне відображення результатів побудови найкращого рішення.

Проведене дослідження підтверджує ефективність вирішення прикладних задач на основі алгоритмів еволюційного моделювання.

Запропонована ІСПР може бути використана в транспортних компаніях для оптимального розподілу маршрутів при доставці вантажів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ситник В. Ф. Системи підтримки прийняття рішень: Навч. посіб. К.: КНЕУ, 2004. 614 с.

2. Нестеренко О. В. Інтелектуальні системи підтримки прийняття рішень: Навч. посібн. / За ред. П.І. Бідюка. Київ: Національна академія упр авління, 2016. 188 с.

3. Вітлінський В. В. Штучний інтелект у системі прийняття управлінських рішень. *Нейро-нечіткі технології моделювання в економіці*. 2012. №1. С. 97–112.

4. Литвин В. В. Інтелектуальні системи підтримки прийняття рішень на основі адаптивних онтологій. *Искусственный интеллект*. 2011. №3. С. 388–395.

5. Золотухин О. В. Исследование методов искусственного интеллекта в системах бесконтактных платежей. *Сталий розвиток України, проблеми та шляхи їх подолання* : матеріали Міжнародної науково-практичної конференції (Маріуполь, 14-15 листопада 2019 р.). 2019. С. 462–464.

6. Интеллектуальные системы принятия решений в нештатных ситуациях с использованием информации о состоянии природной среды / Головани В. А. и др. М.: Эдиториал УРСС, 2001. 304 с

7. Bonozek R.H., Holsappe C.W., Whinston A.B. Foundations of Decision Support Systems. New York: Academic Press, 1981.

8. Черкаський М. В. Еволюція тлумачення поняття «Алгоритм». *Вісник Національного університету «Львівська політехніка»*. 2003. № 492: Комп’ютерні системи та мережі. С. 142-146.

9. Каширина И. Л. Введение в эволюционное моделирование. Учебное пособие. Воронеж: ВГУ, 2007.

10. Емельянов В. В. Теория и практика эволюционного моделирования. М.: ФИЗМАТЛИТ, 2003. 432 с.

11. Азбука ИИ: «Эволюционные алгоритмы». URL: https://nplus1.ru/material/2016/07/06/evodevo (дата звернення 03.04.2020).

12. Гулаєва Н. М. Еволюційні алгоритми. *Вісник Київського національного університету імені Тараса Шевченка*. 2013. №2. С. 141–150.

13. Іщук Я. Ю. Про деякі аспекти навчання еволюційних алгоритмів. *Науковий часопис НПУ імені М. П. Драгоманова.* Серія 2 : Комп'ютерно-орієнтовані системи навчання. 2017. № 19. С. 246-251.

14. Смагин А. А. Интеллектуальные информационные системы: учебное пособие. Ульяновск: УлГУ, 2010. 136 с.

15. Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой : учебное пособие. Москва: Издательство МГГУ им. Н. Э. Баумана, 2017. 446 с.

16. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г. К.Вороновский и др. Х.: ОСНОВА, 1997. 112 с.

17. Талімончик А. С. Дослідження та вдосконалення методів світлячків і зграї вовків при розв’язуванні задач оптимізації і машинного навчання. 2016. №1. С. 77–89.

18. Бабич С. О. Алгоритми ройового інтелекту для задачі динамічного розподілу навантаження. *Науковий огляд*. 2018. №3.

19. Іщук Я. Ю. Про деякі аспекти навчання еволюційних алгоритмів. *Науковий часопис Національного педагогічного університету імені М. П. Драгоманова*. Серія 2 : Комп’ютерно-орієнтовані системи навчання : зб. наук. праць. Київ : Вид-во НПУ імені М. П. Драгоманова, 2017. Вип. 19 (26). С. 246-251.

20. Кононюк А. Ю. Нейроні мережі і генетичні алгоритми. Київ, 2008. 446 с. .

21. Еволюційні стратегії. URL: https://studfile.net/preview/4494757/page:13/ (дата звернення 26.04.2020).

22. Штовба С.Д. Муравьиные алгоритмы. *Exponenta Pro. Математика в приложениях*. 2003. №4. С. 70–75.

23. Муравьиные алгоритмы URL: https://asvk.cs.msu.su/sites/all/themes/professional_theme/files/%D0%9B9_%D0%9C%D1%83%D1%80%D0%B0%D0%B2_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B.pdf (дата звернення 27.04.2020)

24. Использование алгоритмов муравьиных колоний при решении задач оптимизации календарного планирования. URL: http://masters.donntu.org/2018/fknt/strelnikov/library/article4.pdf (дата звернення 20.04.2020)

25. Ситник Н. В. Проектування баз і сховищ даних. Київ: КНЕУ, 2004. 348 с.

26. Эль-Хатиб С. А. Компьютерная система сегментации медицинских изображений методом муравьиных колоний. *Радіоелектроніка, інформатика, управління*. 2015. С. 49–57.

27. Мартынов А. В. Гибридный алгоритм решения задачи коммивояжера. *Известия ЮФУ. Технические науки*. 2015. С. 36–44.

28. Васянин В. А. Задачи построения доставочных и сборочных маршрутов перевозки мелкопартионных грузов во внутренних зонах иерархической автотранспортной сети. *Математичне моделювання в економіці*. 2016. С. 102–131.

29. Гладков Л. А. Гибридный алгоритм решения транспортных задач с ограничением по времени. *Известия Южного федерального университета. Технические науки*. 2015. №6. С. 180–191.