Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота бакалавра

з теми: «Багатопотокові методи формування маршрутів та їх оптимізація»

Виконав: здобувач вищої освіти групи KN1-B21
спеціальності 122 Комп’ютерні науки

Яремко Руслан Олегович

Керівник: Щирба Віктор Самуїлович,
кандидат фізико-математичних наук, доцент

Рецензент: Оптасюк Сергій Васильвич,
кандидат фізико-математичних наук, доцент

м. Кам’янець-Подільський – 2025 р.

Зміст

АНОТАЦІЯ

В кваліфікаційній роботі бакалавра на основі мурашиного алгоритму досліджено специфіку та перспективи використання паралельного програмування при обробці графових моделей.

Актуальність задачі обумовлена потребою використання високопродуктивних обчислювальних підходів при розв’язанні широкого кола прикладних задач від логістики в транспорті до телекомунікаційних мереж і систем штучного інтелекту.

Матеріал роботи знайде своє застосування і в навчальному процесі, зокрема, при вивченні освітніх коспонент «Алгоритми та структури даних», «Дискретна математика», в ряді вибіркових компонент, а також у практичній діяльності фахівців з комп’ютерних наук.

Ключові слова: паралельне програмування, теорія графів та її застосування, оптимізаційні алгоритми на графах.

ВСТУП

У сучасному світі питання ефективного формування маршрутів є надзвичайно актуальним у багатьох сферах — від логістики та транспорту до телекомунікаційних мереж і систем штучного інтелекту. Зі зростанням обсягів даних, складності маршрутних структур і вимог до швидкості обробки, виникає потреба у використанні високопродуктивних обчислювальних підходів. Одним із перспективних напрямів у цій галузі є застосування багатопотокових (паралельних) методів, які дозволяють значно зменшити час обчислень шляхом розподілу навантаження між кількома потоками.

Особливу увагу дослідників привертають біоінспіровані алгоритми, серед яких важливе місце займає мурашиний алгоритм — метаевристичний підхід, що ґрунтується на моделюванні поведінки колоній мурах у природі. Цей алгоритм демонструє високу ефективність при розв’язанні задач комівояжера, оптимізації мережних маршрутів, плануванні розподілу ресурсів та інших комбінаторних графових задач.

Поєднання мурашиного алгоритму з багатопотоковими технологіями відкриває нові можливості для підвищення ефективності обчислень, особливо у випадках, коли потрібно одночасно паралельно оцінювати велику кількість можливих маршрутів. Такий підхід дозволяє не лише прискорити процес знаходження оптимального шляху, але й забезпечити кращу масштабованість системи.

Об’єктом дослідження є процес формування оптимальних маршрутів у складних графових структурах.

Предметом дослідження є застосування багатопотокового мурашиного алгоритму для пошуку найкоротших шляхів та оптимізації маршрутизації.

Метою роботи є дослідження специфіки використання багатопотокових методів формування та оптимізації маршрутів мурашиний алгоритмом, а також аналіз впливу паралельного виконання на ефективність пошуку оптимальних рішень.

У роботі проаналізовано теоретичні основи алгоритму, розглянуто архітектуру багатопотокових систем, а також здійснено експериментальні дослідження з порівнянням продуктивності послідовної та паралельної реалізацій. Кваліфікаційна робота складається з анотації, вступу, трьох розділів, висновків, переліку використаних джерел та додатку, в якому розміщено код програмного продукту.

В першому розділі роботи проведено огляд літературних джерел з теми дослідження, розглянуто мотивацію та перспективи використання паралельного програмування при обробці графів, причини появи та засоби блокування конфліктних ситуації, що можливі при роботі паралельний потоків, специфіку використання пулу потоків.

В другому розділі розглянуто суть і принципи мурашиного алгоритму, особливості його використання при багатопотоковій обробці.

В третьому розділі описано практичну роботу з паралельними процедурами в задачі оптимізації методом мурашиного алгоритму.

На підставі проведених досліджень опубліковано дві роботи:

1. Віктор ЩИРБА, Руслан ЯРЕМКО Засоби блокування роботи паралельних потоків. Вісник Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 17. 2024, с 214-217.

2. Руслан ЯРЕМКО Багатопотокові методи формування маршрутів та їх оптимізація Збірник матеріалів наукової конференції за підсумками науково-

дослідної роботи здобувачів вищої освіти фізико-математичного факультету у 2024-2025 н. р. 9-10 квітня 2025 року. с. 109-112.

Виступав з доповіддю «Дослідження алгоритмів пошуку максимального потоку в потоковій мережі» на науковій конференції студентів і магістрантів університету за підсумками науково-дослідної роботи в 2024-2025 навчальному році.

Розроблені в роботі матеріали можуть впроваджуватися у навчальний процес фізико-математичному факультету Кам’янець-Подільського національного університету імені Івана Огієнка.

РОЗДІЛ 1 СПЕЦИФІКА ВИКОРИСТАННЯ ПАРАЛЕЛЬНИХ ПРОЦЕДУР ПРИ ДОСЛІДЖЕННІ ГРАФІВ

1.1. Огляд літературних джерел

В теоретичних та прикладних дослідженнях графи використовують для моделювання бінарних відношень між об'єктами. У цьому контексті графи розглядають як множини точок, що з'єднуються відрізками. Таких підхід до означення графів присутній в усіх підручниках та в навчально-методичних посібниках, що містять відомості про графи.

Також варто зазначити, що навчальний зміст розділів про графи в кожному з них досить ідентичний. Тому я вважаю, що достатньо посилатися на будь-яке із літературних джерел, і не став включати в список використаних джерел аналогічні матеріали, обмежившись лише підручником [7].

Про використання графів також є сила силенна літературних джерел, але не можна не відмітити, що окрім підручників, навчальних та навчально-методичних посібників багато уваги використанню графів приділено і у наукових статтях, наприклад, [9, 10, 11]. Тут я назвав лише частину тих наукових публікацій, що описують використання мурашиного алгоритму, дослідження якого міститься в моїй кваліфікаційній роботі.

Для більш широкого ознайомлення з питаннями про використання графів доцільно використати літературні джерела з дослідження операцій. Я вказав лише одне джерело [1]. Бакалаври спеціальності програмна інженерія глибоко знайомляться з такою освітньою компонентою.

«У рамках дисципліни *дослідження операцій* (ДО) вивчається велика кількість практичних задач, які зручно інтерпретувати як задачі оптимізації на графах. Прикладами таких задач є відшукання найкоротшого маршруту

між двома населеними пунктами …» [1]. Саме ця задача методом мурашиного алгоритму буде досліджуватися в моїй кваліфікаційній роботі.

Про мурашиний алгоритм також є чимало літературних джерел, серед яких я вибрав [2, 3, 4, 9, 10, 11]. Варто зазначити, що перші два джерела стосуються кваліфікаційних робіт магістра. Отже, здобувачі вищої освіти інших закладів освіти також звертають увагу на розв'язування оптимізаційних задач з графовими моделями за допомогою мурашиних алгоритмів.

Такого типу задачі поділяються на два типи: пошук ланцюгів та пошук циклів. Хоча ідея використання мурах в двох типах алгоритмів одна і та ж, але підходи побудови алгоритмів відрізняються. Першому типу алгоритмів присвячено роботи [2, 3, 4, 11], а другому – [9, 10].

Робота [9], як і моя кваліфікаційна робота, розглядає питання використання паралельного програмування при дослідженні графів, але в ній досліджується задача комівояжера, а я розглядаю задачу пошуку оптимальних ланцюгових маршрутів.

Питання організації паралельних обчислень, проблеми конфліктів та некоректної роботи потоків, синхронізації роботи, блокування тощо описуються в працях[5, 6, 8, 12].

1.2. Використання паралельного програмування при обробці графів

Паралельне програмування є однією з найважливіших та найбільш затребуваних технологій у сучасному світі комп'ютерних наук, особливо коли мова йде про обробку графів. Графи, які представляють собою множину вершин і ребер, широко використовуються у різних галузях, таких як аналіз соціальних мереж, оптимізація транспортних потоків, моделювання

біологічних систем, обробка великих обсягів даних та багато іншого. Саме завдяки паралельним алгоритмам стає можливим швидке та ефективне виконання складних обчислень, які інакше потребували б значних ресурсів і часу.

Сучасні графові алгоритми часто виконують обробку великих наборів даних, що ускладнює їхнє виконання на традиційних однопотокових системах. Наприклад, алгоритми пошуку шляхів, такі як алгоритм Дейкстри або алгоритм Флойда-Воршалла, можуть мати складність $O(n^2)$ або навіть $O(n^3)$, що робить їх непрактичними для обробки великих графів у реальному часі. Використання паралельного програмування дозволяє значно прискорити ці процеси шляхом розподілу обчислень між кількома процесорами або навіть між обчислювальними вузлами в кластерних системах.

Один з найбільш популярних підходів до паралельної обробки графів базується на використанні багатопотоковості та багатоядерних процесорів. Завдяки сучасним процесорам, які мають десятки і навіть сотні ядер, можливо розпаралелити виконання графових алгоритмів таким чином, що кожне ядро працює над окремою частиною графа або над незалежними операціями в межах одного алгоритму. Це дозволяє суттєво зменшити час виконання таких задач, як кластеризація графів, пошук мінімального кістякового дерева, аналіз центральності вершин тощо.

Крім того, сучасні обчислювальні архітектури, такі як графічні процесори (GPU), пропонують ще більший рівень паралелізму завдяки своїй здатності виконувати тисячі обчислень одночасно. Використання таких технологій, як CUDA (Compute Unified Device Architecture) або OpenCL, дозволяє реалізовувати паралельні алгоритми, які можуть значно пришвидшити обробку графів. Наприклад, алгоритми пошуку в глибину (DFS) та пошуку в ширину (BFS), які використовуються для аналізу

зв’язності графів або побудови дерев обходу, можуть бути реалізовані таким чином, що кожна вершина або кожне ребро обробляється окремим потоком.

Ще однією важливою концепцією у паралельній обробці графів є використання розподілених обчислень. У великих графових системах, наприклад у соціальних мережах або у веб-графах, обсяг даних може перевищувати можливості однієї машини. Тому застосовуються технології, такі як Apache Spark, Hadoop, GraphX та інші розподілені обчислювальні платформи, які дозволяють розподіляти графові обчислення між багатьма вузлами у кластері. Це забезпечує масштабованість та ефективність обробки графових структур навіть у випадках, коли дані не поміщаються в пам’ять одного обчислювального вузла.

Однак використання паралельного програмування для обробки графів пов’язане і з низкою викликів. Один з найголовніших викликів – це балансування навантаження між потоками або вузлами. У деяких графах, особливо у тих, що мають нерівномірний розподіл зв’язків, деякі вузли можуть мати набагато більше суміжних вершин, ніж інші. Це призводить до того, що одні обчислювальні вузли можуть бути перевантажені, тоді як інші простоюють. Для вирішення цієї проблеми використовуються спеціальні алгоритми розподілу навантаження, які рівномірно розподіляють обчислення між доступними ресурсами.

Ще одним важливим аспектом є синхронізація між потоками або процесами. У паралельних алгоритмах обробки графів часто виникає необхідність доступу до спільних структур даних, що може призвести до конфліктів при одночасному оновленні. Для цього використовуються різні механізми синхронізації, такі як блокування, атомарні операції або механізми узгодженого доступу, щоб уникнути умов гонки та забезпечити коректність обчислень.

Таким чином, використання паралельного програмування у сфері обробки графів є надзвичайно перспективним напрямом, який дозволяє суттєво підвищити ефективність виконання алгоритмів, зменшити час обчислень та забезпечити обробку великих обсягів даних. Завдяки сучасним технологіям, таким як багатоядерні процесори, графічні процесори та розподілені обчислення, дослідники та інженери мають у своєму розпорядженні широкий спектр інструментів для розв’язання найскладніших задач, пов’язаних з аналізом та обробкою графових структур. Незважаючи на певні виклики, розвиток паралельного програмування продовжує відкривати нові можливості для роботи з графами, що сприяє подальшому розвитку багатьох наукових та технічних галузей.

1.2.1. Мотивація використання паралельного програмування

Навіть із візуального аналізу зображення графу стає зрозумілим, що із збільшенням розмірності задачі (зростанням числа вершин та ребер) складність задачі пошуку оптимального маршруту значно підвищується. Тому дослідження ефективності оптимізаційних алгоритмів на графах, які працюють з великою кількістю даних, є актуальною, злободенною науковою проблемою сучасності.

Мурашиний алгоритм, на якому будуть зосереджені мої дослідження, є одним з ефективних відносно нових алгоритмів для розв’язання як задачі комівояжера, так і інших аналогічних задач пошуку маршрутів для прикладних задач, що моделюються на графах.

Як зазначено в [9, с. 106] «Алгоритм показує свою працездатність і хорошу надійність при достатній кількості ресурсів. Але необхідна кількість ресурсів та, відповідно, і час роботи швидко зростають зі збільшенням

розмірності задачі. Для вирішення цієї проблеми використовується розпаралелювання алгоритмів. Також воно застосовується для зниження передчасної збіжності до локального оптимуму, стимуляції різноманітності і пошуку альтернативних рішень тієї ж проблеми».

Про паралельне програмування, як технологію написання програм, що забезпечує виконувати декількох задач одночасно, досить часто почали з’являтися публікації з появою багатопроцесорних комп’ютерів.

Для запровадження паралельного програмування багатопроцесорні машини є достатньою, але не обов’язковою умовою. Широко використовується модель кластерних систем, коли декілька незалежних обчислювальних машин завдяки наявності мережі працюють як одна монолітна обчислювальна система. Більше того, комп’ютер з лише одним центральним процесором (уточняємо – одноядерним) фактично може одночасно обслуговувати декілька програм базуючись на мультипрограмному режиму роботи. Його суть полягає в тому, що всі працюючі потоки отримують центральний процесор у своє розпорядження лише на маленькі проміжки часу (кванти), а в інший час або виконують якийсь фрагмент програми, або перебувають на різних фазах стану готовності і очікування. Процесор по черзі обслуговує потреби всіх існуючих потоків і створює створює ілюзію паралельної роботи.

Поява багатоядерних процесорів і сучасних комп'ютерних систем спонукала до розробки спеціальних алгоритмів паралельного програмування та дослідженню нюансів їх використання. З однієї сторони, паралельна обробка даних дозволяє суттєво підвищити швидкість виконання програми, що особливо важливо в задачах з великою кількістю обчислювальних процедур чи даних, а з іншої розробка алгоритмів потребує спеціальних навиків, пов’язаних з синхронізацією роботи потоків.

Робота з потоками потребує особливої уваги через можливу появу, так званих, конфліктів, що зазвичай можуть виникати при одночасному використанні і можливій змінні спільних ресурсів, а також використанні потоком результатів роботи іншого потоку.

Для забезпечення коректної роботи різних потоків і гарантування відсутності помилок їх взаємодії використовують спеціальні методи блокування та синхронізації. Для організації ефективного управління роботою великої кількісті потоків досвідчені програмісти використовуються такий механізм як пул потоків**.**

1.2.2. Конфлікти, що можливі при роботі паралельних потоків

Найчастіше при роботі з потоками проблеми виникають через, так званні, конфлікти даних. Вони виникають тоді, коли два або декілька потоків одночасно намагаються змінити одні і ті ж самі ресурси, значення однієї і тієї ж змінної. Звичайно, це може призвести до викривлення результатів.

В моїй програмі при використанні методів паралельного програмування також можливий конфлікт даних. Це відбудеться тоді, коли один або навіть декілька загонів мурах одночасно попадають в одну і ту ж вершину і при цьому в переліку ще не заборонених вершин існуватимуть альтернативні або спільні для цих загонів елементи.

Для виправлення ситуації, точніше недопущення конфлікту, треба буде спочатку провести процедури «випаровування» існуючого об’єму відкладеного феромонів, далі провести відкладання по черзі кожним загоном нової партії феромонів і лише після цього вибирати маршрути слідування відповідно до процедур мурашиного алгоритму.

Іншим прикладом конфліктної ситуації може бути взаємне блокування.

Простим життєвим прикладом такої ситуації є випадок, коли, наприклад, один стакан наповнений молоком а інший чаєм і потрібно їй перелити з одного в інший. Такі ж ситуації виникають при переставлянні рядків чи стовпців у матриці, елементів у масиві тощо. Проблема легко вирішується використанням тимчасового місця зберігання.

Отже, взаємне блокування – це ситуація, що виникає при паралельному програмуванні, коли два або кілька потоків очікують один на одного для того, щоб звільнити свої ресурси і одержати ресурси іншого, але без цього жоден з них не може продовжити роботу. Це фактично призводить до аварійної зупинки виконання програми.

Подібна ситуація цілком реальна і моїй програмі з елементами паралельних процедур, якщо декілька загонів мурах різними шляхами з невеликими показниками феремонів потрапляють у спільну вершину, що містить альтернативні шляхи слідування і з якої ще не проглядається фінішна вершина.

Існує ще один випадок появи конфліктної ситуації, який в літературних джерелах називають гонками. Він виникає тоді, коли результат виконання багатопотокової програми залежить від порядку завершення роботи потоків. Якщо порядок виконання їх дій змінюється, то результати роботи програми вцілому можуть бути різним кожного разу.

В своїй програмі я не вбачаю появи такої ситуації. Тому не зупиняюся на цьому випадку

1.2.3. Засоби блокування не коректної роботи паралельних потоків

Для недопущення проблем, пов’язаних з появою конфліктних ситуацій, використовують спеціальні механізми блокування, які регулюють доступ до спільних ресурсів. Прикладами таких програмних засобів є м'ютекси, семафори, монітори та бар'єри.

М'ютекс дозволяє лише одному потоку отримувати доступ до певної ділянки коду або ресурсу. Всі інші потоки в цей час змушені чекати, поки м'ютекс не буде звільнений. Його можна порівняти із світлофором на пішохідному переході. Поки для пішоходів горить зелене світло, усі автомобілі не залежно від кількості їх потоків змушені чекати.

Семафор дозволяє певній кількості потоків одночасно отримати доступ до ресурсу. Його можна порівняти із блоком світлофорів із стрілками, які можуть дозволити одночасний рух по декількох смугах.

Монітор надає ексклюзивний доступ до ресурсу для одного потоку, поєднуючи в собі блокування і синхронізацію.

Бар'єр дозволяє цілій групі потоків синхронізуватися, чекаючи при цьому, поки всі потоки досягнуть певної точки перед тим, як продовжити виконання програми. Це корисно, коли потрібно, щоб усі потоки завершили певний етап роботи перед початком наступного.

1.2.4. Пул потоків

Одним із недоліків паралельного програмування є необхідність оголошувати і описувати код кожного потоку навіть, якщо вони виконують однакові процедури.

В мурашиному алгоритмі пропонується кожному загону мурах надати в розпорядження окремий потік. Тоді покрокові процедури в кожному потоці

будуть аналогічні з використанням своєї бази даних номерів вершин про пройдені саме цим загоном. Алгоритми відкладання феромону і процедури вибору оптимального маршруту будуть співпадати. Це спонукає до залучення в програму пулу потоків.

Пул потоків дозволяє формувати і керувати певною кількістю потоків без потреби створювати кожного разу нові потоки при виконанні аналогічних завдань. Зазвичай, пул потоків застосовують для оптимізації роботи з аналогічними задачами, що виконуються паралельно.

Механізм роботи пулу потоків можна поділити на три складові:

- Створюється певна (фіксована) кількість потоків. Пул потоків обслуговує паралельну роботу стільки потоків, скільки максимально можливо (за замовчуванням 25).
- Кожне нова задача ставиться у чергу та розв’язується одним з наявних потоків.
- Коли потік завершує розв’язання задачі, то він повертається в пул і готовий до розв’язання наступної задачі.

Пул потоків також тісно пов'язаний із методами блокування. Якщо декілька потоків хочуть отримати доступ до спільного ресурсу, то необхідно використовувати м'ютекси або інші засоби блокування.

Висновки до розділу 1

Прикладні задачі великої розмірності нереально вирішувати без використання сучасних високопродуктивних алгоритмів і комп’ютерних технологій. Одним із напрямків пошуку вирішення проблеми швидкого

розв’язання задачі є розвиток принципів паралельної та розподіленої обробки даних.

Переваги паралельного програмування роблять його важливим інструментом у розробці сучасного програмного забезпечення. Але, потрібно враховувати можливі складнощі, пов’язані в першу чергу з можливістю конфліктних ситуацій.

Запорукою успіху виступають засоби блокування потоків і пул потоків. Їх використання значно ускладнює процес програмування. В невеликих задачах, зокрема, демонстраційних програмах на графах, на це потрібно затратити додатково декілька днів, а виграш по часу роботи програми складатиме мільйоні долі секунди.

Тому в теоретичних дослідженнях, зокрема, в науковій статті [9], лише описують можливість застосування паралельного програмування. Я в своїй роботі також лише опишу послідовність дій з модифікації запропонованого алгоритму з без потокового у багатопотоковий варіант.

РОЗДІЛ 2 БАГАТОПОТОКОВІ МЕТОДИ ФОРМУВАННЯ МАРШРУТІВ ТА ЇХ ОПТИМІЗАЦІЯ

2.1 Основи багатопотокових методів в графових моделях

Відшукання оптимальних маршрутів є досить актуальною задачею для багатьох галузей, наприклад, логістики, транспортних перевезень та мережевих комунікацій. Стрімке зростання об’ємів даних та потреба оперативної їх обробки спонукають до використання багатопотокових методів ефективного відшукання найкращих розв’язків. Класичні методи відшукання оптимального шляху, такі як, наприклад, алгоритми Дейкстри чи Беллмана-Форда, ефективно працюють з статичними даними, але вони стають повільними при великих об’ємах допустимих маршрутів та змінних даних. Тому застосування паралельних розрахунків та еволюційних алгоритмів, таких як мурашиний алгоритм, є актуальним напрямком досліджень.

Природньо виникає потреба дослідити графові моделі багатопотоковими методами формування маршрутів, використовуючи переваги паралельного програмування, оцінити ефективність модифікованих алгоритмів та розглянути можливості аналізу ефективності пошуку оптимізаційних маршрутів шляхом використання паралельних розрахунків.

Багатопотокові алгоритми паралельних обчислень дозволяють значно прискорити процес відшукання оптимального розв’язку за рахунок перерозподілу процедур розв’язання підзадач між кількома процесорами кластерної системи або ядрами одного багатоядерного процесора.

В задачах пошуку оптимальних маршрутів використання принципів багатопотоковості надає можливість забезпечити ефективну обробку альтернативних варіантів маршрутів шляхом паралельного обчислення. Це, в свою чергу, значно зменшує час розв’язання загальної задачі.

Основними підходами досягнення такої мети при дослідженні графових моделей за допомого паралельності є:

1. Поділ задачі на підзадачі і розподіл підзадач між потоками. В цьому випадку кожен потік обробляє окрему, наперед задану, частину графа або набір декількох певний можливих шляхів.
2. Асинхронне (незалежне) виконання обчислень. Кожен потік може працювати незалежно від інших, що дозволяє не координувати роботу між ними і скоротити загальний час відшукання розв’язку.
3. Використання механізму синхронізації. Головна програма слідкує і координує роботу потоків при доступі їх до загальних ресурсів для недопущення можливих конфліктів.

Показовим прикладом модифікації алгоритму у варіант багатопотокової обробки може слугувати мурашиний алгоритм. Цей алгоритм є одним із найбільш ефективних підходів до розв’язання задачі пошуку циклів у задачі комівояжера і також пошуку оптимальних простих ланцюгових маршрутів у графах.

Ідея мурашиного алгоритму базується на імітації поведінки мурахи, яка шукає найкоротший маршрут між джерелом з їжею до мурашника.

Можна виділити чотири етапи цього алгоритму:

1. На першому етапі (**Ініціалізація**) в пам’ять заноситься в зручній формі початкова інформація про граф із зазначенням вагами ребер (дуг). Це може бути як вагова матриця, так і множина впорядкованих трійок, де

вказується стартова вершина ребра (дуги), кінцева вершина та вага цього ребра (дуги).

2. **Розпаралелення**. При розміщенні мурах за кожною з них закріплюється окремий потік (це просто і досить зручно в задачі комівояжера) або проводиться розбиття графа на частини, за якими закріплюється окремий потік.
3. **Обхід графа.** В мурашиному алгоритмі поведінка «мурахи» визначається рівнем феромону на кожному з ребер. Спочатку потрібно задати початкові рівні феромону на кожне ребро (може бути стала величина для всіх ребер або для прискорення роботи – обернено пропорційна до ваги ребра.
4. **Оновлення феромону.** На цьому етапі алгоритму необхідно передбачити збільшення кількості, підсилення рівня феромону на маршрутах, по яких пройшли мурахи, імітуючи відкладання феромону. Також зменшення на певний процент (коефіцієнт) наявність феромону, імітуючи цим самим випаровування феромону. Зміна кількості феромону змінює ймовірність вибору мурахами того чи іншого маршруту в наступних ітераціях.
5. **Конвергенція алгоритму або аналіз та вибір оптимального розв'язку.** Відомо, що однократне проходження маршрутами лише в окремих випадках надає оптимальний розв'язок. Потрібно проводити багатократні повторення, аж коли маршрути стабілізуються, обирається оптимальне рішення. Для підвищення ефективності мурашиного алгоритму застосовується багатопотокова обробка, яка дозволяє одночасно запускати кілька мурах та обробляти їх маршрути паралельно.

Оптимізація за допомогою паралельних обчислень.

Паралельні обчислення дозволяють значно прискорити виконання алгоритмів, особливо в задачах великої розмірності. Основні підходи до паралельної оптимізації:

- **Розпаралелювання мурах** – кожен потік обробляє певну групу мурах незалежно від інших.
- **Оптимізація оновлення феромону** – використання механізму блокувань або атомарних змінних для уникнення колізій під час оновлення феромону.
- **Динамічне балансування навантаження** – рівномірний розподіл обчислювального навантаження між потоками для максимального використання ресурсів процесора.

Порівняння ефективності. Аналіз ефективності багатопотокових методів роботи мурашиного алгоритму показав, що:

- Використання багатопотоковості зменшує час виконання алгоритму на 30-50% у порівнянні з однопотоковими версіями.
- Мурашиний алгоритм у паралельному виконанні знаходить оптимальний маршрут швидше на 40% завдяки одночасному обчисленню декількох варіантів шляхів.
- Оптимізація оновлення феромону дозволяє знизити конфлікти доступу до пам’яті та покращити продуктивність обчислень.

Висновки.

Використання багатопотокових методів для оптимізації маршрутів значно підвищує ефективність розрахунків та зменшує час виконання алгоритмів. Паралельне виконання мурашиного алгоритму дозволяє швидше знаходити оптимальні шляхи, зменшуючи навантаження на центральний процесор.

2.2 Основи мурашиного алгоритму

Мурашиний алгоритм є одним із метаевристичних методів оптимізації, який базується на колективній поведінці біологічних систем. Його ключова ідея полягає в імітації поведінки реальних мурах, які шукають найкоротші шляхи між гніздом (мурашником) та джерелом їжі. Мурахи, що пересуваються, залишають феромонний слід, який інші мурахи використовують для вибору найкращого маршруту. З плином часу інтенсивність феромону оновлюється, що дозволяє алгоритму адаптуватися до змінних умов та знаходити оптимальні рішення.

Основні етапи роботи мурашиного алгоритму:

1. **Ініціалізація** – задаються параметри алгоритму, створюється граф, де вершини представляють можливі стани, а ребра – зв’язки між ними.
2. **Розміщення мурах** – на початку ітерації кожна мураха випадковим чином обирає початкову позицію.
3. **Побудова рішень** – кожна мураха вибирає наступний крок згідно з імовірнісною моделлю, що враховує довжину шляху та рівень феромону.
4. **Оновлення феромонів** – після завершення проходу рівень феромону на маршрутах оновлюється, що сприяє підсиленню хороших рішень.
5. **Перевірка критерію завершення** – алгоритм повторюється до досягнення визначеної кількості ітерацій або стабільного рішення.

Проілюструємо роботу алгоритму на конкретному прикладі. Нехай схема рухів мурах задається графом, що зображено на рисунку 2.1. Довжини окремих ділянок нехай задано деякими числами, що позначено на рисунку величиною ваги ребер.

Будемо вважати, що частина мурах уже здійснила подорожі і відклала на маршрутах певну кількість феромону, наприклад, величинами обернено пропорційними довжині ребер.

Позначимо мурашник вершиною 0. Рухаючись з мурашника по ребру (0,1)до джерела їжі (їх може бути декілька, наприклад, у вершинах 7 і 8), мураха у вершині 1 натикається на перешкоду. Потрібно змоделювати вибір шляху і.д.

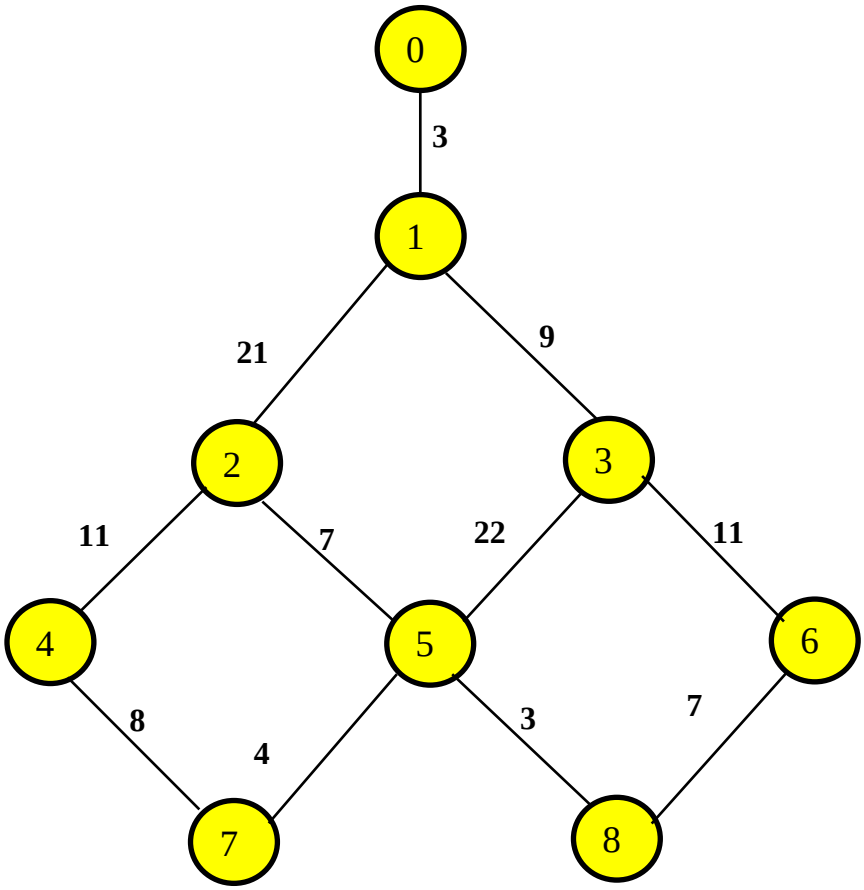


Рисунок 2.1. Граф подорожі мурахи.

Щоб програмно реалізувати мурашиний алгоритм, потрібно спочатку добре продумати етап ініціалізації алгоритму, задавши його керуючі параметри (наприклад, α – коефіцієнт, який задає відносну значущість шляху, тобто кількість феромона на ділянці, β – постійна величина, яка показує значення довжини шляху, ρ – коефіцієнт кількості феромона, який мураха залишатиме на шляху, Q – величина, яка відноситься до кількості феромона, що був залишений на маршруті і кількість ітерацій алгоритму тощо).

Крім цього, мураха не одинока в колонії, а рухається ціла популяція мурах, яка потрібно помістити в один або декілька пунктів мережі.

Важливим моментом є вибір методу задання графа: теоретико-множинний, матрицею суміжності, матрицею інцидентності тощо.

2.3 Принципи мурашиного алгоритму

Мурашиний алгоритм – це один із потужних та ефективних методів оптимізації, який заснований на природних принципах самоорганізації мурах. У природному середовищі мурахи, незважаючи на те, що є простими істотами без складної системи комунікації, здатні колективно знаходити оптимальні шляхи до джерела їжі. Вони роблять це завдяки своїй здатності залишати спеціальні хімічні речовини – феромони, які служать своєрідними мітками для інших особин колонії. Саме цей механізм і був покладений в основу алгоритму, що дозволяє вирішувати широкий спектр задач оптимізації.

Основна ідея мурашиного алгоритму полягає в децентралізованій взаємодії численних агентів, що діють на основі простих правил. Кожна

мураха виконує досить примітивні дії, однак, у сукупності з іншими агентами вони формують ефективний механізм пошуку найкоротших та найкращих маршрутів. Завдяки багаторазовому повторенню цього процесу система самостійно знаходить оптимальні рішення для заданої задачі.

Фундаментальним принципом функціонування алгоритму є механізм феромонів. Коли мураха знаходить певний маршрут, вона залишає на ньому слід у вигляді феромону. Чим більше мурах проходить цим маршрутом, тим більше накопичується феромону, що підвищує ймовірність того, що інші особини також оберуть цей шлях. Внаслідок цього поступово формується природний процес підсилення ефективних рішень та зменшення ролі менш оптимальних маршрутів.

Таким чином, можна виділити кілька ключових принципів, які визначають ефективність роботи мурашиного алгоритму:

1. Колективна поведінка

Один із головних принципів полягає у використанні великої кількості агентів, які працюють незалежно, але при цьому взаємодіють за допомогою феромонних слідів. Це дозволяє знайти оптимальні рішення шляхом випробування різних маршрутів та їх подальшої оцінки. Чим більше агентів бере участь у процесі, тим точнішим і більш ефективним буде кінцевий результат.

2. Імовірнісний вибір шляху

Кожна окрема мураха самостійно обирає свій наступний крок на основі певної ймовірності, яка залежить від концентрації феромону на різних маршрутах та від їхньої довжини. Це дозволяє реалізувати адаптивний механізм пошуку, що сприяє знаходженню найбільш вигідних рішень.

3. Підсилення хороших рішень

Оскільки маршрути, які ведуть до більш ефективних рішень, отримують більшу кількість феромону, вони мають більше шансів бути вибраними в майбутніх ітераціях. Це забезпечує природний процес покращення якості знайдених рішень, адже з кожним новим циклом алгоритм стає все більш точним та наближається до оптимального результату.

4. Ефект випаровування феромону

Щоб уникнути передчасного зосередження на локальних мінімумах, алгоритм передбачає механізм випаровування феромонів. Це означає, що з часом рівень феромону на маршрутах поступово зменшується. Завдяки цьому мурахи змушені шукати нові можливі варіанти рішень, а не лише дотримуватись одного вже знайденого маршруту. Таким чином, алгоритм не застрягає на одному шляху, а продовжує досліджувати альтернативні можливості, що підвищує його ефективність.

5. Паралельна обробка

Велика перевага мурашиного алгоритму полягає у тому, що кожен агент працює незалежно від інших. Це дозволяє реалізовувати алгоритм у багатопотокових середовищах, де завдання можуть виконуватись паралельно. Завдяки цьому алгоритм працює швидко та ефективно навіть при розв’язанні складних задач, що потребують значних обчислювальних ресурсів.

Мурашиний алгоритм є потужним інструментом для вирішення задач оптимізації, які зустрічаються в різних сферах науки та техніки. Він базується на природному механізмі пошуку найкращих рішень за допомогою взаємодії великої кількості агентів. Основні принципи алгоритму, такі як колективна поведінка, ймовірнісний вибір шляху, підсилення ефективних рішень, випаровування феромонів та паралельна обробка, забезпечують його високу продуктивність та точність. Завдяки цим характеристикам мурашиний

алгоритм успішно використовується у таких галузях, як транспортні системи, логістика, телекомунікації та навіть біоінформатика.

Таким чином, застосування мурашиного алгоритму дозволяє знаходити оптимальні рішення навіть у найскладніших задачах, використовуючи прості, але водночас дуже ефективні механізми, які підказані самою природою.

Разом з тим, не можна не відмітити недоліків мурашиного алгоритму, до яких, зокрема, належать:

– складність теоретичних висновків, оскільки завершальне рішення формується як результат побудови послідовності випадкових розв’язків;

– складність невизначення часу збіжності алгоритму, але, як не парадоксально, збіжність гарантується;

– висока складність побудови ітерацій;

– надмірна прив’язаність та залежність результатів алгоритму від стартових параметрів задачі;

– для ряду задач з великою кількістю вершин графа мурашиний алгоритм неефективний:

– є небезпека втратити оптимальний розв’язок через імовірнісне правило вибору наступного шляху;

– слабка збіжністю через приблизно однаковий вміст як кращих, так і гірших розв’язків при оновленні феромона;

– зберіганням в пам’яті неперспективних варіантів, що збільшує об’єм варіантів пошуку у задачах великої розмірності.

2.4 Суть мурашиного алгоритму

Мурашиний алгоритм (Ant Colony Optimization, ACO) був запропонований Марко Доріґо у 1992 році як засіб вирішення комбінаторних задач оптимізації, таких як задача комівояжера та проблема маршрутизації транспорту.

Основою алгоритму є процес самонавчання, який відбувається за рахунок використання феромонів, що залишають мурахи на своєму шляху.

Суть мурашиного алгоритму полягає в ітеративному пошуку оптимального маршруту шляхом взаємодії штучних агентів (мурах) із середовищем. Основний механізм алгоритму базується на наступних положеннях:

1. **Імітація природної поведінки мурах** – алгоритм використовує аналогію з природною поведінкою мурах, які шукають найкоротші маршрути між гніздом і джерелом їжі.
2. **Використання феромонів** – при русі мурахи залишають на маршруті феромонний слід. Чим кращий маршрут, тим більше феромону залишається, що приваблює інших мурах до цього шляху.
3. **Оновлення феромонного сліду** – феромон на маршрутах випаровується з часом, що дозволяє системі адаптуватися та уникати надмірного використання неефективних шляхів.
4. **Еволюційне навчання** – з кожною ітерацією алгоритм наближається до оптимального розв’язку, відкидаючи гірші маршрути та посилюючи кращі.

Перевагами мурашиного алгоритму є його здатність до швидкої адаптації в динамічних середовищах, можливість роботи з великими наборами даних та простота реалізації.

Однак основним недоліком є необхідність ретельного налаштування параметрів алгоритму та відносно високі обчислювальні витрати у задачах великої розмірності.

У майбутньому очікується розвиток гібридних методів, що поєднують мурашині алгоритми з іншими підходами, такими як нейронні мережі та генетичні алгоритми. Такі методи можуть значно підвищити ефективність оптимізаційних процесів у складних системах.

2.5 Мурашиний алгоритм у багатопотоковій обробці

Алгоритм є одним із найбільш цікавих та перспективних методів вирішення складних оптимізаційних завдань. В основі цього підходу лежить спостереження за поведінкою мурах у природі, їхньою здатністю знаходити найкоротші маршрути між їжею та мурашником. Враховуючи те, що мурахи комунікують через феромони, залишаючи сліди на своїх маршрутах, можна моделювати цей процес у програмному забезпеченні та використовувати його для ефективного розв’язання задач оптимізації.

Мурашиний алгоритм давно використовується в різних сферах науки та техніки. Його основний принцип ґрунтується на тому, що безліч агентів (мурах) досліджують середовище, визначаючи оптимальні шляхи за допомогою ймовірнісного вибору напрямку руху. З часом маршрути, які є найбільш перспективними, підкріплюються, оскільки вони отримують більше феромонів, тоді як менш ефективні поступово зникають через випаровування феромону. Ця еволюційна поведінка дозволяє алгоритму ефективно знаходити рішення задач комівояжера, маршрутизації транспорту, розподілу

ресурсів, кластеризації та навіть у медицині, наприклад, для аналізу зображень або діагностичних завдань.

Багатопотокова обробка є важливим аспектом сучасних обчислень, що дозволяє значно пришвидшити виконання складних алгоритмів. Впровадження багатопотокової обробки у мурашиний алгоритм відкриває нові можливості для оптимізації процесів, дозволяючи паралельно обробляти маршрути мурах та прискорювати оновлення феромонних слідів. У класичному підході кожна мураха незалежно обирає свій маршрут, оцінює його якість і оновлює рівень феромонів на пройденому шляху. Проте, якщо використовувати паралельні обчислення, можна розподілити мурах між кількома потоками, що дає змогу одночасно досліджувати декілька можливих рішень, не витрачаючи зайвого часу на послідовне виконання операцій. Це дозволяє зменшити час обчислень та підвищити ефективність алгоритму.

Багатопотоковий мурашиний алгоритм реалізується різними способами, залежно від особливостей платформи та завдання. Найпоширенішим варіантом є розподіл мурах між окремими потоками, де кожен потік займається побудовою маршрутів для певної групи агентів. Це дозволяє рівномірно завантажити процесор і значно скоротити час виконання алгоритму. Іншим підходом є паралельне оновлення феромонної карти, коли після побудови маршрутів декілька потоків одночасно коригують рівень феромонів у різних ділянках пошукового простору. Таким чином, можна уникнути затримок, пов’язаних із блокуванням спільних ресурсів, і підвищити продуктивність.

Ще одним важливим аспектом багатопотокової обробки у мурашиному алгоритмі є використання сучасних технологій, таких як графічні процесори (GPU), які дозволяють розпаралелити обчислення ще більше. GPU-реалізація мурашиного алгоритму може значно перевищувати за швидкістю традиційні

CPU-варіанти, оскільки вона дає змогу одночасно обчислювати тисячі незалежних операцій. Це особливо корисно для великих оптимізаційних задач, таких як логістика, проектування складних мереж або симуляції фізичних процесів.

Окрім технічних переваг, мурашиний алгоритм у багатопотоковій обробці також має велике значення для практичного застосування в різних галузях. Наприклад, у транспортній індустрії він допомагає розробляти оптимальні маршрути для перевезення вантажів, зменшуючи витрати на пальне та покращуючи час доставки. У сфері телекомунікацій мурашині алгоритми використовуються для оптимізації навантаження на мережі, що дозволяє покращити якість зв’язку та знизити затримки в передачі даних. В медицині вони можуть застосовуватися для розпізнавання образів у діагностичних системах, дозволяючи швидше та точніше ідентифікувати патології.

Таким чином, мурашиний алгоритм у поєднанні з багатопотоковою обробкою є потужним інструментом для вирішення широкого спектра завдань. Його здатність швидко адаптуватися до змінних умов, ефективно досліджувати великі простори рішень і знаходити оптимальні маршрути робить його одним із найбільш перспективних підходів у сучасній оптимізації. Враховуючи постійний розвиток апаратного забезпечення та методів паралельних обчислень, можна очікувати, що застосування цього алгоритму буде лише розширюватися, відкриваючи нові горизонти для науки, техніки та бізнесу.

Висновки до розділу 2

Мурашиний алгоритм — це один із потужних та ефективних методів оптимізації, який заснований на природних принципах самоорганізації мурах.

У природному середовищі мурахи, незважаючи на те, що є простими істотами без складної системи комунікації, здатні колективно знаходити оптимальні шляхи до джерела їжі. Вони роблять це завдяки своїй здатності залишати спеціальні хімічні речовини — феромони, які служать своєрідними мітками для інших особин колонії. Саме цей механізм і був покладений в основу алгоритму.

Основні принципи роботи мурашиного алгоритму полягають у наступному:

1. **Феромонний слід**. Кожна мураха, проходячи ребро графа, залишає за собою певну кількість феромону. Інші мурахи, рухаючись графом, мають більшу ймовірність обрати ребро з більшою концентрацією феромону. Таким чином, більш короткі та оптимальні маршрути, що частіше проходяться мурахами, отримують посилений феромонний слід.
2. **Імовірнісний вибір маршруту**. Вибір наступної вершини відбувається не детерміновано, а з урахуванням ймовірностей, що залежать від концентрації феромону і ваги ребра (зазвичай довжини або вартості). Цей механізм дозволяє уникнути застрягання у локальних мінімумах.
3. **Випаровування феромону**. З часом феромон на ребрах зменшується (випаровується), що дає можливість алгоритму адаптуватися до змін у графі та уникати застарілих рішень.
4. **Колективна поведінка**. Популяція мурах працює разом, одночасно досліджуючи різні ділянки графа. Завдяки взаємодії через феромони, колонія мурах поступово конвергує до найкращого маршруту.

5. **Паралелізм і розподіленість**. Оскільки мурахи діють автономно, їх рухи та обчислення можна розпаралелювати, що значно підвищує ефективність алгоритму при роботі з великими графами.

Таким чином, мурашиний алгоритм реалізує принципи самонавчання і самоорганізації, ефективно шукаючи оптимальні маршрути в складних мережах.

РОЗДІЛ 3 ПРАКТИЧНА РОБОТА З ПАРАЛЕЛЬНИМИ ПРОЦЕДУРАМИ В ЗАДАЧІ ОПТИМІЗАЦІЇ МЕТОДОМ МУРАШИНОГО АЛГОРИТМУ

Реалізація оптимізаційних методів у сучасних обчислювальних системах вимагає глибокого розуміння як самих алгоритмічних підходів, так і принципів їх ефективного впровадження. Особливої уваги заслуговують методи, натхненні біологічними процесами — серед яких мурашиний алгоритм посідає одне з провідних місць. Цей алгоритм, що імітує колективну поведінку колонії мурах у пошуку їжі, став основою для численних практичних застосувань у задачах маршрутизації, логістики та пошуку найкоротших шляхів у складних графах.

У цьому розділі увагу зосереджено на практичній реалізації задачі оптимізації маршруту з використанням методу мурашиного алгоритму в умовах паралельних обчислень. Виконання цього завдання вимагає комплексного підходу: від формалізації математичної моделі до створення алгоритмічної структури, що ефективно розподіляє навантаження між потоками. Так само, як вода, що прокладає собі шлях серед каміння, паралельне виконання дозволяє знайти найефективніший маршрут навіть у найскладнішому середовищі.

Практичне завдання цього розділу полягало не лише в реалізації пошуку оптимального шляху, а й у побудові динамічної моделі, що відображає процес еволюції маршруту під впливом феромонів і змін навколишнього середовища. Завдяки цьому вдалося отримати повноцінну систему, що демонструє переваги самонавчання та адаптивності, властиві мурашиному алгоритму.

3.1 Постановка задачі

Першим кроком у вирішенні задачі оптимізації стало чітке формулювання умов, обмежень та цілей, які має реалізовувати модель. Як і в будь-якій гідродинамічній системі, де визначення напрямку течії залежить від рельєфу та сили тяжіння, в задачі маршрутизації необхідно задати граф з вузлами та ребрами, де кожне ребро має певну вагу — найчастіше це відстань, час, або вартість переміщення між точками.

Задача полягала у знаходженні найкоротшого шляху між стартовою точкою (джерелом) і кінцевою точкою (їжею) в заданому графі, використовуючи мурашиний алгоритм із можливістю паралельного виконання. При цьому важливо забезпечити дотримання таких умов:

- кожна мураха стартує з однієї і тієї ж вершини;
- усі мурахи діють незалежно одна від одної, але оновлюють загальний феромонний слід;
- враховуються часові обмеження, кількість мурах, параметри феромонного випаровування та підсилення.

Цей підхід дозволяє моделювати реальні задачі, пов’язані з плануванням маршрутів транспорту, дротових мереж або логістичних ланцюгів. Постановка задачі є тим джерелом, з якого витікає весь подальший потік розв’язання, що поступово розгалужується в окремі процедури реалізації.

3.2 Реалізація мурашиного алгоритму з урахуванням паралельних обчислень

Після формального опису задачі наступним кроком стало впровадження мурашиного алгоритму в програмну систему з підтримкою паралельної обробки. Це дозволило не лише скоротити час обчислень, а й адаптувати алгоритм до умов реального часу, що має критичне значення для промислових і транспортних задач.

Суть мурашиного алгоритму полягає в тому, що кожна мураха при виборі наступної вершини використовує ймовірнісну формулу, яка враховує кількість феромонів на ребрі та евристичну оцінку привабливості шляху. Саме цю формулу, як показано нижче, ми використали у нашій реалізації, де:

— кількість феромонів на ребрі ;

— евристична привабливість ребра (наприклад, зворотна відстань);

— коефіцієнти впливу феромону та евристики відповідно.

Важливою інженерною задачею стало впровадження багатопотокового середовища, в якому кожна мураха виконується в окремому потоці або групі потоків. Це, у свою чергу, вимагає синхронізації при оновленні феромонних слідів, щоб уникнути конфліктів. Застосування паралелізму дало змогу пришвидшити обробку великих графів, а також дозволило відтворити реалістичну картину поведінки мурах.

Як у складній екосистемі, де всі елементи взаємодіють гармонійно, так і в паралельній моделі необхідно досягти балансу між швидкістю виконання, стабільністю оновлення та точністю маршрутизації. Саме це і було предметом детального розгляду в наступних підрозділах.

3.2.1 Моделювання графа та початкових умов

Створення віртуального середовища, в якому діятиме алгоритм, є ключовим кроком, аналогічним до формування русла для водного потоку.

Граф моделюється у вигляді множини вершин та зв'язків між ними (ребер), де кожне ребро має свою вагу. Під час початкового налаштування було обрано фіксовану кількість вершин (у нашому випадку — 14), серед яких одна є початковою (умовна точка старту), а одна — ціллю (їжа).

Візуалізація графа дозволила точно спроектувати структуру шляху: віддаленість вузлів, варіанти маршруту, наявність тупиків та альтернативних шляхів. Ця схема стала не лише базою для алгоритму, а й важливим засобом сприйняття динаміки руху мурах у процесі обчислення. Кожна вершина позначалася унікальним номером, і всі мурахи стартували одночасно з єдиної початкової точки.

Додатково були задані початкові рівні феромонів, параметри їх випаровування та коефіцієнти впливу, що дозволило зробити алгоритм адаптивним до змін середовища. Усе це формує стабільну основу, яка згодом буде динамічно змінюватися під впливом колективної поведінки агентів.

3.2.2 Пошук шляху та візуалізація руху мурах

На цьому етапі розпочався найдинамічніший процес — безпосередній рух мурах по графу. Кожна мураха самостійно вибирає маршрут, орієнтуючись на поточний рівень феромонів та відстані до суміжних вершин. На практиці це реалізовано у вигляді анімованого руху між точками, де кожен крок відображається на графічній формі у реальному часі.

Мурахи рухаються поступово, залишаючи за собою феромонний слід, який згодом впливає на вибір наступних мурах. З кожною ітерацією система адаптується і найкоротший маршрут набуває дедалі більшої концентрації феромонів. Візуально це виглядає як поступове виділення найефективнішого шляху з-поміж багатьох можливих. Подібно до річки, яка після тривалого часу прокладає собі русло через тверді породи, алгоритм знаходить найкращу траєкторію серед безлічі варіантів.

Особливу увагу було приділено ефекту повернення: після досягнення їжі мураха має повернутися тим самим маршрутом. Цей аспект реалізовано з урахуванням накопиченого феромону, що дозволяє мурахам повторно використовувати найефективніші шляхи, уникаючи зайвих витрат часу та ресурсів.

Паралельно з анімацією система фіксує шлях кожної мурахи у таблиці, що оновлюється покроково. Завдяки цьому можна аналізувати ефективність прийнятих рішень, перевіряти відповідність до очікуваних результатів та проводити оптимізацію параметрів алгоритму. Візуалізація та таблиця разом утворюють прозоре середовище, в якому вся динаміка алгоритму стає доступною для глибокого аналізу.

3.2.3 Інструкція до програми

Під час запуску програми з’являється вікно, в якому по центру розташовано об’єкт (мурашник), одразу після запуску з нього рухаються мурахи в кількості 512 (за бажанням це число можна поміняти на будь яке інше) рухаються в різні напрямки в пошуках їжі, див малюнок 3.1.

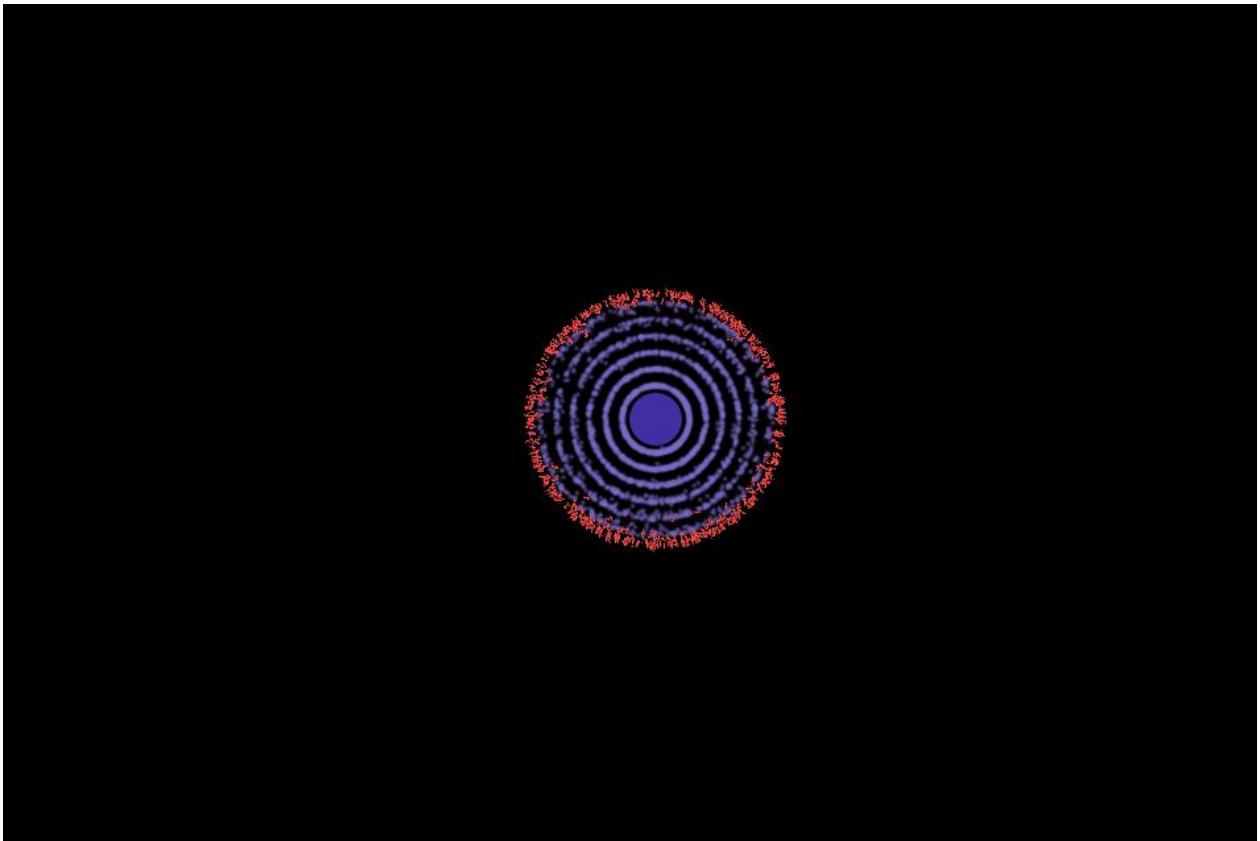
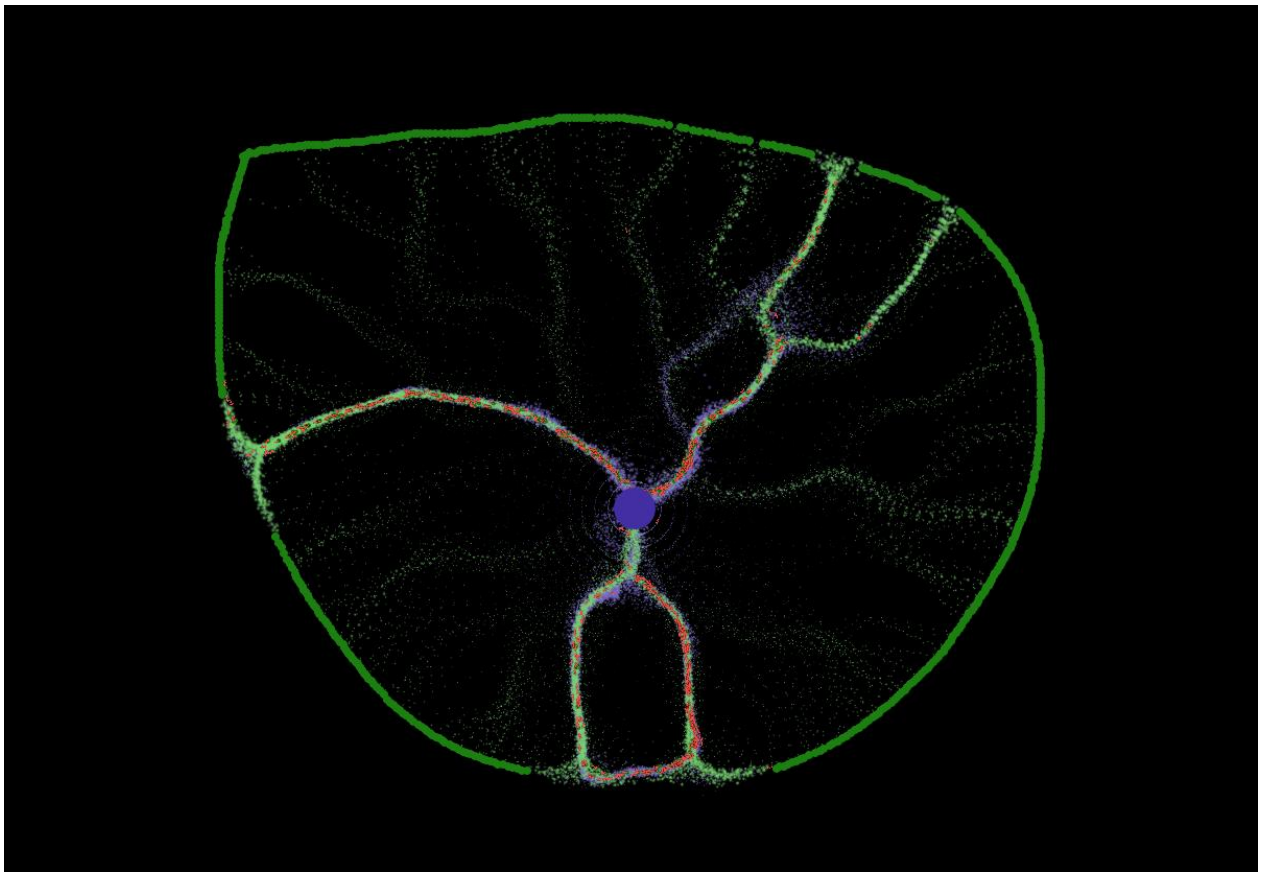


Рисунок 3.1.Візуалізація руху комах на старті програми

Під час запуску ми можемо створити правою клавішею миші їжу, коли мураха її знайде то буде повертатись до мурашника за допомогою слідів



(феромонів), також мураха з їжею залишає особливий феромон з їжею, по яких інші мурахи можуть зрозуміти де є їжа.

Рисунок 3.2.Візуалізація оптимальних маршрутів, коли їжа розташована навколо мурашника.

Висновки до розділу 3

У процесі реалізації аналітичного розділу ми переконалися, що ідеї мурашиних алгоритмів природно переходять у практичні рішення. Використання багатопотокових методів у побудові маршрутів дозволило суттєво скоротити час пошуку оптимального шляху без втрати точності — завдяки ефективному розподілу обчислювального навантаження.

Під час створення візуалізації у Windows Forms стало очевидно, що важливо не лише математичне обґрунтування, а й зручне представлення процесу. Візуалізація надала змогу краще зрозуміти внутрішню логіку алгоритму: накопичення феромонів, адаптивність і повернення найкоротшим маршрутом.

Порівняння послідовного та паралельного виконання показало перевагу багатопотоковості при роботі з великими графами. Водночас було виявлено, що надмірна кількість потоків без синхронізації може спричинити неефективність.

Модель мурашиної поведінки чітко виділила найкоротший маршрут через поступове накопичення феромонів — подібно до стежки, що формується постійним використанням. Це підкреслило здатність алгоритму до самонавчання.

Динамічне оновлення таблиці кроків кожної мурахи дозволило аналізувати процес у реальному часі, що сприяло глибшому розумінню алгоритму та його вдосконаленню.

У результаті дослідження стало ясно: мурашиний алгоритм, навіть у спрощеній формі, є гнучким, адаптивним та ефективним інструментом маршрутизації, особливо у складних та змінних середовищах. Його природна здатність до самоорганізації робить його цінним для подальших досліджень і практичних застосувань.

ВИСНОВКИ

Виконана дипломна робота присвячена дослідженню, розробці та впровадженню багатопотокових методів формування маршрутів з використанням мурашиного алгоритму оптимізації, що є сучасним і перспективним інструментом для задач маршрутизації в складних динамічних системах. В процесі дослідження було системно проаналізовано теоретичні основи алгоритмів оптимізації, особливості багатопотокового програмування та способи підвищення продуктивності обчислювальних процесів.

Загальні науково-практичні результати

Теоретичний аналіз і систематизація знань.

У першому розділі проведено детальний огляд теоретичних основ мурашиного алгоритму, багатопотокового програмування та методів оптимізації маршрутів. Встановлено взаємозв’язок між розвитком методів штучного інтелекту, евристичних алгоритмів і сучасними вимогами до обробки великих обсягів даних у режимі реального часу. Це дозволило сформувати чітку концептуальну базу для подальших розробок.

Розробка та імплементація багатопотокового алгоритму. Другий розділ містить практичну реалізацію мурашиного алгоритму з використанням багатопотокових технологій у середовищі Windows Forms на мові C#. Впровадження паралельних потоків дало змогу значно підвищити швидкодію та ефективність пошуку оптимальних маршрутів, що особливо важливо для задач із великим числом вузлів та динамічно змінними параметрами.

Візуалізація, тестування і оцінка результатів. У третьому розділі проведено комплексне тестування алгоритму на різних типах графів та сценаріях, що імітують реальні умови роботи логістичних

систем. Реалізовано графічне відображення процесу пошуку маршруту, що дозволяє не лише відслідковувати рух мурах, а й аналізувати структуру та якість обчислень. Порівняння послідовної та паралельної реалізації підтвердило суттєве скорочення часу виконання за рахунок розпаралелювання.

Практичне значення

Результати роботи мають високий прикладний потенціал для впровадження у сфері транспортної логістики, управління складськими комплексами, планування маршрутів доставки і розподілу ресурсів. Використання багатопотокових методів дозволяє досягти високої продуктивності навіть на апаратному забезпеченні середнього рівня, що робить рішення доступним для широкого спектра користувачів і підприємств. Мурашиний алгоритм, адаптований під багатопотокову архітектуру, забезпечує стійкість і адаптивність системи в умовах змінних параметрів середовища.

Перспективи подальших досліджень і розвитку

Проведене дослідження відкриває широкі можливості для подальшого розвитку у кількох напрямах:

Розробка гібридних алгоритмів, що поєднують мурашиний підхід із іншими евристиками або методами машинного навчання, для підвищення точності та швидкості пошуку рішень.

Оптимізація алгоритмів багатопотокового виконання з урахуванням розподілених обчислень та використанням апаратних ресурсів кластерів і хмарних платформ.

Створення інтерактивних систем управління логістикою з розширеною візуалізацією, що враховують реальні часові обмеження, умови руху, дорожню ситуацію та інші зовнішні фактори.

Розробка модулів самонавчання та адаптації маршрутів у реальному часі з урахуванням зміни трафіку, погодних умов і технічного стану транспортних засобів.

Загальна оцінка виконаної роботи.

Виконана дипломна робота є вагомим внеском у сферу дослідження методів оптимізації маршрутів і демонструє ефективність використання багатопотокового програмування для складних обчислювальних задач. Отримані результати можуть бути використані як основа для створення комерційних продуктів у сфері логістики та транспорту, а також для подальших наукових досліджень в галузі штучного інтелекту та паралельних обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бартіш М. Я. Дослідження операцій. Ч. 2. Алгоритми оптимізації на графах, 2007. 168 с.
2. Кметь О.І. Мурашиний алгоритм для розв'язання задач маршрутизації транспорту https://openarchive.nure.ua/server/api/core/bitstreams/133ac692-b901-43ba-a916-7a99616c43b1/content
3. Коротач І.В. Дослідження методів мурашиних колоній для вирішення задачі QSAR. https://openarchive.nure.ua download
4. Лекція 10. Мурашині алгоритми та інші види еволюційних алгоритмів https://learn.ztu.edu.ua ШІ_Л-10_ГА2_22
5. Литвинов О.А., Хандецький В.С. Л Л64 Розподілена обробка інформації : [моногр.] Донецьк: ТОВ «Баланс-Клуб», 2013. 314 с.
6. Михайленко Р.М. Паралельні та розподілені обчислення: навч. посіб. Кропивницький: Видавець Лисенко В. Ф., 2021. 153 с.
7. Нікольський Ю. В., Пасічник В. В., Щербина Ю. М. Дискретна математика: Підручник. Київ: Видавнича група BHV, 2007. 368 с.
8. Семеренко В. П. Технології паралельних обчислень: навчальний посібник Вінниця : ВНТУ, 2018. 104 с.
9. Сизько В.А., Бабенко М.В., Лимар Н.М., Госало І.О Розробка паралельного мурашиного алгоритму на прикладі задачі комівояжера. Комп'ютерні науки та інформаційні технології. Збірник наукових праць Дніпровського державного технічного університету. 2020. С 105-108. http://sj.dstu.dp.ua/article/view/211502
10. Тимчук О.С., Проценко Я.А., Парамонов А.І. Застосування алгоритму мурашиної колонії до вирішення задачі декількох комівояжерів без депо. Системи обробки інформації, 2019, випуск 3 (158). С. 73-78.

11. Штовба С. Д., Рудий О. М. Мурашині алгоритми оптимізації. Вісник Вінницького політехнічного інституту. 2004. № 4, С. 62-69.
12. Яровий, А. А. Методи та засоби організації високопродуктивних паралельно-ієрархічних обчислювальних систем із рекурсивною архітектурою : монографія Вінниця : ВНТУ, 2016. 363 с.

ДОДАТКИ

Кодова складова програмного продукту

main.cpp

Головний файл програми: створює вікно, оновлює мурах, їжу, феромони та візуалізує

```
#include <SFML/Graphics.hpp>
#include "Ant.hpp"
#include "Food.hpp"
#include "PheromoneGrid.hpp"
#include <vector>
#include <ctime>
#include <cstdlib>

const int GRID_SIZE = 100;
const int CELL_SIZE = 8;
const int WINDOW_SIZE = GRID_SIZE * CELL_SIZE;
const int NUM_ANTS = 200;

int main() {
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    sf::RenderWindow window(sf::VideoMode(WINDOW_SIZE, WINDOW_SIZE), "Ant Simulation");
    window.setFramerateLimit(60);
```

```
    sf::Vector2i home(GRID_SIZE / 2, GRID_SIZE / 2);
    std::vector<Food> food_sources;
    std::vector<Ant> ants;
    PheromoneGrid pheromones(GRID_SIZE);

    for (int i = 0; i < NUM_ANTS; ++i)
        ants.emplace_back(home);

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::MouseButtonPressed &&
event.mouseButton.button == sf::Mouse::Left) {
                int mx = event.mouseButton.x / CELL_SIZE;
                int my = event.mouseButton.y / CELL_SIZE;
                food_sources.push_back({ sf::Vector2i(mx, my) });
            }
        }

        for (auto& ant : ants)
```

```
            ant.update(food_sources, pheromones, home);

        pheromones.evaporate();

        window.clear();
        pheromones.draw(window, CELL_SIZE);

        for (const auto& food : food_sources)
            food.draw(window, CELL_SIZE);

        for (const auto& ant : ants)
            ant.draw(window, CELL_SIZE);

        sf::CircleShape home_dot(CELL_SIZE);
        home_dot.setPosition(home.x * CELL_SIZE, home.y * CELL_SIZE);
        home_dot.setFillColor(sf::Color(100, 100, 255));
        window.draw(home_dot);

        window.display();
    }
    return 0;
}
```

Ant.hpp

Клас Ant: логіка мурах — випадковий рух, пошук їжі, запам’ятовування шляху, феромони

```
#pragma once
#include <SFML/Graphics.hpp>
#include "Food.hpp"
#include "PheromoneGrid.hpp"
#include <vector>

struct Ant {
    sf::Vector2i pos;
    bool has_food = false;
    std::vector<sf::Vector2i> path;

    Ant(sf::Vector2i start) : pos(start) {}

    void update(std::vector<Food>& food_sources, PheromoneGrid& pheromones,
sf::Vector2i home);
    void moveRandom();
    void draw(sf::RenderTarget& target, int cellSize) const;
};
```

Ant.cpp

```
#include "Ant.hpp"

#include <cstdlib>


void Ant::update(std::vector<Food>& food_sources, PheromoneGrid&
pheromones, sf::Vector2i home) {

    if (has_food) {

        if (!path.empty()) {

            pos = path.back();

            path.pop_back();

            pheromones.add(pos, 2.0f);

        } else {

            has_food = false;

        }

    } else {

        moveRandom();

        path.push_back(pos);

        for (auto& food : food_sources) {

            if (food.active && pos == food.pos) {

                has_food = true;

                food.active = false;

                break;
```

```
            }
        }
    }
}

void Ant::moveRandom() {
    int dx = (std::rand() % 3) - 1;
    int dy = (std::rand() % 3) - 1;
    pos.x = std::max(0, std::min(GRID_SIZE - 1, pos.x + dx));
    pos.y = std::max(0, std::min(GRID_SIZE - 1, pos.y + dy));
}

void Ant::draw(sf::RenderTarget& target, int cellSize) const {
    sf::CircleShape a(cellSize / 3);
    a.setPosition(pos.x * cellSize + cellSize / 4, pos.y * cellSize + cellSize / 4);
    a.setFillColor(has_food ? sf::Color::Blue : sf::Color::Black);
    target.draw(a);
}
```

Food.hpp

Структура Food: представляє їжу, яку можуть знаходити мурахи

```
#pragma once
#include <SFML/Graphics.hpp>

struct Food {
    sf::Vector2i pos;
    bool active = true;

    void draw(sf::RenderTarget& target, int cellSize) const {
        if (!active) return;
        sf::CircleShape shape(cellSize / 2);
        shape.setPosition(pos.x * cellSize, pos.y * cellSize);
        shape.setFillColor(sf::Color::Green);
        target.draw(shape);
    }
};
```

PheromoneGrid.hpp

Клас PheromoneGrid: сітка феромонів, випаровування, візуалізація

```
#pragma once
#include <SFML/Graphics.hpp>
#include <vector>
```

```
struct PheromoneGrid {
    int size;
    std::vector<std::vector<float>> grid;

    PheromoneGrid(int s) : size(s), grid(s, std::vector<float>(s, 0.0f)) {}

    void add(sf::Vector2i pos, float value) {
        if (pos.x >= 0 && pos.x < size && pos.y >= 0 && pos.y < size)
            grid[pos.y][pos.x] += value;
    }

    void evaporate() {
        for (auto& row : grid)
            for (auto& val : row)
                val = std::max(0.0f, val - 0.01f);
    }

    void draw(sf::RenderTarget& target, int cellSize) const {
        for (int y = 0; y < size; ++y) {
            for (int x = 0; x < size; ++x) {
                float intensity = grid[y][x];
                if (intensity > 0.01f) {
```

```
                sf::RectangleShape dot(sf::Vector2f(cellSize, cellSize));
                dot.setPosition(x * cellSize, y * cellSize);
                dot.setFillColor(sf::Color(255, 100, 255, std::min(255.0f, intensity *
30)));
                target.draw(dot);
            }
        }
    }
};
```

Utils.hpp

Допоміжні функції: clamp, конвертації, випадковість

```
#pragma once
#include <algorithm>
#include <cmath>

template<typename T>
T clamp(T value, T min, T max) {
    return std::max(min, std::min(value, max));
}
```