КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

ОПТИМІЗАЦІЯ МАРШРУТІВ НА ОСНОВІ МУРАШИНИХ АЛГОРИТМІВ

Виконав: студентка 4-го курсу
Єлизавета ЗАЯЦЬ _____
(підпис)

Науковий керівник:
професор кафедри теоретичної кібернетики
доктор фіз.-мат. наук, професор
Анатолій ПАШКО _____
(підпис)

Засвідчую, що в цій роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри теоретичної кібернетики
«01» _____06_____ 2023 р.,
протокол № ___
Завідувач кафедри
доктор фіз.-мат. наук, професор
Юрій КРАК _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи: 41 сторінок, 21 ілюстрацій, 1 таблиця, 11 використаних джерел.

У цій роботі розглянуто використання алгоритму мурашиного колонії для планування туристичного маршруту на гірській місцевості та пошуку оптимального рішення. Спочатку було вивчено визначення планування маршруту та порівняння алгоритмів, здатних вирішувати визначену проблему. Потім був описаний метод моделювання планування маршруту. Також було проаналізовано поточний стан досліджень тривимірного планування туристичного маршруту з прорахунком перешкод на змодельованій гірській місцевості, а також існуючі проблеми планування тривимірного шляху.

Вказані основні теоретичні знання поклали фундамент для дослідницької частини роботи. Використана програмне забезпечення Intellij IDEA та мову програмування Java для проведення експериментів роботи з покращеним мурашиним алгоритмом з проекцією на двовимірний випадок пошуку та планування оптимального туристичного маршруту на гірській місцевості, імітуючи поведінку мурашиної колонії.

В подальшому оптимізований алгоритм можна застосовувати у сфері робототехніки для навчання роботів з метою роботи в тривимірному просторі. Наразі планування шляху та навчання є актуальними темами у робототехніці.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі швидкість та ефективність переміщення мають велике значення у багатьох сферах життя, включаючи логістику, транспорт, розподіл ресурсів та маршрутизацію. Побудова оптимального шляху має велике практичне значення і є актуальною проблемою в багатьох країнах, включаючи Україну та інші країни світу.

В Україні, зокрема, шляхи та маршрутизація мають велике значення в багатьох сферах, таких як транспортна логістика, автомобільний рух, доставка товарів та послуг, а також в туризмі. Оптимальне планування шляху може значно зменшити час та витрати на переміщення, покращити безпеку на дорогах та знизити вплив на навколишнє середовище. Врахування таких факторів, як дорожні умови, трафік, обмеження шляхів та відстані, стає ключовим завданням у вирішенні цих проблем.

В світовому контексті, пошук та побудова оптимального шляху є однією з найважливіших задач в галузі комбінаторної оптимізації. Ця проблема вивчається в багатьох країнах і використовується для вирішення різноманітних завдань, таких як навігація в GPS-системах, маршрутизація в мережах зв'язку, планування маршрутів для дронів, переміщення техніки та багато інших.

В рамках даної роботи буде розглянуто одне з таких завдань, що полягає у побудові раціонального логістичного маршруту.

Побудова оптимального шляху для туристів на гірській місцевості з використанням мурашиного алгоритму вимагає уваги до деяких проблем та особливостей. Перш за все, географічні обмеження, такі як різні типи терену, висотні різниці та природні перешкоди, впливають на доступність та проходимість деяких точок. Крім того, гірська місцевість може мати кілька варіантів шляхів до однієї точки, і вибір оптимального шляху потребує врахування відстані, часу проходження, складності маршруту та висотних різниць. Врахування факторів безпеки, таких як круті схили та небезпечні умови, є також важливим. Значною мірою успішність побудови залежить від

врахування взаємозв'язків між точками, таких як послідовність проходження та взаємна залежність. Природні резервати та обмеження доступу повинні бути дотримані для збереження екологічної рівноваги. Необхідно також враховувати фізичні обмеження туристів, які можуть мати обмежену фізичну форму або мобільність. На практиці, використання мурашиного алгоритму вимагає налаштування параметрів та проведення ітеративного процесу для забезпечення збалансованості між пошуком та збереженням знайдених рішень. Врахування цих проблем та особливостей сприяє успішній побудові оптимального шляху для туристів на гірській місцевості.

Математичні методи можуть використовуватися для пошуку оптимальних розташувань, групування, послідовності або виключення дискретних подій. Ці задачі належать до класу NP-повних питань, які не можуть бути вирішені за поліноміальний час.

З появою біоніки в середині 1950-х років люди постійно надихаються механізмами біологічної еволюції. Було запропоновано багато нових методів для вирішення складних оптимізаційних задач, таких як нейронні мережі, генетичні алгоритми, імітація відпалу та еволюційні обчислення. Ці нові методи успішно застосовуються для вирішення практичних задач.

У рамках цієї роботи надається найбільший інтерес мурашиними алгоритмам, натхненними спостереженнями за поведінкою справжніх мурах.

Мурашині алгоритми серйозно досліджуються європейськими вченими із середини 90-х років. Справа в тому, що застосування мурашиних алгоритмів є перспективним методом оптимізації, який базується на моделювання поведінки колонії мурах. Мурашині колонії, а також інші комахи, які живуть у колоніях, мають цікаві характеристики з точки зору колективної поведінки цих сутностей. Зокрема, мурашині колонії проявляють високоорганізовану соціальну структуру, що дозволяє їм самоорганізовуватися без централізованого контролера для виконання складних завдань, спрямованих на виживання всієї колонії. Ці можливості, такі як поділ праці, полювальна поведінка, сортування потомства та спільний транспорт, надихнули різні види алгоритмів мурашиних

колоній. Перший такий алгоритм був натхненний здатністю мурах знаходити найкоротший шлях між джерелом їжі та їх гніздом.

У всіх цих прикладах мурахи координують свою діяльність за допомогою стигмергії , яка є непрямим способом комунікації, посередником якої є зміни в середовищі. Рухаючись, мурахи відкладають феромони (хімічні речовини) на землю, щоб позначити шляхи, які можуть бути пройденими іншими членами колонії, які потім посилюють феромони на цьому шляху. Ця поведінка призводить до самозміцнювального процесу, який призводить до шляхів, позначених високою концентрацією феромонів, тоді як менш використовувані шляхи мають зменшену концентрацію феромонів через їх випаровування. Однак, справжні мурахи можуть обирати шлях, який не має найвищої концентрації феромонів, щоб знайти нові джерела їжі або скоротити шлях. (???)Оскільки в основі алгоритму лежить моделювання пересування мурах по різних шляхах, такий підхід може стати ефективним способом пошуку раціональних рішень для задач оптимізації логістичного маршруту на гірській місцевості.

На гірській місцевості при прокладанні шляху до вершини завжди з'являтися природні перешкоди. Такими перешкодами можуть виявитися схили, водойми або завалені стежки. У зв'язку з цим, актуальним завданням є тривимірне завдання побудови туристичного маршруту на гірській місцевості за умови наявності перешкод.

Метою роботи є розробка інформаційної системи, що реалізує вирішення тривимірного завдання побудови туристичного маршруту на гірській місцевості із застосуванням мурашиних алгоритмів.

Об'єктом дослідження є логістичні процеси побудови маршрутів на гірській місцевості з застосуванням мурашиних алгоритмів.

Предметом дослідження є тривимірне завдання побудови оптимального туристичного маршруту на гірській місцевості з застосуванням мурашиних алгоритмів.

Для досягнення поставленої мети дослідження передбачається розв’язання наведених нижче ***завдань***:

- порівняти різні алгоритми для розв'язання оптимізаційних завдань і провести їх порівняльний аналіз;

- вивчити теоретичну основу мурашиних алгоритмів та їх принципів;

- проаналізувати використання мурашиних алгоритмів для побудови оптимального туристичного маршруту на гірській місцевості;

- розробити алгоритм роботи програмного забезпечення, що реалізує оптимізацію побудови туристичного маршрут з урахуванням мурашиних алгоритмів;

- спроектувати та розробити програмне забезпечення, що реалізує оптимізацію побудови маршруту з використанням мурашиних алгоритмів.

Методологічну основу роботи відображають ряд наукових ***методів***: діалектичний метод наукового пізнання, що відображає взаємозв'язок між теорією і практикою, історичний аналіз, формально-логічний метод, системний підхід, порівняльне дослідження, статистичний експеримент та інші методи дослідження..

Структура роботи зумовлюється її метою та завданнями. Робота складається зі вступу, трьох розділів основного матеріалу, висновків та списку використаної літератури.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдань дослідження

У нашому дослідженні розглядається використання мурашиного алгоритму для розв'язання задачі побудови туристичного маршруту на гірській місцевості.

Метаевристичний метод оптимізації з використанням мурашиного алгоритму передбачає співпрацю колонії штучних мурах для знаходження ефективних рішень складних дискретних задач оптимізації. Співпраця є ключовим елементом алгоритмів: обчислювальні ресурси спрямовуються на групу відносно простих агентів (штучних мурів), які взаємодіють непрямо за допомогою стигмергії - непрямої комунікації, посередником якої є оточення. [8].

Застосування мурашиних алгоритмів у розв'язку проблеми пошуку оптимального туристичного маршруту на гірській місцевості обґрунтоване їх природним впливом, здатністю враховувати варіативність гірських умов, працездатністю у великому масштабі, гнучкістю та адаптивністю до змінних умов. Використання цих алгоритмів може сприяти знаходженню оптимальних маршрутів для туристів у гірських районах.

Особливості та умови зазначені вище дозволяють випробувати ефективність мурашиних алгоритмів для вирішення актуального тривимірного завдання побудови оптимального туристичного маршруту на гірській місцевості за умови наявності перешкод [11,12].

Метою роботи є розробка інформаційної системи, що реалізує вирішення тривимірного завдання побудови туристичного маршруту на гірській місцевості із застосуванням мурашиних алгоритмів.

Об'єктом дослідження є логістичні процеси побудови маршрутів на гірській місцевості з застосуванням мурашиних алгоритмів.

Предметом дослідження є тривимірне завдання побудови оптимального туристичного маршруту на гірській місцевості з застосуванням мурашиних алгоритмів.

Для досягнення поставленої мети дослідження передбачається розв’язання наведених нижче ***завдань***:

- порівняти різні алгоритми для розв'язання оптимізаційних завдань і провести їх порівняльний аналіз;
- вивчити теоретичну основу мурашиних алгоритмів та їх принципів;
- проаналізувати використання мурашиних алгоритмів для побудови оптимального туристичного маршруту на гірській місцевості;
- розробити алгоритм роботи програмного забезпечення, що реалізує оптимізацію побудови туристичного маршрут з урахуванням мурашиних алгоритмів;
- спроектувати та розробити програмне забезпечення, що реалізує оптимізацію побудови маршруту з використанням мурашиних алгоритмів.

1.2 Огляд алгоритмів пошуку оптимального шляху

Існують різні алгоритми вирішення задачі побудови оптимального маршруту за різних умов та конфігурації. Проведемо огляд відомих алгоритмів.

Алгоритм A*

Цей алгоритм також знаходить найкоротший шлях між двома вузлами, але враховує і евристичну оцінку відстані до цільового вузла. Він використовує дві функції оцінки: g(n), що представляє вартість пройденого шляху від початку до поточного вузла, та h(n), що оцінює вартість шляху від поточного вузла до цільового. Алгоритм спрямований на пошук шляху з найменшою вагою g(n) + h(n) [1].

Алгоритм пошуку в ширину

Цей алгоритм починає з початкового вузла і просувається до всіх сусідніх вузлів на тому ж рівні, перш ніж переходити до наступного рівня. Він шукає найкоротший шлях в ненаправленому графі без ваг ребер [3].

Алгоритм пошуку в глибину

Цей алгоритм просувається вздовж графу шляхом обходу вузлів до досягнення кінцевого вузла або стику з тупиковою ситуацією. Він не гарантує знаходження найкоротшого шляху, але може бути корисним для знаходження будь-якого шляху в графі. [2].

Алгоритм Дейкстри

Цей алгоритм знаходить найкоротший шлях між двома вузлами стартуючи від початкового вузла та просуваючись до кожного сусіднього вузла з найменшою вагою ребра. Він працює тільки для графів з невід'ємними вагами ребер [3].

Алгоритм Дейкстри зі стисненими даними

Алгоритми Дейкстри зі стисненими даними використовують пріоритетні черги (heap) для ефективного вибору найменшої ваги на кожному кроці. Наприклад, алгоритм Дейкстри з використанням бінарної купи дозволяє зменшити часову складність алгоритму до O((|E|+|V|)log|V|), де |E| - кількість ребер, а |V| - кількість вузлів у графі. [3].

Алгоритм Беллмана-Форда

Цей алгоритм використовується для знаходження найкоротшого шляху в орієнтованому або неорієнтованому графі, навіть якщо граф містить ребра з від'ємними вагами. Він може виявити наявність від'ємних циклів в графі. Алгоритм використовує принцип динамічного програмування та виконує релаксацію ребер графа протягом V-1 ітерацій (де V - кількість вузлів графа), щоб знайти найкоротші шляхи від початкового вузла до всіх інших вузлів [3].

Алгоритм Джонсона

Цей алгоритм також використовується для знаходження найкоротших шляхів в орієнтованому або неорієнтованому графі з вагами ребер, включаючи

ребра з від'ємними вагами. Алгоритм Джонсона базується на перетворенні графа шляхом додавання нового вузла та ребер до початкового графа. Потім він використовує модифікований алгоритм Беллмана-Форда для кожного вузла графа, щоб знайти найкоротші шляхи між усіма парами вузлів. Алгоритм Джонсона дозволяє ефективно знаходити найкоротші шляхи в графі навіть з від'ємними циклами. [3].

Генетичний алгоритм

Генетичні алгоритми походять від ідеї еволюції та природного відбору, і вони можуть ефективно вирішувати задачі оптимізації.

Основна ідея генетичних алгоритмів полягає в моделюванні процесу природного відбору та еволюції, щоб шукати оптимальні рішення. Алгоритм працює з популяцією індивідів, кожен з яких представляє потенційний розв'язок задачі. Вони проходять через ітерації, які включають етапи вибору, схрещування (комбінування генетичного матеріалу), мутації (невеликі випадкові зміни) та оцінки (визначення якості або придатності кожного індивіда) [2].

Мурашині алгоритми

Мурашині алгоритми - це метаевристичні алгоритми, натхненні поведінкою колоній мурах при пошуку їжі. Вони використовуються для розв'язання задач оптимізації, включаючи планування маршрутів, в яких шукається найкоротший шлях у графах з вагами.

Основна ідея мурашиних алгоритмів полягає в тому, що кожна "мураха" під час руху по графу залишає феромони на пройдених ребрах, що вказує на якість цього шляху. Інші мурашки приймають рішення щодо вибору шляху, керуючись якістю феромону на ребрах. Поступово, під впливом феромонів, відбувається утворення коротших шляхів.

Основні кроки та їх короткий опис в контексті побудови оптимальних маршрутів такі:

1. Ініціалізація феромонів: Кожному ребру графа призначається початкове значення феромону. Значення феромону може бути встановлене як стале, так і визначатися на основі певних евристичних правил.

2. Пошук шляху мурахами: Кожна мураха починає свій шлях зі стартового вузла і крок за кроком рухається по графу. На кожному кроці нею приймається рішення, куди рухатися, враховуючи феромони і евристичну інформацію, яка вказує на потенційну якість шляху.

3. Оновлення феромонів: Після того, як всі мурахи завершили свій шлях, феромони оновлюються на основі внесених змін. Зазвичай, феромони випаровуються з часом, а нові феромони додаються на пройдені ребра залежно від якості шляху.

4. Повторення: Описані вище кроки повторюються протягом кількох ітерацій або досягнення заданої умови зупинки. Поступово, за допомогою процесу відбору та оновлення феромонів, якість шляхів поліпшується, і мурашиний алгоритм знаходить наближений оптимальний шлях.

Мурашині алгоритми мають деякі переваги, такі як здатність працювати з графами великого розміру, адаптивність до змін у середовищі та здатність знаходити наближені оптимальні розв'язки. Однак вони також мають свої обмеження, такі як залежність від параметрів алгоритму та можливість застрягання в локальних оптимумах [11,12].

1.3 Порівняльний аналіз алгоритмів розв'язання логістичних завдань

Алгоритм A* є евристичним алгоритмом пошуку оптимального шляху. Він поєднує в собі пошук в ширину та евристичну оцінку, що дозволяє йому знаходити найкоротші шляхи в графі з вагами. Переваги алгоритму A* включають ефективність та можливість знаходження оптимального рішення. Він використовує евристичну функцію для приблизного оцінювання вартості досягнення кожної вершини, що дозволяє йому рухатися в напрямку найбільш обіцяючих шляхів[1, 2]. Однак, обмеження алгоритму A* полягають у високому використанні пам'яті, особливо для великих графів, і можливості потрапити в безкінечну петлю у графах з циклами.

Алгоритм пошуку в ширину є простим та ефективним алгоритмом, який дозволяє знаходити найкоротші шляхи в графі з вагами. Він починає з початкової вершини і просувається наступними вершинами на всіх можливих рівнях віддаленості від початкової вершини. Переваги алгоритму пошуку в ширину включають простоту реалізації та гарантоване знаходження найкоротших шляхів, якщо вони існують. Однак, він може бути неефективним для великих графів та графів зі швидким зростанням глибини.

Алгоритм пошуку в глибину є іншим простим алгоритмом, який дозволяє знаходити шляхи в графі з вагами. Він просувається глибше в граф, переходячи в наступну недосліджену вершину, доки не досягне кінцевої вершини або не стикається з тупиком. Переваги алгоритму пошуку в глибину полягають у простоті реалізації та малому використанні пам'яті. Однак, обмеження алгоритму включають можливість застрягання в тупиках та відсутність гарантії знаходження найкоротших шляхів.

Алгоритм Дейкстри є класичним алгоритмом для пошуку найкоротших шляхів в графі з неотрицательними вагами. Він просувається від початкової вершини до інших вершин, поступово оновлюючи найкоротші відомі шляхи до них. Переваги алгоритму Дейкстри включають ефективність та здатність знаходити найкоротші шляхи в графі зі змінними вагами[1]. Обмеження алгоритму полягають у вимозі неотрицательних ваг ребер та високому використанні пам'яті.

Алгоритм Дейкстри зі стисненими даними є модифікацією класичного алгоритму Дейкстри, яка оптимізує використання пам'яті. Він використовує структуру даних зі стисненими бітами для збереження інформації про найкоротші шляхи. Це дозволяє зменшити вимоги до пам'яті, але зберігає ефективність алгоритму Дейкстри [1]. Однак, обмеження включають більшу складність реалізації та можливість втрати точності через стиснення даних.

Алгоритм Беллмана-Форда є алгоритмом для пошуку найкоротших шляхів в графі з вагами, які можуть бути від'ємними. Він просувається по всіх ребрах графа та оновлює найкоротші відомі шляхи. Переваги алгоритму

Беллмана-Форда включають здатність працювати з вагами ребер будь-якого значення та знаходити від'ємні цикли[1]. Однак, обмеження включають більшу обчислювальну складність та повільнішу швидкість роботи для великих графів.

Алгоритм Джонсона є алгоритмом для пошуку найкоротших шляхів в графі з вагами, включаючи графи з від'ємними вагами. Він використовує модифікацію графа, щоб усунути від'ємні цикли та застосовує алгоритм Дейкстри для знаходження найкоротших шляхів [1]. Переваги алгоритму Джонсона включають здатність працювати з вагами будь-якого значення та виявлення від'ємних циклів. Однак, обмеження включають більшу обчислювальну складність і більший обсяг пам'яті.

Мурашиний алгоритм є метаевристичним алгоритмом, який імітує поведінку мурах для пошуку оптимального шляху. Він використовує ідею випаровування феромонів та вибору шляхів на основі ймовірностей. Переваги мурашиного алгоритму включають здатність знаходити наближені оптимальні рішення та здатність адаптуватися до змінних умов. Він також може працювати з графами великого розміру. Однак, обмеження включають випадковість результатів та потребу в налаштуванні параметрів алгоритму [2, 3].

Порівнюючи вищезгадані алгоритми, найгірше б у роботі повели б себе алгоритм пошуку в глибину та Джонсона. Перший зазначений неефективно визначає тупики та обходить велику кількість непотрібних вершин перед досягненням кінцевої вершини. Другий вимагає модифікації графа та використання алгоритму Дейкстри для кожної вершини, що може призвести до значного збільшення обчислювального часу та обсягу пам'яті. У випадку пошуку оптимального туристичного маршруту на гірській місцевості, де можуть бути великі графи з численними вершинами, це може бути неефективним підходом.

Враховуючи контекст даної роботи застосування мурашиних алгоритмів мають переваги, наведені нижче:

- Адаптивність: Мурашині алгоритми мають здатність адаптуватися

до змінних умов. Будуючи маршрут, можуть виникати непередбачувані фактори, такі як зміни погоди, недоступність шляхів або змінність відстаней та складностей, мурашиний алгоритм може знаходити прийнятні рішення, адаптуючись до змін.

- Здатність до знаходження наближених оптимальних рішень:

Основна ідея алгоритмів є випаровування феромонів та вибору шляхів на основі ймовірностей. Це дозволяє їм знаходити наближені оптимальні рішення, що можуть бути задовільними для багатьох практичних ситуацій.

- Підтримка врахування ваги шляхів: У контексті побудови туристичного маршруту можуть бути різні фактори, такі як відстань, висота, складність маршруту та інші. Мурашині алгоритми можуть бути модифіковані для врахування різних типів ваги шляхів та забезпечення пошуку оптимального маршруту, враховуючи ці фактори.
- Паралельне виконання: Мурашині алгоритми можуть бути легко розпаралелені, що дозволяє їм прискорити обчислення. Це особливо важливо для великих графів або ситуацій, де потрібно швидко знайти оптимальний маршрут.

Аналізуючи всі обрані алгоритми, більший потенціал для роботи з пошуком оптимального туристичного маршруту на гірській місцевості представляють саме мурашині алгоритми, враховуючи притаманні властивості. Однак, важливо враховувати ймовірні обмеження, такі як потреба в налаштуванні параметрів та можливість отримання наближених рішень з певною випадковістю.

РОЗДІЛ 2
ЛОГІСТИЧНА МОДЕЛЬ ТУРИСТИЧНОГО МАРШРУТУ НА ГІРСЬКІЙ МІСЦЕВОСТІ НА ОСНОВІ МУРАШИНИХ АЛГОРИТМІВ

2.1 Дослідження мурашиних алгоритмів. Використання мурашиних алгоритмів для вирішення оптимізаційних завдань

Основна ідея мурашиних алгоритмів полягає в імітації поведінки мурах для знаходження оптимальних рішень у складних задачах.

Однією з важливих концепцій є феромонна комунікація. Мурахи залишають феромонні сліди під час свого руху, і ці сліди використовуються для комунікації з іншими мурахами. Чим більша кількість феромону на шляху, тим ймовірніше, що інші мурахи оберуть цей шлях. Феромони служать як позитивний зворотний зв'язок, що допомагає мурахам зосереджуватись на більш обійнятому шляху та відкривати нові шляхи.

Подвійне відношення також є важливим принципом мурашиних алгоритмів. Мурахи мають здатність одночасно використовувати локальну та глобальну інформацію. Вони сприймають своє оточення, але також враховують інформацію, отриману від інших мурах. Це дозволяє мурашиним алгоритмам комбінувати індивідуальні знання з колективними знаннями для знаходження оптимальних рішень [4,5].

Самоорганізація є ще однією важливою властивістю мурашиних алгоритмів. Мурахи взаємодіють одна з одною і самоорганізуються відповідно до правил феромонної комунікації. Немає централізованого керівництва або глобального плану, але колективні дії мурах призводять до виникнення глобальної емерджентності - спонтанного формування групової поведінки, яка призводить до знаходження оптимальних рішень.

Адаптивність є ще однією значущою властивістю мурашиних алгоритмів. Вони можуть адаптуватися до змін у середовищі та вхідних умовах. Феромонні

сліди оновлюються на основі якості знайдених рішень, що дозволяє алгоритму адаптуватися та покращувати свою продуктивність протягом часу.

Використання мурашиних алгоритмів у багатьох оптимізаційних задачах пояснюється їхніми основними властивостями. Феромонна комунікація дозволяє ефективно обмінюватися інформацією та координувати дії агентів. Подвійне відношення допомагає збалансувати локальний та глобальний пошук, забезпечуючи знаходження оптимальних рішень. Самоорганізація дозволяє адаптуватися до змін у середовищі та знаходити нові шляхи. Адаптивність дозволяє підлаштовуватися до змінних умов і покращувати результати з часом.

Ці властивості роблять мурашині алгоритми ефективними у вирішенні складних оптимізаційних задач, включаючи побудову оптимального туристичного маршруту на гірській місцевості [5,6].

Феромонна система є ключовою складовою мурашиних алгоритмів і імітує комунікацію між мурахами за допомогою феромонів. Феромони - це хімічні речовини, які мурахи залишають на своєму шляху, при проходженні через ребра графа або вершини [7]. Ці феромони слугують як комунікаційний канал, за допомогою якого мурахи обмінюються інформацією про знайдені шляхи та розв’язки. Система дозволяє мурахам обмінюватись інформацією про якість шляхів та рішень, враховувати фактори евристики та адаптуватись до змін в середовищі, що призводить до знаходження оптимальних розв’язків в складних задачах.

Варто зазначити стигмергію, яка є основною властивістю мурашиних систем, що дозволяє мурахам взаємодіяти і співпрацювати один з одним без прямого обміну інформацією.

Загалом, концепція стигмергії у феромонних системах забезпечує децентралізовану та розподілену координацію між агентами, що дозволяє ефективно розв'язувати складні задачі оптимізації та пошуку.

У системах, таких як колонії мурах, стигмергія використовується як механізм комунікації та координації між окремими агентами. Основна ідея полягає в тому, що мурашки залишають слід (феромон) під час своїх рухів у

навколишньому середовищі. Цей феромон служить якісною оцінкою того, наскільки привабливим є певний шлях для інших мурашок.

Стигмергія в мурашиних системах базується на взаємодії між феромоном та мурахами. Коли мураха рухається по довільному шляху, вона залишає слід феромону. Інші мурахи, які згадають цей слід, мають більшу ймовірність обрати той же шлях. При цьому частота випаровування феромону залежить від часу, тобто старі феромонні сліди послаблюються і втрачають свою привабливість [10].

Ця форма комунікації дозволяє мурашиним колоніям знаходити та вибирати найкоротші шляхи до джерел їжі. Шляхи, по яких мурахи рухаються більш часто, будуть мати більшу кількість феромону і, отже, стануть більш привабливими для інших мурах. Поступово менш привабливі шляхи будуть залишені без феромону та припинять використовуватися [10, 12].

Головна мета мурашиних алгоритмів полягає в знаходженні найкращого рішення шляхом імітації поведінки мурашиної колонії та використання феромонних слідів для визначення оптимальних шляхів у просторі можливих рішень. Поведінка справжніх мурах у пошуку їжі базується на неявній оцінці рішення, тобто коротші шляхи будуть пройдені раніше, ніж довші, і, отже, вони отримуватимуть підсилення феромонами швидше. Натомість, штучні мурахи оцінюють рішення з урахуванням якоїсь міри якості, яка використовується для визначення сили підсилення феромонами, яке мурахи здійснюють під час повернення до мурашника. (рис. 2.1 - 2.5)[11].

Поміж позитивним зворотним зв'язком (відкладання феромону), необхідно також моделювати негативний зворотний зв'язок, а саме випаровування феромону. Негативний зворотній зв’язок забезпечує подальший пошук інших оптимальних шляхів [12, 13].

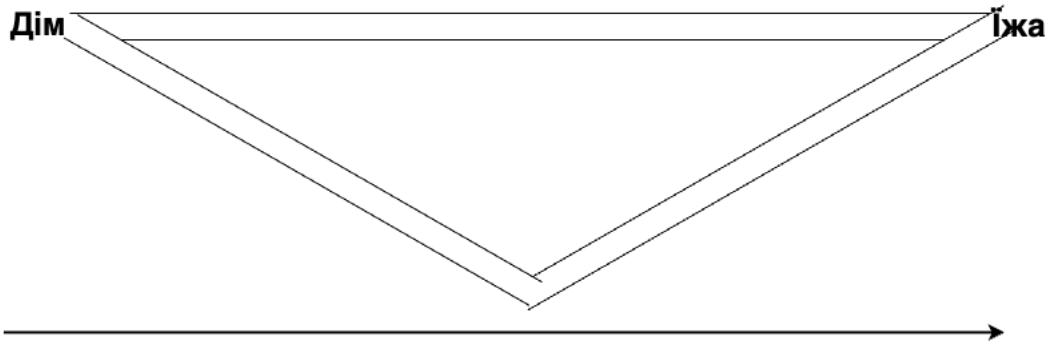

Рис. 2.1. Мурахи в гнізді

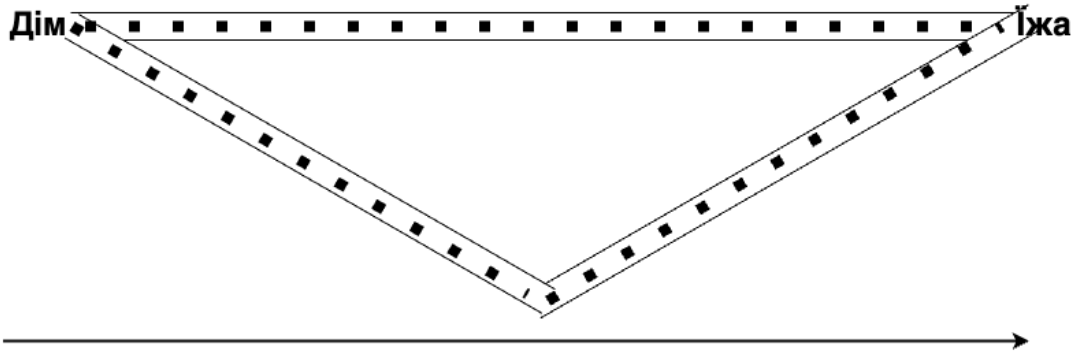

Рис. 2.2. Початок руху мурашиної колонії

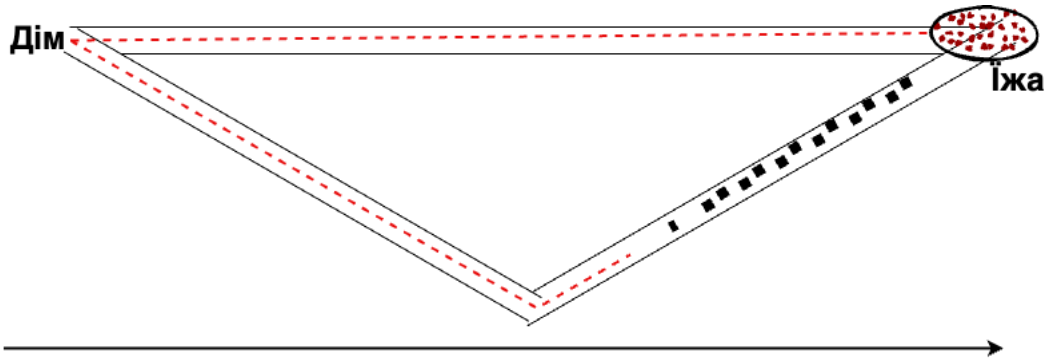

Рис. 2.3. Мурахи з коротшого шляху прийшли перші

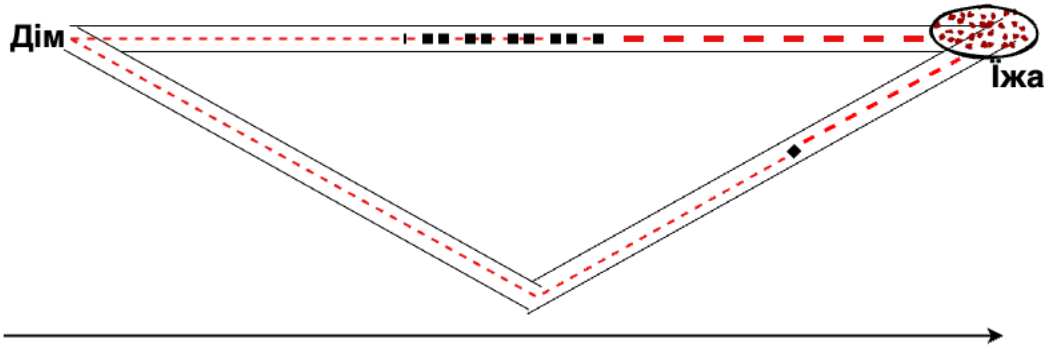

Рис. 2.4. Мурахи повертаються коротшим шляхом, виділивши більше феромонів

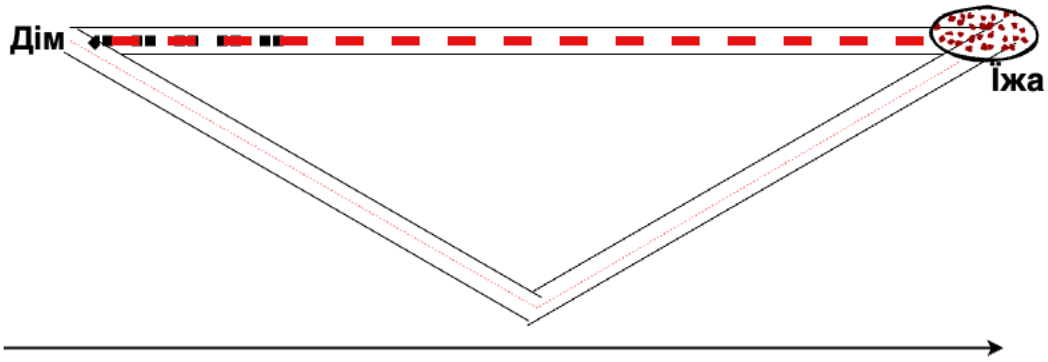

Рис. 2.5. Випаровування феромонів на довшому шляху

Реалізація алгоритмів може відрізнятися головним чином в таких аспектах: правила прийняття рішень, правила глобального оновлення і локального оновлення, які регулюють кількість феромону на різних шляхах.

Графова теорія визначає проблему як пошук циклу Гамільтона з найменшою вагою для заданого повного зваженого графа. Вона широко застосовується в інженерних додатках, а також деяких промислових проблемах, таких як планування машин, виробництво виробів та проблеми призначення частот. Повний зважений граф G = (N, E) може бути використаний для представлення задачі комівояжера, де N - множина n міст, а E - множина ребер (шляхів), які повністю з'єднують всі міста. Кожному ребру (i, j) ∈ E призначається вартість dij, яка представляє відстань між містами i та j. dij може бути визначений у просторі Евкліда і задається наступним чином:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (1)$$

Ймовірнісне правило переходу визначається:

$$p_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in J_k(i)} [\tau_{il}(t)]^\alpha [\eta_{il}]^\beta} & if\ j \in J_k(i) \\ 0 & otherwise \end{cases} \quad (2)$$

де nij - це евристична видимість ребра (i, j), зазвичай вона дорівнює значенню 1/dij;

dij - відстань між вузлом i та вузлом j;

m - кількість мурах;

Jk (i) - це множина вузлів, які залишилися для відвідування, коли мураха знаходиться в вузлі i;

Lk - довжина маршруту, пройденого мурахою k;

Q - довільна константа;

α та β - два налаштовувані позитивні параметри, які контролюють відносну вагу феромонної стежки та евристичної видимості:

Якщо α = 0, то закритий вершині I більш ймовірно буде обрано. Це відповідає класичному стохастичному жадному алгоритму.

Якщо навпаки β = 0, працює лише підсилення феромону: цей метод приведе систему до ситуації застою, тобто ситуації, в якій всі мурахи генерують підоптимальний маршрут.

Тому компроміс між довжиною ребра та інтенсивністю феромону є необхідним. Після завершення кожної мурахи свого маршруту кількість феромону на кожному шляху буде відкоригована згідно з рівнянням (1 - p), де p - параметр затухання феромону (0 < p < 1), який представляє випаровування сліду феромону, коли мураха обирає місто та вирішує перейти.

Мурашиний алгоритм пошуку шляху складається з наступних кроків:

1. Ініціалізація. Кількість феромону на кожному ребрі графа визначається малими сталими значеннями. Генерація початкової популяції мурах, розташованих у випадкових початкових точках.

2. Пошук рішення.

Використовується так зване псевдовипадкове пропорційне правило: ймовірність того, що мурашка перейде з вузла i у вузол j, залежить від випадкової змінної q, розподіленої рівномірно на інтервалі [0, 1], та попередньо заданого параметра q0.

$$j = \begin{cases} \arg\max_{u \in allowed_k(i)} \{ [\tau_{iu}]^{\alpha} . [\eta_{iu}]^{\beta} \} & \text{if } q < q_0 \\ J & \text{otherwise} \end{cases} \qquad (6)$$

J є випадковою змінною, визначеною відповідно до рівняння (2). Ця стратегія очевидно збільшує різноманітність пошуку, уникнення раннього потрапляння в локальний оптимальний розв'язок і застрягання.

3. Відновлення феромону.

Місцеве оновлення феромону виконується всіма мурашками після кожного кроку побудови. Кожна мурашка застосовує його лише до обраного вузла. Рівняння (2) використовується в основному для уникнення вибору дуже сильних шляхів феромону і для збільшення ймовірності дослідження інших шляхів. Коли край між вузлом і та вузлом j відвідали всі мурахи, правило місцевого оновлення зменшує рівень феромону на цьому краї. Таким чином, місцеве оновлення робить вже відвіданий край менш привабливим для наступної мурашки.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho.\tau_0 \qquad (7)$$

де ρ – параметр розпаду феромону $0 < \rho \leq 1$;

$\tau 0 = 1/n$ – початкові значення шляхів феромону;

Lnn – вартість, отримана за допомогою евристичного алгоритму найближчого сусіда;

n – кількість вузлів на графі;

4. Обчислення оптимального шляху. Після того, як m мурашок пройшли всі вузли, обчислюється довжина оптимального шляху.

5. Глобальне оновлення феромонів. Після того, як всі мурахи пройшли через всі міста, оновлюється лише кількість феромонів на оптимальному шляху за допомогою рівняння (8):

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho.\Delta\tau_{ij}(t) \qquad (8)$$

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{1}{L_{gb}}, \text{if } (i,j) \in \textit{global best tour} \\ 0 \qquad \text{otherwise} \end{cases} \qquad (9)$$

де ρ є постійним значенням;

Lgb – довжина найкращого глобального маршруту.

4. Додаткові дії

Схематичне представлення алгоритму пошуку шляху представлений на рис. 2.6. у вигляді блок схеми.

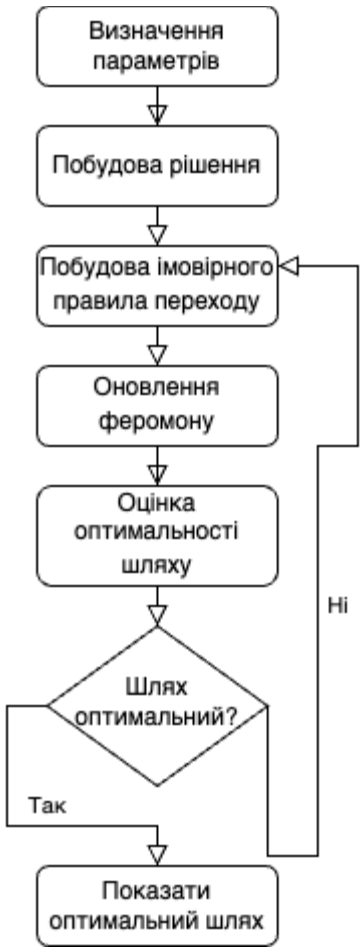

Рис. 2.6. Блок-схема алгоритму пошуку оптимального шляху мурашиною колонією

2.2 Оптимізація туристичного маршруту на гірській місцевості

Гірська місцевість має багато особливостей, які необхідно враховувати при побудові математичної моделі туристичного маршруту. Застосування мурашиних алгоритмів для розробки моделі дозволить врахувати особливості та знаходити оптимальні маршрути з урахуванням факторів, які впливають на переміщення у горах.

Для визначення математичної моделі і подальшої її оптимізації віднесені варто визначити такі етапи:

1. Визначити точки інтересу: Ідентифікувати основні точки інтересу на гірській місцевості, такі як визначні місця, перехрестя, позначені шляхи, природні пам'ятки, місця відпочинку і т.д. Кожна точка інтересу буде вузлом в графі.
2. Встановити зв'язки між точками інтересу: Встановити ребра (шляхи) між вузлами графу, що відповідають зв'язкам між точками інтересу. Варто врахувати відстань або вартість переміщення між кожною парою точок інтересу. Визначити вагу ребер: Призначити вагу кожному ребру графу, яке відповідає шляху між двома точками інтересу [15]. Вага може бути визначена зусиллями для подолання шляху.
3. Врахувати особливості гірської місцевості: Залежно від специфіки гірської місцевості, необхідно враховувати додаткові фактори, такі як висота, рельєф, складність шляхів, наявність джерел води або перешкод. Ці фактори можуть впливати на вартість або доступність певних шляхів та відображатися у вазі ребер.
4. Розробити математичну модель: На основі графу з визначеними вузлами, ребрами і вагами, розробіть математичну модель, яка відображає задачу пошуку оптимального туристичного маршруту. Це може бути задача мінімізації вартості при проходження між вузлами з урахуванням обмежень.

При побудові логістики туристичного маршруту на гірській місцевості потрібно отримати рішення, відповідне вимогам:

1. Маршрут ефективний: Мінімізація вартості проходу між точками.
2. Безпека туристів: Оцінка ризиків, врахування особливостей терену, встановлення правил безпеки та заходів для запобігання нещасним випадкам
3. Природне середовище стале: Мінімізація впливу туристичної діяльності на природне середовище гірської місцевості шляхом обмеження доступу до вразливих компонентів екосистеми.

4. Зручність: Наявність інфраструктури, такої як шляхи, сховища, місця відпочинку, інформаційні таблички та інші зручності, які сприяють задоволенню потреб туристів.

Візуалізація застосування мурашиного алгоритму для побудови шляху на гірській місцевості представлено на рис. 2.7, 2.8

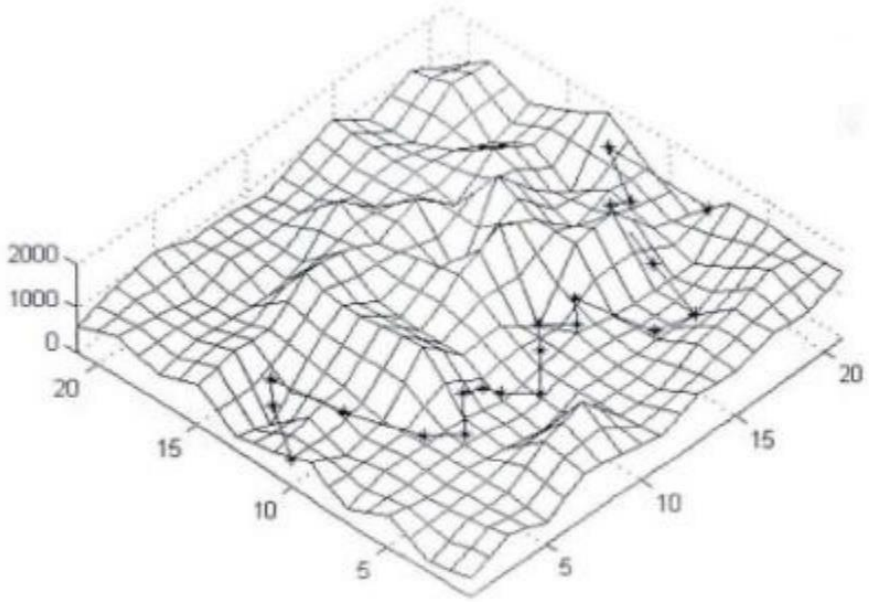

Рис. 2.7. Застосування мурашиного алгоритму у процесі пошуку шляху

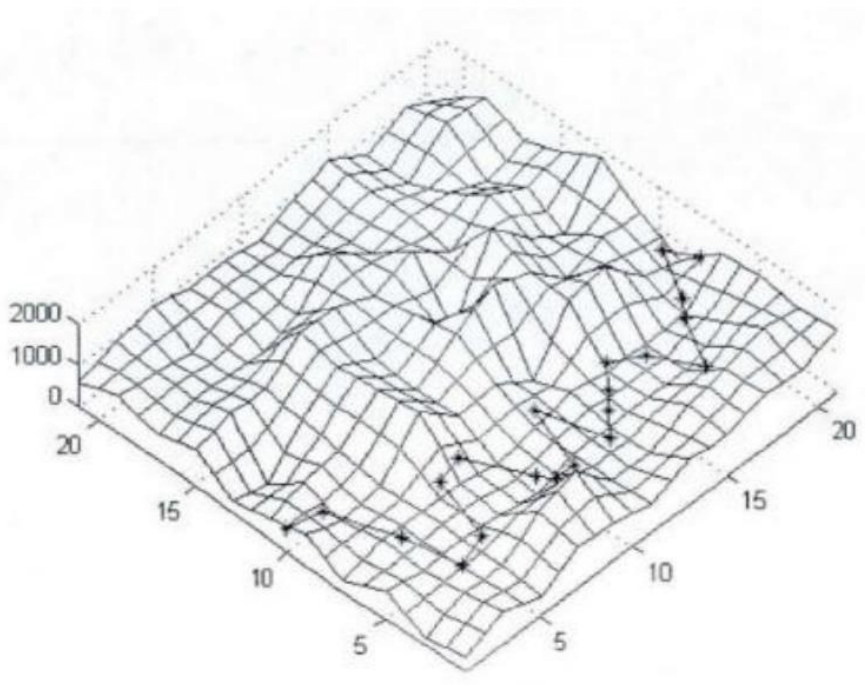

Рис. 2.8. Застосування оптимізованого мурашиного алгоритму у процесі пошуку шляху

Процес обчислення покращеного алгоритму тривимірного планування маршруту в даному дослідженні виглядає наступним чином:

1. Побудова моделі середовища, ініціалізація кожного параметра та введення базових даних. Встановлення лічильника оптимальних кроків N5 = 0. Встановлення Nr = 0, де Nr - кількість усіх знайдених мурахою шляхів після кожного глобального оновлення феромону. Встановлення допустимого списку allowed = 0; встановлення прапорця напрямку direct = 0; визначення положень початкової точки та цільової точки в моделі середовища; розміщення всіх мурах в початкових положеннях.

2. Для кожної мурахи k з поточним вузлом P(ia, ja, ka) як центром, вибрати та перейти до наступного вузла Pa+1(ia+1, ja+1, ka+1).

3. Виконання локального оновлення феромону.

4. Визначення, чи можуть всі мурахи знайти шлях до годувального місця вперше, якщо ні, повернутися до другого кроку. Початок повторення другого і третього кроків.

5. Глобальне оновлення феромону для визначення, чи знайшла мураха оптимальний шлях. Якщо відповідає вимогам, вивести оптимальний результат. Якщо ні, повернутися до другого кроку і повторити другий та третій кроки знову. Досягнувши всіх умов, алгоритм завершується.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка алгоритму роботи програмного забезпечення, що реалізує пошук оптимального маршруту урахуванням мурашиних алгоритмів

Процес проектування програмного забезпечення починається з моделювання, які реалізуються при оптимізації пошуку туристичного маршруту з імплементацією мурашиних алгоритмів. Для чіткого розуміння вимог і подальшого проектування складено блок-схему процесів програми. Блок-схема потрібна для визначення послідовності дій, які будуть реалізовані, і складається з різних блоків, які з'єднані стрілками.

Блоки на схемі мають різні форми і відрізняються по їх призначенню, Блоки можуть мати різні форми і символізувати різні дії або операції. Наприклад, прямокутник представляє операції обробки даних, ромб - рішення або умову, паралелограм - введення або виведення даних, та інші символи використовуються для конкретних дій.

Беручи до уваги результати проведеного аналізу та дослідження у другому розділі, розробимо блок-схеми роботи програмного забезпечення, що реалізує оптимізацію логістики з урахуванням мурашиних алгоритмів Такі блок-схеми мають враховувати роботу програми з урахуванням вибору маршрутів із різними ймовірностями, стратегію уникнення перешкод та різнорозмірності елементів складу див. **ДОДАТОК А**.

3.2 Проектування програмного забезпечення, що реалізує пошук оптимального маршруту урахуванням мурашиних алгоритмів

Задля глибшого розуміння логіки і зв’язків у розробляємому програмному забезпеченні скористаємося об’єктно-орієнтованим підходом та розробимо UML (Unified Modeling Language) діаграми. UML діаграми – графічне

відображення функціональності, компонент та алгоритмів. Вони несуть одразу декілька важливих ролей для розробки, такі як:

1. Візуалізація структури та взаємозв’язків
2. Опис функціональності та алгоритмів
3. Документування

3.2.1 Розробка діаграми прецедентів

Розроблена діаграми прецедентів спроектована з точки зору користувача програмного забезпечення. На ній відображений основний функціонал і можливості системи. Прецеденти описують взаємодію між користувачами і системою, зосереджуючись на тому, що система повинна зробити для задоволення потреб користувачів.

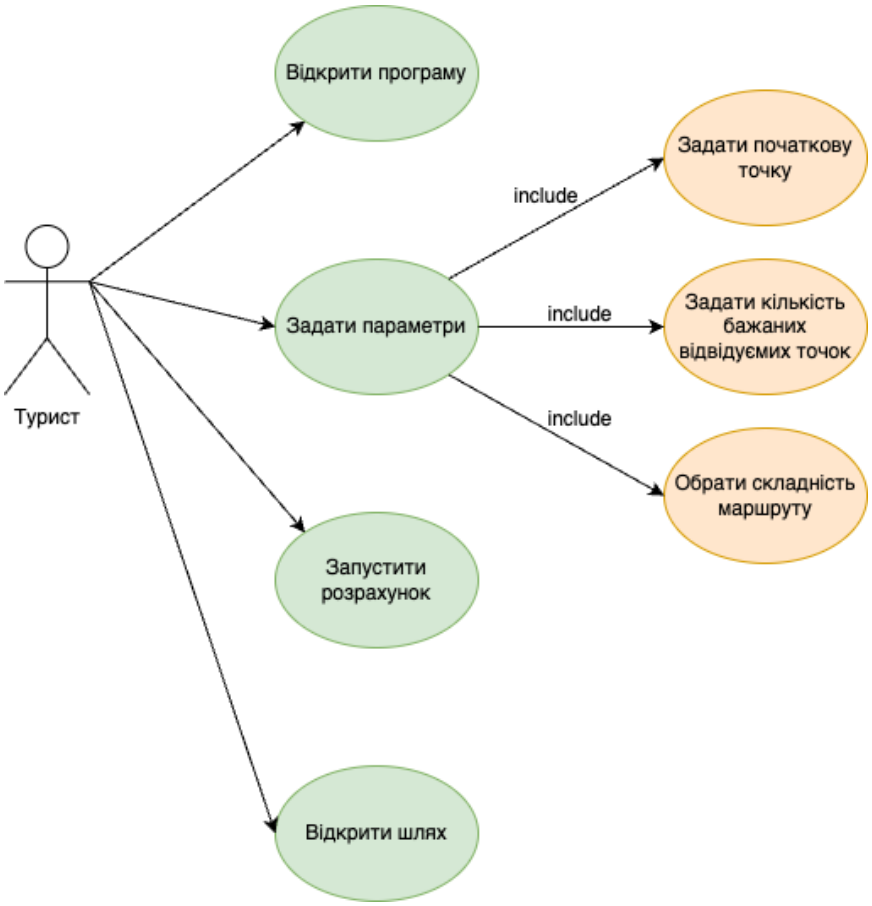

Рис. 3.1. Спроектована діаграма прецедентів для актора-туриста

3.2.2 Розробка діаграми класів

Діаграма класів візуалізує внутрішню структуру розробленої системи. Саме на етапі проектування класів аналізується і формується ґрунтовні взаємозв'язки компонентів, їх характеристики. Проте, діаграма є потужним інструментом на інших етапах, таких як аналіз, розробка та підтримка.

Рис. 3.2. Спроектована діаграма класів системи

На спроектованій діаграмі класів існують різні взаємозв'язки класів як: асоціація, агрегація та залежність. Зв'язок асоціації між класом основної програми і класів Mountain та ACO пояснюється безпосередньою підв'язкою до логіки програми: для конкретної гори та підлаштований до неї алгоритм.

Алгоритм залежить від вибору гори за рахунок обрахованих параметрів, перешкод, точок зупинок та іншого. Зв'язок агрегації між класом алгоритму і

класом Мураха наявний за рахунок того, що в основі алгоритму лежать об’єкти мурахи.

3.2.3 Діаграма послідовності

Діаграма послідовності дозволяє ілюструвати послідовність подій і взаємодію об'єктів в системі, допомагаючи уточнити поведінку системи та ідентифікувати можливі помилки чи оптимізаційні можливості.

Вона скаладається з наступних елементів:

- об’єкти – учасники взаємодії;
- повідомлення – відображення передачі інформації або запити між об’єктами;
- життєвий цикл – стан об’єкта та його переходи протягом послідовності дій;
- лінії життєвого циклу – тривалість існування об'єкта під час виконання послідовності.

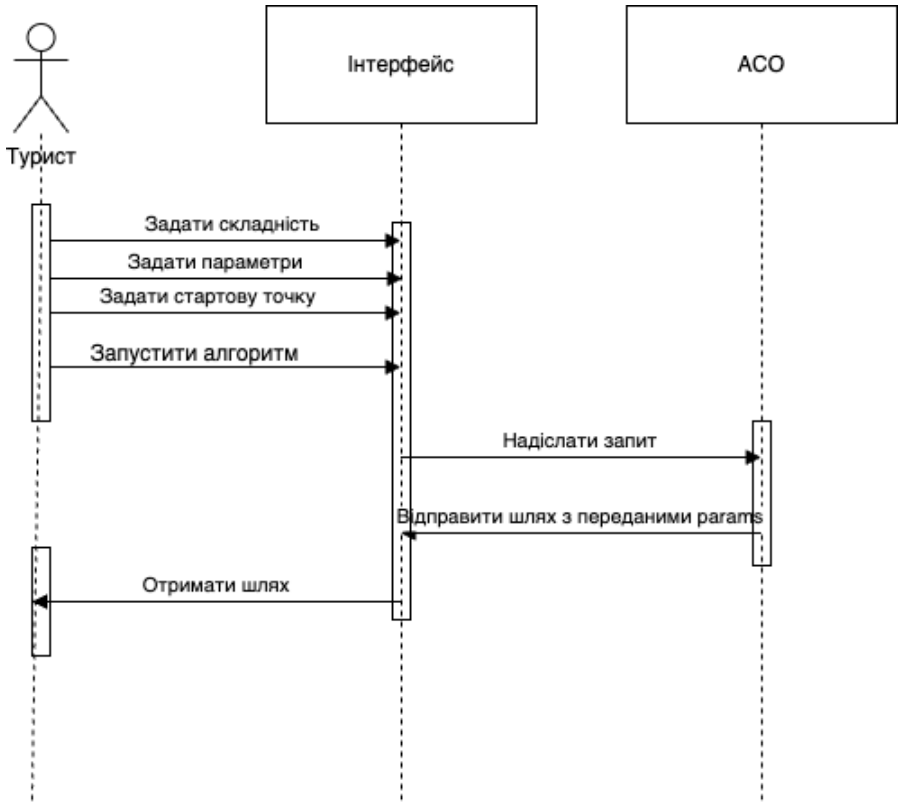

Рис. 3.3. Діаграма послідовності

3.3 Методи та засоби реалізації програмної розробки

Реалізація системи потребує засобів, які б давали змогу розробити весь перелічений функціонал. З урахуванням усіх потреб та задля зручності було вирішено скористатися популярним інтеграційним середовищем розробки IntelliJ IDEA 2022.3.2.

Вибір середи розробки IntelliJ IDEA забезпечує потужне та зручне середовище розробки для Java. Продукт включає в себе вбудований засіб для налагодження, візуальний конструктор інтерфейсу користувача, систему управління версіями, вбудований побудову та запуск проектів, а також зручний редактор, великий набір інструментів та підтримка розширень дозволив ефективно працювати над проектом і досягти бажаних результатів.

Мовою програмування була обрана Java17, яка наразі дуже популярна, має широкий спектр сфер застосувань, наявність і підтримку фреймворків та потужне ком’юніті. Java є повністю об'єктно-орієнтованою мовою. Вона підтримує основні принципи об'єктно-орієнтованого програмування, такі як спадкування, поліморфізм, інкапсуляція та абстракція. Інший критерій в прийнятті рішення про користування даною мовою – це наявність управління пам'яттю, що допомагає запобігти багатьом типам програмних помилок, таким як переповнення буфера або витік пам'яті.

3.4 Результати роботи та тестування ПЗ

Задача була спроектована на двовимірний випадок, який в подальшому можливо розгорнути на тривимірну реалізацію без вагомих змін у покращеному алгоритмі.

Принципи роботи:

1. Робота програми націлена на визначену єдину гірську місцевість завдяки унікальності параметрів для побудови математичної моделі гір.

2. На вхід подається кількість точок, початкова кількість феромонів, складність шляху (зусилля на проходження шляху) та матриця ваг, здійснена реалізація подана в додатку **Б**.
3. Далі запускається робота алгоритму і вираховує на кожній ітерації ініціалізацію параметрів, наявні наступні кроки по шляху та їх вибір, оновлення феромонів.
4. Останній крок роботи програми це вивід оптимального шляху та часу на його проходження.

Отже, маємо імітовану гору, яка для спрощення реалізації подана піксельно, де градієнтом визначається висота підйому (зелена та червона зони – це нижча та вища відповідно).

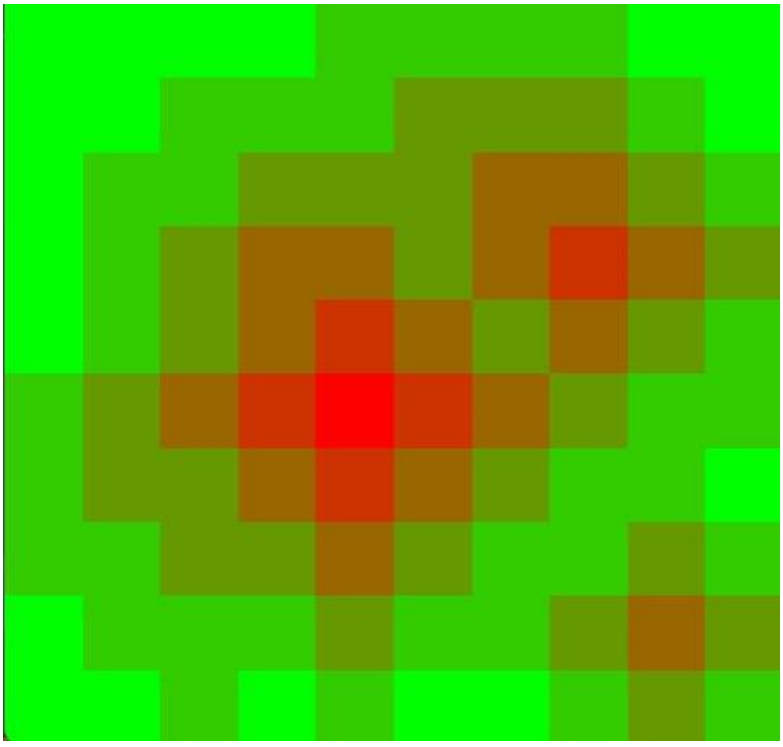

Рис. 3.4. Зображена структура гірської місцевості

Для змодельованої гірської місцевості визначимо перешкоди або певні особливості маршруту, такий як можливий напрям руху. Отримаємо орієнтований граф для задачі. Проекція даної дії на модель гори виглядатиме мапою рис. 3.5. Позначення трактуються відповідно:

- чорні позначки – неможливі для проходження перешкоди;
- блакитні – можливість руху тільки зверху-вниз і зліва-направо;
- рожеві – можливість руху лише низу-вверх та справа-наліво.

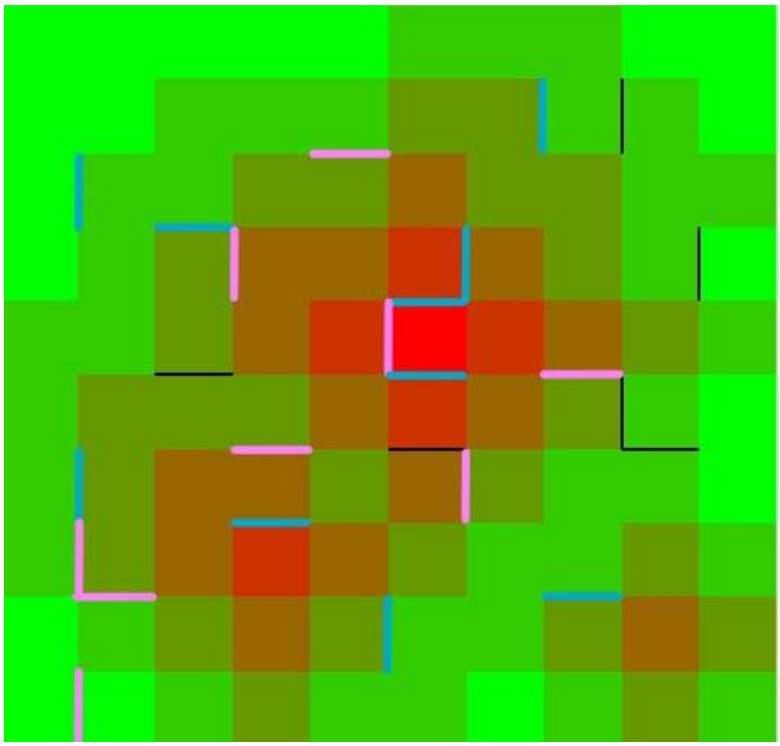

Рис. 3.5. Мапа гори з спроектованими напрямками руху та перешкодами з орієнтованого графа

Протестуємо реалізований алгоритм на випадку побудови оптимального шляху з найнижчої точки зображеної гори до її вершини. Оптимальний шлях знайдено і покладено на мапу змодельованої гірської місцевості на рис. 3.6. Білою лінією позначено оптимальний туристичний маршрут.

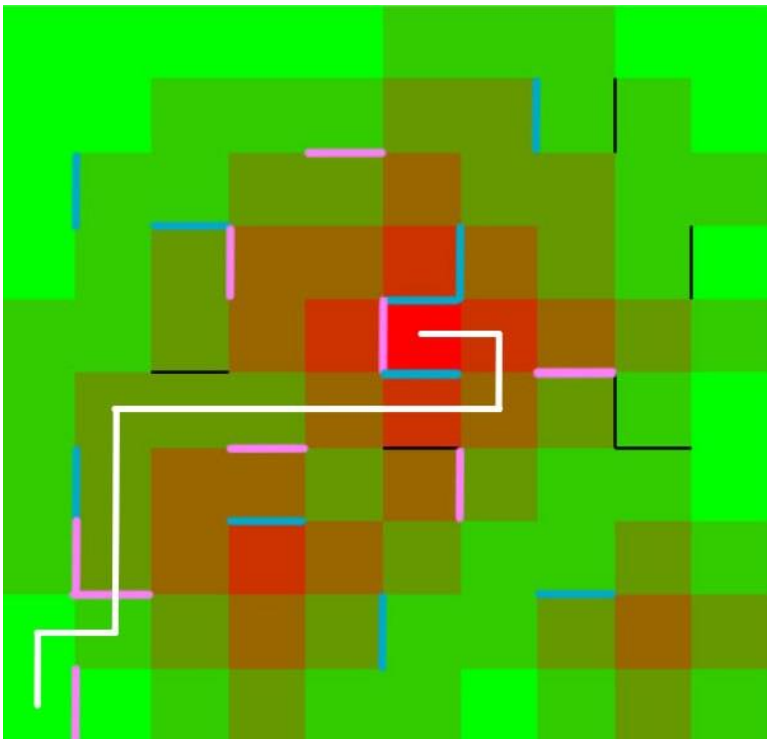

Рис. 3.6. Побудова оптимального шляху з найнижчої точки до вершини

Протестуємо оптимізований мурашиний алгоритм на випадку побудови оптимального туристичного шляху з вершини до найнижчої точки. Оптимальний шлях знайдено і накладено на мапу гори на рис. 3.7. Жовтою лінією позначений змодельований маршрут.

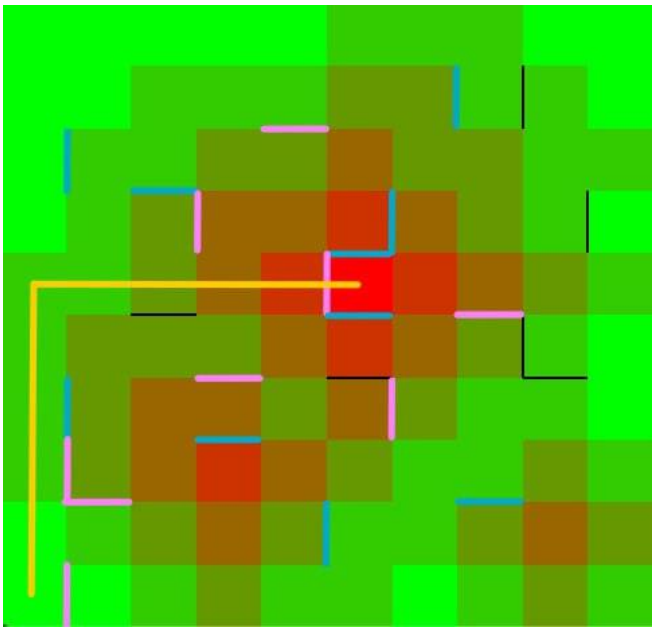

Рис. 3.7. Побудований оптимальний маршрут з точки вершини до найнижчої точки

В рамках роботи над пошуком оптимального туристичного шляху на гірській місцевості проведено порівняння ефективності роботи програмного забезпечення з застосуванням звичайного мурашиного алгоритму оптимізації і покращеного. Результати порівняльного аналізу показані в табл. 1

Таблиця 3.1. – Порівняння роботи покращеного та звичайного мурашиних алгоритмів

	Критерій порівняння	1	1	3	Середнє значення
Покращений	Значення оптимального шляху	43.188	39.860	37.081	10.043
	Запланований час	1.801	1.692	1.838	1.777
Класичний	Значення оптимального шляху	51.018	42.761	47.214	46.988
	Запланований час	9.506	4.048	6.184	6.579

Перевага алгоритму проявляється головним чином у значенні отриманих оптимальних результатів, ефективності роботи алгоритму та його практичності. Взявши для прикладу експеримент моделювання, можна побачити з таблиці 1, що вдосконалений алгоритм мурашиного колонії перевершує базовий алгоритм мурашиного колонії як з точки зору ефекту планування шляху, так і ефективності планування.

ВИСНОВКИ

Планування шляху та навчання є актуальними темами в світі. У цій роботі головним чином представлено оптимізований алгоритм мурашиного колонії у плануванні шляху для пошуку оптимального туристичного маршруту на гірській місцевості.

Була виконана наступна робота: представлено принцип планування оптимального шляху з урахуванням перешкод і створено математичну модель гірської місцевості. Для моделювання середовища використовується метод сітки, найбільш підходящий для тривимірного простору, і нарешті тривимірний простір розбивається на вузли у просторі. Знаходження оптимального шляху полягає в виборі набору мінімальних точок зі стартової точки до цільової точки серед цих вузлів, а вузли в наборі утворюють оптимальний шлях у тривимірному просторі. Поставлена мета роботи досягнута.

В роботі було використане програмне забезпечення Intellij IDEA та мова програмування Java для експериментальної реалізації покращеного алгоритму для тривимірного простору за допомогою проекції на двовимірний візуалізації планування маршрутів. Алгоритм мурашиної колонії був покращений шляхом зміни параметрів для підвищення ефективності планування маршруту туристів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стівен С. Скієн Посібник з проектування алгоритмів. М: Springer, 2004. 810 с.

2. Седжвік Р. Уейна К. Алгоритми. *PC WEEK/RE*. 2006. № 27. С.33-36.

3. Семенюта Е.В. Застосування мурашиних алгоритмів для пошуку оптимальних маршрутів вантажоперевезень. *Вісник Донецького національного технічного університету*, 1997. № 6. С. 8-14.

4. Blum, C., & Dorigo, M. The ant colony optimization metaheuristic: algorithms, applications, and advances. Swarm intelligence. 2005. 1(1), 1-6.

5. Dorigo, M., & Stützle, T. Ant colony optimization. MIT press, 2004. 319 с.

6. Gambardella, L. M., Dorigo, M., & Birattari, M. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1998. 1(1), 53-66.

7. Stützle, T., & Hoos, H. H. MAX-MIN ant system. Future generation computer systems, 2000. 16(8), с. 889-914.

8. Goss , S., & Aron, S. Distinguishing the role of forager and stegmoneur ant trail pheromones using a simple experimental design. Insectes Sociaux 1989, No. 36(2), С. 129-140.

9. С. Оптимізація траси автомобільної дороги на рельєфі з перешкодами методом імовірнісної дорожньої карти. *Вісник Сібаді*, 2012. № 2(28). С. 88-93.

10. Holland, O., & Melhuish, C. Stigmergy, self-organization, and sorting in collective robotics. Artificial life, 1999. No. 5(2), 173-202.

11. Bhattacharya P. Roadmap-based path planning- Using theVoronoi diagram for a clearance-based shortest path / P.Bhattacharya, M. L. Gavrilova // Robotics & Automation Magazine, IEEE. 2008. No. 15(2).–P. 58 – 66.

12. Pan P. Improved Ant Colony Algorithm for Path Planning of Soccer Robot. Journal of Convergence Information Technolog. 2013. No. 8(7). P. 958– 965.

13. Bang-Jensen, J., & Gutin, G. Digraphs: Theory, Algorithms and Applications (2nd ed.). Springer, 2008. C. 77-86.

14. H. Alazzam, O. AbuAlghanam, and A. Sharieh, Best path in mountain environment based on parallel A* algorithm and apache spark, J. Supercomput. 2021, c. 78, pp. 1–20.

15. Bernard Kolman, Robert E. Beck, Elementary Linear Programming with Applications (Second Edition), Academic Press, 1995. C. 48

ДОДАТОК А
Блок-схема пошуку оптимального туристичного маршруту

ДОДАТОК Б
Реалізація основного класу алгоритму ACO

```java
public class ACO {
    public static int numberOfAnts = 1000;
    public static int numberOfNodes = Mountain.TOTAL_NODES;
    public static int numberOfIterations = 5000;
    public static int mapSize = Mountain.HEIGHT_MAP.length;
    public static int endPoint;
    public static int startPoint;
    public static double evaporationRate = 0.5;
    public static double initialPheromone = 1;
    public static double elevationCoefficient = 0.8;
    public static double[][] pheromones;
    public static double[][] efforts;
    public static Ant[] ants;

    public static void initialize(int start, int end) {
        Mountain.initWeights();
        startPoint = start;
        endPoint = end;
        pheromones = new double[numberOfNodes][numberOfNodes];
        efforts = new double[numberOfNodes][numberOfNodes];
        ants = new Ant[numberOfAnts];

        for(int i = 0; i < numberOfNodes; i++) {
            for(int j = 0; j < numberOfNodes; j++) {
                pheromones[i][j] = initialPheromone;
                efforts[i][j] = Mountain.WEIGHT_MAP[i][j] + elevationCoefficient * Mountain.getElevationChange(i, j);
            }
        }

        for(int i = 0; i < numberOfAnts; i++) {
            ants[i] = new Ant(startPoint, totalEffort: 0);
        }
    }

    public static Integer[] getAvailableNextMoves(int antIndex) {
        List<Integer> availableNextMoves = new ArrayList<>();
        // find coordinates of current ant
        int currentPosition = ants[antIndex].currentPosition;

        int x = currentPosition / mapSize;
        int y = currentPosition % mapSize;

        if (x > 0) {
            availableNextMoves.add((x - 1) * mapSize + y);
        }
        if (x < mapSize - 1) {
            availableNextMoves.add((x + 1) * mapSize + y);
        }
        if (y > 0) {
            availableNextMoves.add(x * mapSize + y - 1);
        }
        if (y < mapSize - 1) {
            availableNextMoves.add(x * mapSize + y + 1);
        }

        return availableNextMoves.stream()
            .filter(move -> Mountain.WEIGHT_MAP[currentPosition][move] < Integer.MAX_VALUE)
            .filter(move -> !ants[antIndex].path.contains(move))
            .toArray(Integer[]::new);
    }
```

```java
public static int selectNextMove(int antIndex) {
    Integer[] availableNextMoves = getAvailableNextMoves(antIndex);

    if (ants[antIndex].currentPosition == endPoint || availableNextMoves.length == 0) {
        return -1;
    }

    double[] transitionProbabilities = new double[numberOfNodes];
    double total = 0;

    for (int i : availableNextMoves) {
        transitionProbabilities[i] = pheromones[ants[antIndex].currentPosition][i] / efforts[ants[antIndex].currentPosition][i];
        total += transitionProbabilities[i];
    }

    for (int i : availableNextMoves) {
        transitionProbabilities[i] /= total;
    }

    Random rand = new Random();
    double randomValue = rand.nextDouble();
    double cumulativeProbability = 0.0;

    for (int i : availableNextMoves) {
        cumulativeProbability += transitionProbabilities[i];
        if (randomValue <= cumulativeProbability) {
            return i;
        }
    }

    return -1;
}
```

```java
public static void updatePheromones() {
    for (int i = 0; i < numberOfNodes; i++) {
        for (int j = 0; j < numberOfNodes; j++) {
            pheromones[i][j] *= (1 - evaporationRate);
        }
    }

    for (Ant ant : ants) {
        for (int i = 0; i < numberOfNodes; i++) {
            pheromones[ant.currentPosition][i] += 1 / ant.totalEffort;
        }
    }
}
```

```java
public static void solve(int startPoint, int endPoint) {
    initialize(startPoint, endPoint);

    for (int iteration = 0; iteration < numberOfIterations; iteration++) {
        for (int antIndex = 0; antIndex < numberOfAnts; antIndex++) {
            int nextMove = selectNextMove(antIndex);
            if (nextMove == -1) {
                continue;
            }
            ants[antIndex].totalEffort += efforts[ants[antIndex].currentPosition][nextMove];
            ants[antIndex].currentPosition = nextMove;
            ants[antIndex].path.add(nextMove);
        }

        updatePheromones();
    }

    Ant bestAnt = Arrays.stream(ants)
        .filter(ant -> ant.currentPosition == endPoint)
        .min(Comparator.comparingDouble(ant -> ant.totalEffort))
        .orElse( other: null);

    if (bestAnt == null) {
        System.out.println("No path found");
        return;
    }

    List<Integer> bestPath = bestAnt.path;
    bestPath.add( index: 0, startPoint);

    System.out.println("Best path: " + bestPath);
    System.out.println("Total effort: " + bestAnt.totalEffort);
}
}
```