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

Кваліфікаційна робота магістра

з теми: «Метод оптимізації маршрутів в мережах зі змінною пропускною здатністю»

Виконав: здобувач вищої освіти групи Kn1-M24
спеціальності 122 Комп’ютерні науки
Колесник Денис Олександрович
Керівник: Щирба Віктор Самуїлович,
кандидат фізико-математичних наук, доцент
Рецензент: Оптасюк Сергій Васильович,
кандидат фізико-математичних наук, доцент

м. Кам’янець-Подільський – 2025 р.

Зміст

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці та дослідженню методу оптимізації маршрутів у мережах зі змінною пропускною здатністю на прикладі динамічної задачі комівояжера.

У роботі використано апарат теорії графів та динамічного програмування для моделювання мережевих задач. Основним інструментом дослідження обрано мурашиний алгоритм.

Запропоновано та реалізовано модифікований мурашиний алгоритм, який адаптований для роботи в динамічному середовищі. Встановлено, що в динамічній задачі оптимальний результат залежить не лише від послідовності вершин, а й від оптимальної стартової точки та напрямку руху.

Практичне значення роботи полягає в можливості використання розробленого методу для оптимізації логістичних маршрутів, транспортних потоків та управління комунікаційними мережами в умовах, де мережеві параметри є нестабільними.

ВСТУП

Актуальність теми. Застосування графових моделей у комп'ютерно-інформаційних технологіях і досі залишається досить актуальною темою для теоретичних досліджень, оскільки прикладні графові мережі постійно зростають у розмірах, що, у свою чергу, веде до підвищення складності алгоритмічного розв’язання. Сьогодні графові алгоритми знайшли своє практично неоціненне застосування у таких областях, як економіка, транспорт, медицина, соціологія, фінанси та багато інших, що підвищує значущість дослідження проблем використання графів. Особливої актуальності набувають задачі пошуку оптимальних маршрутів, серед яких ключове місце займає задача комівояжера (Traveling Salesman Problem – TSP), особливо у її динамічній постановці, де пропускна здатність або ваги ребер мережі змінюються з часом. В умовах сучасних транспортних, логістичних та комунікаційних систем, які є динамічними за своєю природою, розробка та модифікація ефективних методів маршрутизації є критично важливою для мінімізації витрат та часу.

Мета дослідження – розробити та дослідити метод оптимізації маршрутів у мережах зі змінною пропускною здатністю шляхом модифікації мурашиного алгоритму для ефективного розв’язання динамічної задачі комівояжера.

Для досягнення поставленої мети визначено наступні завдання:

1. Провести аналіз існуючих графових моделей і алгоритмів пошуку оптимальних маршрутів, зокрема в контексті динамічних систем.
2. Сформулювати математичну модель задачі комівояжера в умовах змінної пропускної здатності мережі, використовуючи апарат орієнтованих зважених графів.

3. Розробити модифікацію Мурашиного алгоритму, яка враховує динамічну зміну ваг ребер (пропускної здатності/вартості).
4. Провести порівняльний аналіз ефективності запропонованого модифікованого алгоритму порівняно з базовим підходом на прикладі тестових даних.

Об'єкт дослідження – процес оптимізації маршрутів у графових мережах.

Предмет дослідження – метод оптимізації маршрутів у мережах зі змінною пропускною здатністю на основі модифікованого мурашиного алгоритму.

У роботі використано методи теорії графів для моделювання мереж та маршрутів, теорії динамічного програмування для аналізу багатокрокових процесів прийняття рішень, а також методи евристичної оптимізації (зокрема, Мурашиний алгоритм) та комп’ютерного моделювання для розробки та оцінки ефективності запропонованого алгоритму.

Наукова новизна одержаних результатів полягає у:

- Розробці модифікації мурашиного алгоритму, спеціально адаптованої для розв’язання динамічної задачі комівояжера в мережах зі змінною пропускною здатністю/вагою ребер.
- Обґрунтуванні застосування принципу динамічного програмування для формування стратегії оновлення феромонів (ваг) у модифікованому мурашиному алгоритмі.

Практичне значення одержаних результатів полягає в тому, що розроблений модифікований алгоритм може бути використаний для підвищення ефективності планування логістичних маршрутів, оптимізації транспортних потоків, а також у проектуванні та управлінні комп'ютерними та комунікаційними мережами, де пропускна здатність елементів є змінною в часі.

За результатами дослідження опубліковано одну роботу: Віктор ЩИРБА, Дмитро КОЛЕСНИК Метод оптимізації маршрутів в мережах зі змінною пропускною здатністю. Вісник Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 17. 2024, с 208-211.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні основи теорії графів та проведено огляд літературних джерел, присвячених мережевим задачам. У другому розділі проаналізовано прикладні динамічні задачі та їх моделювання за допомогою орієнтованих графів і методу динамічного програмування. У третьому розділі представлено модифікацію мурашиного алгоритму для відшукання оптимального маршруту в динамічній задачі комівояжера.

РОЗДІЛ 1 ПРИКЛАДНІ МЕРЕЖЕВІ ЗАДАЧІ ТА ГРАФОВІ МОДЕЛІ

1.1 Огляд літературних джерел

Успішний розвиток економічної складової будь-якої галузі виробництва загалом чи окремо взятого підприємства зокрема залежить в значній від вдалих рішень, пов’язаних з управлінням. Безумовно, наскільки б не було розвинена інтуїція в керівного складу, на скільки не був би великим життєвий досвід керівників, охопити усі питання та багатогранні фактори не можуть і тому ніхто не гарантує, що ухвалене рішення абсолютно правильним.

Тому широкого розвитку набули дослідження, так званих, математичних методів в економіці, з’явився науковий напрямок дослідження операцій, левову частину в теоріях яких займають питання планування за допомогою графів. У побудованих математичних (графових) моделях пошук правильних рішень в основному зводиться до пошуку оптимальних маршрутів в графічній моделі. Цим можна пояснити наявність широкого кола як наукової, так і навчальної літератури з питань дослідження теорії графів.

Виходячи з завдань, що поставлено мені для магістерського дослідження, із доступних джерел інформації, які я розглядав, для дослідження і використання на мою думку достатньо вибрати 12 джерел. Їх умовно можна умовно поділити на три групи: загальні відомості про графи, прикладні задачі, що моделюються за допомогою графів та алгоритми обробки графів.

Загальні відомості, які стосуються даних про графи і використовуються в кваліфікаційній роботі, я описав в пункті 1.2. Більш детальну інформацію можна одержати з навчальної чи навчально-методичної літератури з дискретної математики, яка досить широко представлена, наприклад, в Інтернеті.

Для своїх викладок я використовував в основному підручник з дискретної математики [11]. При цьому мене найбільше цікавив матеріал про основні типи графів та методи їх задання в пам’яті комп’ютера.

Практичне застосування теорії графів, зокрема, описано в [1, 5, 6, 7, 8]. В підручниках з дискретної математики про застосування графів говориться мало.

Більше інформації про застосування графів для моделювання прикладних задач можна одержати з навчальної літератури, що стосується дослідження інформацій.

Методи побудови оптимальних маршрутів, алгоритми та специфіка їх застосування для різних типів пошукових задач описані, зокрема, в [2, 3, 8, 9, 10, 12]. В цьому питанні потрібно спочатку розглянути модель і встановити тип оптимізаційної задачі, а тоді вибирати з існуючих алгоритмів той, який вам найбільше підходить.

В основу свого дипломного дослідження я взяв задачу комівояжера, яка описана, зокрема, в [7, 4].

В роботі [7] фактично розміщений навчальний матеріал, а елементи наукової новизни містяться в науковій статті [4].

В роботі [4] мене зацікавило перш за все словосполучення «в динамічній задачі» і я намагався провести модифікацію запропонованого алгоритму таким чином, щоб одержати, на мою думку, новіші чи модифіковані, більш загальні результати.

1.2 Основні поняття теорії графів

Граф, як математичне поняття, досить активно використовується при моделюванні цілого ряду прикладних задач. Зазвичай, граф подають як зображення, що складається з точок і відрізків, але таке визначення не підходить для комп’ютерної обробки графічних моделей.

Найчастіше перевагу надають табличному представленню, зокрема, за допомогою матриці суміжності. В залежності від специфіки моделі іноді використовують подання через множини вершин та ребер або через матрицю інцидентності.

Враховуючи характерні властивості прикладних задач, які я розглядатиму в своїй роботі, я обрав задання графу за допомогою вагової матриці суміжності і в даному підрозділі використовуватиму лише ті поняття з теорії графів, які стосуються цих прикладних задач.

Найпростішим типом графів є неорієнтовані графи. В них матриця суміжності буде симетричною, але більш загальним прикладом виступають орієнтовані та мішані графи, тобто графи, в яких кожне ребро має напрямок (орієнтовані ребра) та графи, які мають як орієнтовані, так і неорієнтовані ребра. Матриця суміжності для них буде не симетричною.

Оскільки кожне неорієнтоване ребро можна замінити парою протилежно направлених ребер, то для загальності теоретичних викладок варто відразу вважати, що граф орієнтований.

Також врахуємо, що в ряді специфічних задач особливістю моделі виступає наявність кратних ребер чи петель. Отже, в побудованій моделі графи можуть бути мультиграфами чи псевдографами відповідно. Легко бачити, що коли на кожному кратному ребрі та кожній петлі встановити по додатковій вершині, то такий модифікований граф стане простим (без кратних ребер та петель). Отже, в кваліфікаційній роботі достатньо обмежитися розглядом простих орієнтованих графів.

Абсолютна більшість прикладних задач, що моделюються за допомогою графів, пов’язана з задачами маршрутизації. Незважаючи на те, що під маршрутом, взагалі кажучи, потрібно розуміти послідовність вершин і ребер, в якій кожні два сусідні ребра мають спільну вершину, я, враховуючи, що в роботі розглядатиму лише прості графи, обмежуся лише ланцюгами і

маршрути розглядатиму як послідовність вершин від стартової вершини до фінішної.

Я не буду обмежуватися лише простими ланцюгами, в яких жодна вершина не повторюється тобто, що маршрут не містить циклів. Навпаки, в задачі комівояжера вимагається щоб стартова точка співпадала з фінішною тобто, весь шлях був циклом.

В кваліфікаційній роботі не передбачається дослідження моделей, пов'язаних з деревами та планарними графами. Тому я не розглядатиму ці поняття.

У реальних прикладних задачах, які моделюються за допомогою графів, досить часто необхідно враховувати потребу в використанні не лише наявності зав'язків, але й величину їх впливу, що відображається додатковою інформаціє. Це може бути сила магнітного поля, сила притягання, відстань, час чи вартість проїзду тощо. Тоді для моделювання використовують, так званні, зважені графи, тобто графи, в яких кожному ребру ставлять у відповідність деяке дійсне число, яке називають вагою ребра.

«Не зважені» графи є частковим випадком зважених графів, в яких вага кожного ребра рівна одиниці. Якщо в не зваженому графі довжину маршруту визначають за кількостю ребер в маршруті, то в зваженому графі довжина маршруту дорівнює сумі ваг ребер маршруту.

Задавати зважені графи також зручно ваговою матрицею. Вона є узагальненням поняття матриці суміжності. У ваговій матриці в *i*-тому рядку та *j*-тому стовпцеві проставляємо значення a_{ij}, якщо вага орієнтованого ребра, що веде з *i*-тої вершини в *j*-ту вершину, становить a_{ij}. Якщо ребро відсутнє, то величину (вагу) ребра, зазвичай, покладають рівною $+\infty$ або 0.

Природньо, що в задачах маршрутизації важливим є питання існування маршруту. В теорії графів воно відоме як питання зв'язності. Будемо розглядати лише зв'язні орієнтовані графи, тобто графи, у яких існує

орієнтований маршрут з будь-якої вершини в іншу або умовно зв’язні орієнтовані графи (стають зв’язними, якщо орієнтовані мости замінити на не орієнтовані).

В багатьох алгоритмах, зокрема при розв’язуванні задачі комівояжера мурашиним алгоритмом, вимагають щоб граф був повним (для будь-яких двох вершин існує ребро, яке веде з однієї вершини в іншу). Для орієнтованих графів це означає, щоб існувала пара протилежно направлених ребер.

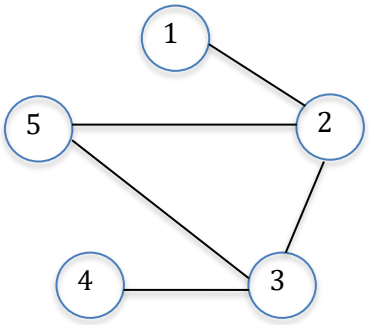
Не повні вагові графи можна перетворити в умовно повні, якщо вагу неіснуючого ребра встановити рівною вазі найкоротшого маршруту, що з’єднує ці вершини, або рівною $+\infty$, якщо ці вершини не зв’язні.

В подальшому будемо розглядати оптимізаційні задачі, які пов'язані з пошуком найкоротших або мінімальних шляхів і часто моделюються як задачі на повних графах.

Тому в своїй роботі буду розглядати найбільш загальні моделі для такого типу задач, а саме скінченні повні (або умовно повні) орієнтовані зв’язні вагові графи і подаватиму їх для комп’ютерної обробки за допомогою вагової матриці.

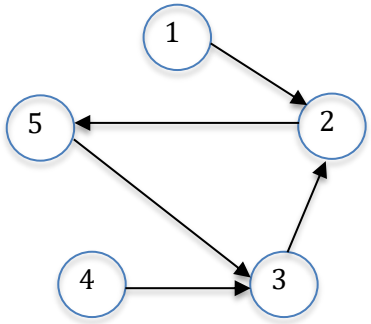
Продемонструю на прикладах, як графічне представлення різного типу графів можна подати у вигляді вагової матриці.

Для графу, зображеного на рисунку 1.1, вагову матриця в навчальній літературі, зазвичай, називають матрицею суміжності і подають не у вигляді таблиці, а у вигляді матриці. При цьому замість знаку «∞», зазвичай, використовують число 0.



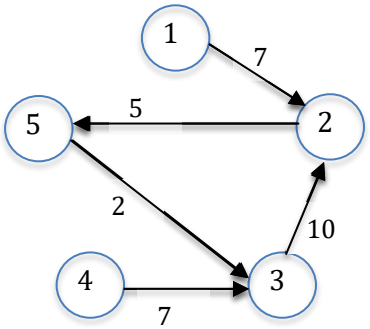
Вершини	1	2	3	4	5
1	∞	1	∞	∞	∞
2	1	∞	1	∞	1
3	∞	1	∞	1	1
4	∞	∞	1	∞	∞
5	∞	1	1	∞	∞

Рисунок 1.1. Графічне зображення (зліва) та вагова матриця (справа) простого неорієнтованого графу



Вершини	1	2	3	4	5
1	∞	1	∞	∞	∞
2	∞	∞	∞	∞	1
3	∞	1	∞	∞	∞
4	∞	∞	1	∞	∞
5	∞	∞	1	∞	∞

Рисунок 1.2. Графічне зображення (зліва) та вагова матриця (справа) простого орієнтованого графу (вага кожного ребра вважається рівною 1)



Вершини	1	2	3	4	5
1	∞	7	14	∞	12
2	∞	∞	7	∞	5
3	∞	10	∞	∞	15
4	∞	17	7	∞	22
5	∞	12	2	∞	∞

Рисунок 1.3. Графічне зображення (зліва) та вагова матриця (справа) простого умовно зв’язного орієнтованого графу з вагою

1.3 Основні типи задач, пов’язаних з графами

В теорії графів (див., наприклад, [11]) при розгляді різних типів задач, пов’язаних з графами, ставлять різноманітні завдання. Умовно задачі з дослідження графів можна поділити на три групи:

1. Встановлення властивостей графа.

2. Пошук шляху в графі.

3. Операції над графами.

До першого типу задач віднести ряд задач на дослідження конкретного графу. Сюди можна віднести, наприклад, такі, як визначення кількості ребер, доведення ізоморфізму графів, встановлення кількості вершин непарного степеня, визначення кількості висячих вершин, відшукання мостів, перевірку графа на планарність тощо.

До другого типу відносять задачі декількох видів пошукових задач: пошук якого небуть маршруту з однієї вершини в іншу чи пошук всеможливих маршрутів з однієї вершини в іншу, пошук оптимального (найдовшого чи найкоротшого) маршруту з однієї вершини в іншу, пошук центральних чи периферійних вершин, пошук простих ланцюгів, пошук ейлерових маршрутів тощо.

Пошук оптимального маршруту може бути пов’язаний не тільки з довжиною ребер, але й властивостями вершин. Наприклад, при перевезенні небезпечних вантажів намагаються вибрати маршрут через населені пункти з найменшою кількістю жителів або щоб кількість мостів по трасі була мінімальною тощо.

До третьої групи відносять задачі, пов’язані з операціями над графами: побудова кістякових дерев, визначення перетину графів чи доповнення графів, вилучення окремих вершин або ребер тощо.

Кожна із цих задач має свої алгоритми розв’язання і, отже, алгоритмів різного призначення для дослідження графів є дуже велика кількість.

В своїй роботі я буду досліджувати задачі пов’язані з алгоритмами пошуку оптимальних маршрутів.

Задачі з дослідження графів знаходять широке застосування в різноманітних сферах науки та техніки таких, як логістика, транспортні перевезення, проектування комп'ютерних мереж, дослідження соціальних процесів, енергетична сфера тощо. Зокрема, розглядаючи групу оптимізаційних задач (пошук найдовшого чи найкоротшого шляху графів), можна виділити їхню актуальність у таких важливих практичних аспектах, як транспортна задача та задача планування будівельних робіт.

Транспортна задача в графовій моделі передбачає оптимізацію перевезення деяких ресурсів між пунктом постачання та одним або декількома пунктами споживання. Прикладом останньої задачі служить задача перевезення деякого товару з підприємства-виробника чи бази постачання по декількох торгових точках міста.

Ця задача потребує визначення найкращого або групи найкращих шляхів за встановленими критеріями оптимальності: з мінімальними витратами палива, з врахуванням аналізу пропускної здатності ребер графа, з врахуванням часових обмежень (наприклад, якнайшвидше для пожежних машин або не запізнитися на потяг для водія таксі), з дотриманням специфічних вимог до маршруту (наприклад, обмеження на вантажопідйомність або на безпеку перевезення).

Для вирішення таких оптимізаційних задач можна використовувати алгоритм лінійного програмування або спеціалізовані алгоритми, наприклад, алгоритм Форда-Фалкерсона для відшукання максимального потоку.

При плануванні будівельних робіт, графи використовуються для проектування оптимальних транспортних розв’язок чи комунікаційних мереж.

Зокрема, проектуючи автомобільні дороги чи залізничні шляхи, важливо вибрати не просто найкоротший маршрут, але й мінімізувати витрати на проведення будівельних робіт. Отримуємо зразок багатокритеріальної задачі.

Для різних типів задач розроблено велику кількість спеціалізованих алгоритмів. Наприклад алгоритм Дейкстри спеціалізується на пошуку найкоротших маршрутів, а алгоритм Крускала використовують для оптимізації (мінімізації) загальної вартості мережі.

Отже, задачі другої групи, які стосуються пошуку маршруту в графі, є цікавими не лише теоретичному плані, але й становлять значний практичний інтерес. Методи їх розв'язання дозволяють вирішувати досить складні проблеми в галузях, пов'язаних із транспортом, комунікаціями та інфраструктурою. Це постає ключовим аспектом актуальності алгоритмів дослідження графів.

Висновки до розділу 1

Застосування теорії графів в сучасних інформаційно-комп'ютерних технологіях постає однією з найбільш важливих напрямків дослідження у сучасній науці. Проведений аналіз літературних джерел показав, що застосування графів у прикладних інформаційно-комп'ютерних технологіях має величезний потенціал. Графи можуть використовуватися при моделюванні і дослідженні багатьох типів прикладних задач. Одними із найбільш вживаними виступають задачі на пошук оптимальних маршрутів, які описуються несиметричною ваговою матрицею, яка описує скінчений орієнтований граф з вагою.

РОЗДІЛ 2. ПРИКЛАДНІ ДИНАМІЧНІ ЗАДАЧІ В МОДЕЛЯХ ОРГРАФІВ

2.1. Основні поняття динамічної системи

Динамічна система — це множина взаємопов'язаних елементів, що змінюється з часом. Під змінами необхідно розуміти зміни станів системи, її властивостей. При цьому час є домінуючим фактором динаміки. Тут важливо розглядати послідовність процесів, станів. Наприклад, що раніше одягати сорочку чи пальто (в якій послідовності), а скільки часу на це піде тут не важливо.

Для дослідження динамічної системи формують послідовність станів системи s_0, s_1, ..., s_n, в якій описують зміни властивостей динамічних процесів системи, починаючи з початкового, стартового стану s_0, і завершуючи кінцевим, фінішним станом s_n.

Варто зазначити, кінцевих (фінішних) станів може бути декілька. Наприклад, залежно від того, яке управління вибрати, куди повертати автомобілем, за один і той же відведений проміжок часу можна потрапити в різні місця, фінішні точки.

Кожен зі станів характеризується набором з k значень певних властивостей системи v_1, v_2, ..., v_k.

Наприклад, у прогнозі погоди вказується значення температури повітря, вологості, тиску, швидкості та напряму вітру тощо через кожні 3 години впродовж дня, якщо прогноз розрахований на один день, або кожного дня, якщо на декілька днів.

Отже, динамічна система, зазвичай, може бути багатопараметричною, а кількість станів залежить від кількості вузлів (фіксованих точок) на часовому інтервалі.

При моделюванні динамічної системи ключовими характеристиками виступають властивості зв'язків між складовими елементами системи. Тоді наявність зв'язку моделюється наявністю відповідного ребра між відповідними вершинами графа, яким відповідає той чи інший елемент системи.

Якщо сила зв'язку піддається числовій характеристиці, а не лише фіксується його наявність, то ребру присвоюється деяка вага. Тоді математична модель задається не матрицею суміжності а ваговою матрицею.

Оскільки матриця не обов'язково повинна бути симетричною, то в загальному випадку розглядають орієнтовані графи (орграфи).

В окремих прикладних задачах зв'язки є багатопараметричними. Наприклад, для газових магістралей вказується максимальний об'єм подачі газу по цій чи іншій вітці та собівартість передачі одиниці об'єму газу тощо. Те саме, можна сказати про перенесення різних елементів забруднення. В таких випадках ребру присвоюється декілька параметрів, а модель подається декількома таблицями (матрицями).

Кожен стан s_j пов'язаний із множиною керувань $\{u_j^k\}$, кожне з яких забезпечує перехід динамічної системи з стану s_j у деякий наступний стан s_k. Наступність, тобто неможливість повернення часу в минуле, забезпечується нерівністю $k > j$. Зрозуміло, що реалізація будь-якого керування u_j^k забезпечує одержання деякого прибутку чи, навпаки, пов'язане із затратами, тобто воно має деяке цінове значення. Позначимо його через c_j^k.

В задачі відшукання оптимального керування динамічної системи потрібно вказати таку послідовність керувань, яка переводить систему з стартового стану s_0 у фінішний стан s_n і при цьому загальна сума значень затрат чи прибутків була б екстремальною (мінімальною чи максимальною відповідно). Шукану послідовність керувань називають оптимальним

керуванням, що переводить динамічну систему з стартового стану s_0 у фінішний стан s_n.

Для відшукання оптимального керування динамічною системою спочатку сформулюємо задачу в термінах орієнтованих графів. Побудуємо орієнтований граф, що складається з n вершин, які відповідають станам s_0, s_1, ..., s_n. Вершини s_j і s_k з'єднаємо дугою (орієнтованим ребром), якщо існує керування u_j^k, яке забезпечує перехід динамічної системи з стану s_j у наступний стан s_k (як зазначено вище, $k > j$). Вага дуги (s_j, s_k) становить c_j^k. Не обов'язково $c_j^k > 0$ (замість очікуваного прибутку окремі керування можуть призвести до збиткових ситуацій). Зрозуміло, що граф антициклічний (повернення в часі не можливе).

Задача відшукання оптимального керування рівносильна задачі відшукання в антициклічному орієнтованому графі маршруту, який з'єднує вершину s_0 з кінцевою вершиною s_n та має максимальну чи мінімальну сумарну вагу. Цей маршрут називають оптимальним маршрутом з вершини s_0 у вершину s_n.

Алгоритм розв'язування задачі пошуку оптимального керування методом динамічного програмування ґрунтується на принципі оптимальності Беллмана: для оптимального виходу із стану s_j, необхідно вибрати таке керування u_j^k, яке разом з оптимальним керуванням зі стану s_k становитиме оптимальне керування з стану s_j.

Тобто, для кожного стартового чи проміжного стану динамічної системи в результаті здійснення будь-якого числа кроків, керування на наступному, найближчому кроці потрібно обирати таким чином, щоб воно разом з проведеним оптимальним керуванням на усіх наступних кроках алгоритму привело до максимально чи мінімального можливого результату на всіх кроках, що ще залишилися, включаючи також і даний.

Кожну багатокрокову задачу прийняття якого-небуть керування можна сприймати як процес зміни станів динамічної системи з дискретним розбиттям часового інтервалу. Поняття стану динамічної системи відіграє дуже велику роль алгоритмізації дискретного динамічного програмування та може мати найрізноманітнішу інтерпретацію.

Ідея методу динамічного програмування відносно багатокрокових дискретних задач прийняття рішення з адитивною цільовою функцією заключається в покроковій оптимізації (найкраще рішення приймається на кожному кроці). Прийняття оптимального рішення на кожному кроці алгоритму, за виключенням останнього, відбувається з врахуванням усіх можливих результатів на наступних кроках.

Прийняття рішення на завершальному кроці набуває особливого значення тому, що прийняття рішення можна уже здійснювати не дивлячись на майбутній час. Отже, враховуючи максимальну можливу ефективність рішення, в алгоритмі динамічного програмування потрібно спочатку планувати останній, завершальний крок.

Розглянемо декілька прикладних задач з використанням алгоритму методу динамічного програмування в орієнтованому графі. Зокрема, цей метод можна з успіхом використати для розв’язання таких популярних задач, як задачі заміни обладнання, розподілу фінансів між виробничими підприємствами, розподілу обмеженого ресурсу, відому як завантаження транспортного засобу або задача про рюкзак.

2.2. Задача заміни обладнання

Задача заміни обладнання потребує встановлення оптимальних термінів проведення ремонтних робіт із заміни застарілого обладнання на нове. Внаслідок експлуатації обладнання відбувається його зношення, зростають

виробничі витрати на його модернізацію, витрати на ремонтні роботи і обслуговування, зменшується продуктивність і, так звана, ліквідна вартість.

Критерієм доцільності подальшого використання такого обладнання є сумарні затрати на обслуговування обладнання впродовж певного періоду.

Побудуємо концептуальну модель запропонованої задачі. Нам задані наступні початкові величини:

- Стартова, початкова вартість обладнання *start*;
- ліквідна вартість обладнання *likv*(*i*), де *i* - термін використання *i* років;
- затрати *zatr*(*i*) на обслуговування обладнання впродовж року, яке до початку річного періоду уже використовувалося *i* років.

По завершенню експлуатаційного періоду вивільнене обладнання необхідно продати. Потрібно встановити оптимальну стратегію (терміни) заміни обладнання за період перспективних *n* років експлуатації, яка мінімізує сумарні затрати на обслуговування обладнання.

Поточний стан такої динамічної системи визначається двома характеристиками (параметрами) *k* і *j*, де *k* – кількість років, що пройшли від часу першого придбання обладнання, *j* – вік обладнання, яке використовується на даний момент. Стан динамічної системи будемо позначати через s_{kj}. *Система локальних керувань* довільного стану складатиметься з двох можливих керувань:

- "*заміна*" – тобто продаж старого обладнання та закупівля нового;
- "*збереження*" – тобто продовження експлуатації обладнання впродовж наступного року.

Таким чином, керування "*заміна*" переміщує систему з стану s_{kj} в наступний стан $s_{k+1,1}$, а керування "*збереження*" переміщує систему з стану s_{kj} у наступний стан $s_{k+1,\,j+1}$.

Для визначення затрат на обслуговування обладнання після вибору керування при стані системи s_{kj} скористаємось наступним чином:

- якщо вибрано "заміна", то затрати становлять *start* + *zatr*(0) - *likv*(*j*),
- якщо вибрано "збереження", то затрати становлять *zatr*(*j*).

Нехай, для конкретності, використовувати обладнання планується 4 роки і закупівельна його вартість *start* становить 3000 одиниць вартості. Розглянемо задачу при таких бухгалтерських даних:

Таблиця 2.1. Бухгалтерські відомості про вартість обладнання.

i	0	1	2	3	4
likv(*i*)	–	1000	850	650	400
zatr(*i*)	100	900	1300	1800	–

Орієнтований граф задачі про заміну обладнання подано на рисунку 2.1. У вершинах графа розміщено індекси *k* та *j* відповідних станів s_{kj}. Граф матиме розгалужену кореневу структуру.

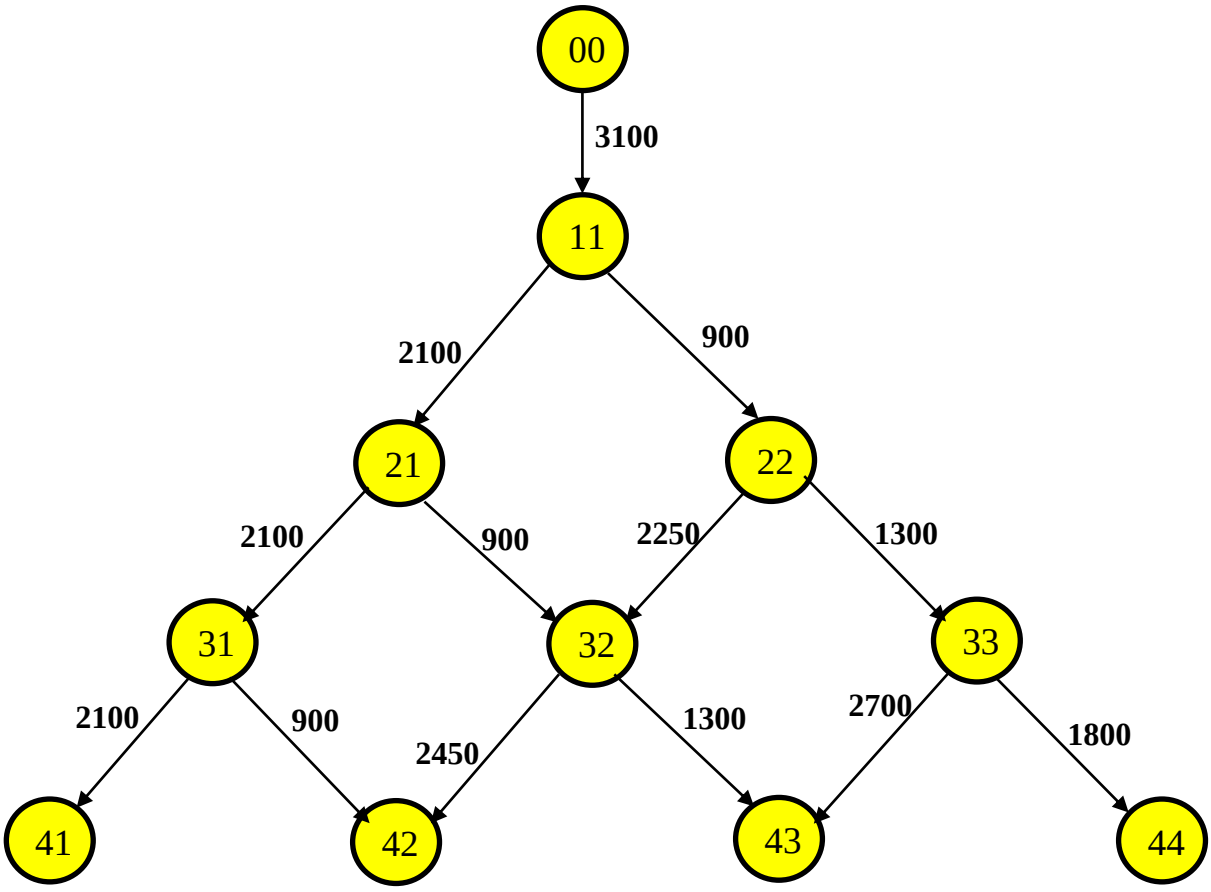


Рисунок 2.1. Орієнтований граф задачі про заміну обладнання

Ребро (00, 11) відображає передстартову роботу з прийняття обладнання на баланс підприємства, що полягає в закупці обладнання та його встановленні. Відповідно і вага цього ребра співпадає з балансовою вартістю, що співпадає з сумою вартості обладнання та вартості встановлення.

Послідовність вершин 11, 21, 31, 41 відображає експлуатацію обладнання без проведення ремонтно-профілактичних робіт, тобто без затрат на ці роботи. Тому вага відповідних ребер співпадає з балансовою вартістю, яка рівна стартовій балансовій вартості мінус ліквідність за один рік.

Послідовність вершин 11, 22, 33, 44 відображає період експлуатації обладнання з щорічним проведенням ремонтно-профілактичних робіт, тобто з затратами на ці роботи та ще й врахуванням балансової вартості.

Всі інші послідовності враховують відсутність проведення ремонтних робіт в окремі роки.

Для визначення оптимального маршруту (мінімальної довжини) скористаємось принципом зворотного руху, який полягає в умовній заміні напряму всіх дуг на протилежний. Як бачимо, найкоротшим буде зворотній маршрут 43, 32, 21, 11, 00. Тоді розв’язком задачі буде шлях 00, 11, 21, 32, 43.

2.3. Задача оптимального розподілу інвестицій

Нехай декільком підприємствам для розширення виробництва виділяються додаткові фінансові ресурси, які розбито на n однакових частин. В залежності від кількості одержаних часток грошей підприємство одержує певний приріст продукції. Необхідно так розподілити фінансові ресурси, щоб максимізувати сумарний приріст випуску продукції.

Стан s_{ij} системи підприємств зручно задавати параметрами i та j, де i вказує номер підприємства, а j вказує, що система підприємств уже одержала суму j грошових одиниць. Тоді граф матиме форму клітинок, де по вертикалі

стоятиме відповідне підприємство, а горизонталі відображатимуть кількість одержаних грошових одиниць.

Нехай, наприклад, n =3 (фінансові ресурси поділено на 3 частини) і підприємств також є три. Можливий приріст випуску продукції задається таблицею 2.2.

Таблиця 2.2. Показники приросту випуску продукції підприємств у залежності від додаткових фінансових ресурсів

Номери підприємств	Кількість часток додаткових фінансувань			
	0	1	2	3
№1	1	12	18	24
№2	2	8	16	28
№3	4	10	20	30

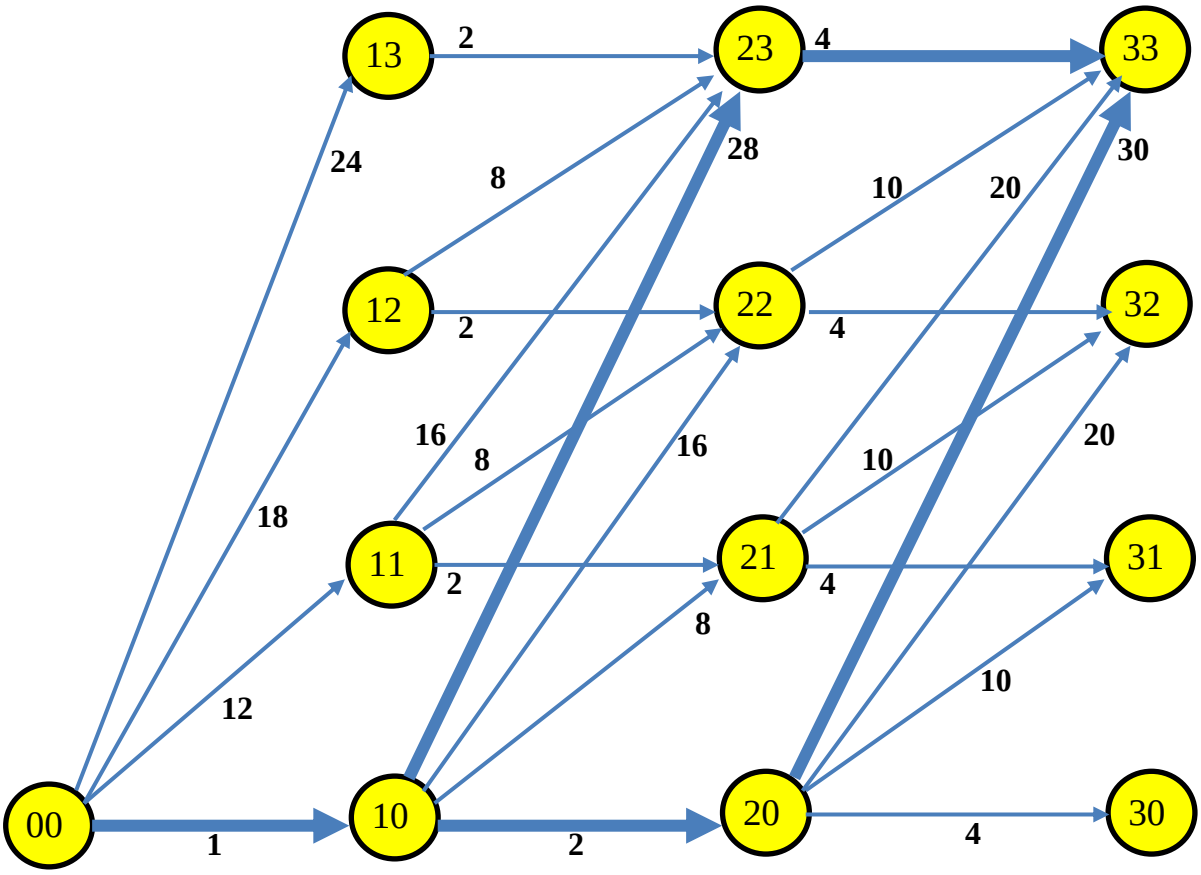


Рисунок 2.2. Граф-модель задачі про розподіл фінансових ресурсів

Локальні керування система для переходу зі стану s_{xy} будемо проводити за такими алгоритмами:

- $s_{xy} \rightarrow s_{x+1y}$ – означає, що підприємство № $x+1$ не одержало фінансів;
- $s_{xy} \rightarrow s_{x+1y+1}$ – означає, що підприємство № $x+1$ одержало одну частину фінансів;
- $s_{xy} \rightarrow s_{x+1y+2}$ – означає, що підприємство № $x+1$ одержало дві частини фінансів;
- $s_{xy} \rightarrow s_{x+1y+3}$ – означає, що підприємство № $x+1$ одержало три частини фінансів.

Оптимальний маршрут у цій задачі також шукаємо зворотнім шляхом. Оптимальних маршрути виявилося два. Вони виділені товстими лініями. Найбільше зростання приросту продукції буде, якщо всі фінанси віддати другому або третьому підприємстві.

Найбільший приріст випуску продукції в цих випадках буде становити 33 одиниці.

2.4. Задача оптимального розподілу ресурсів

Задача оптимального розподілу ресурсів досить схожа на задачу лінійного програмування про оптимальний план випуску продукції: підприємство для виготовлення одиниці j-го виду продукції витрачає a_{ij} одиниць i-го виду ресурсів, запаси якого становлять b_i одиниць а прибуток від виробництва та реалізації одиниці j-го виду продукції становить c_j. Потрібно встановити план випуску продукції, який забезпечує підприємству найбільший прибуток. Відмінність полягає в тому, що на відміну від звичайної постановки задачі виробництво різних видів продукції потрібно здійснювати послідовно (спочатку першу продукцію, потім другу і т.д.).

В такому формулюванні задача оптимального розподілу обмеженого ресурсу стає еквівалентною іншим оптимізаційним задачам,

зокрема, задачі про рюкзак чи задачі про завантаження автомобіля: *є декілька предметів різного об'єму чи ваги, кожен з яких має певну вартість. Потрібно в рюкзаку (або в автомобілі) обмеженого розміру чи вантажопідйомності розмістити предмети з максимальною сумарною вартістю.*

Нехай, наприклад, автомобіль (може бути будь яка ємкість для розміщення предметів – рюкзак, коробка, чемодан, контейнер чи транспортний засіб – літак, судно, тощо) місткістю n одиниць завантажується предметами m найменувань і при розміщенні одного i-го предмету об'ємом v_i прибуток становить c_i одиниць. Поточний стан системи s_{ij} будемо визначати двома параметрами i та j. Вони означають, що з перших i предметів сумарно завантажено j одиниць товару. Керування з вершини s_{ij} означає завантажити x_{i+1} предметів $(i + 1)$-го виду. При такому управлінні система переходить у стан s_{i+1u}, де $u=j + x_{i+1} \bullet v_{i+1}$.

Проілюструємо ці викладки на наступному прикладі. Нехай при n=4 маємо такі дані:

Таблиця 2.3. Дані про прибуток від розміщення одного предмету та його об'єм

Номер предмету (i)	1	2	3
Об'єм одного предмету (v_i)	2	3	1
Прибуток від розміщення одного предмету (c_i)	33	42	16

Модель задачі у вигляді графу показано на рисунку 2.3.

Перший предмет можна завантажити максимум два рази. Отже, $x_1 = 0$, 1 або 2. Так само $x_2 = 0$ або 1 і $x_3 = 0$, 1, 2, 3 або 4. На дугах вказано прибуток при завантаженні відповідної кількості предметів.

Послідовність індексів 10, 20, …, 30 з нулями на другому місці означає, що ні один з трьох предметів не завантажено.

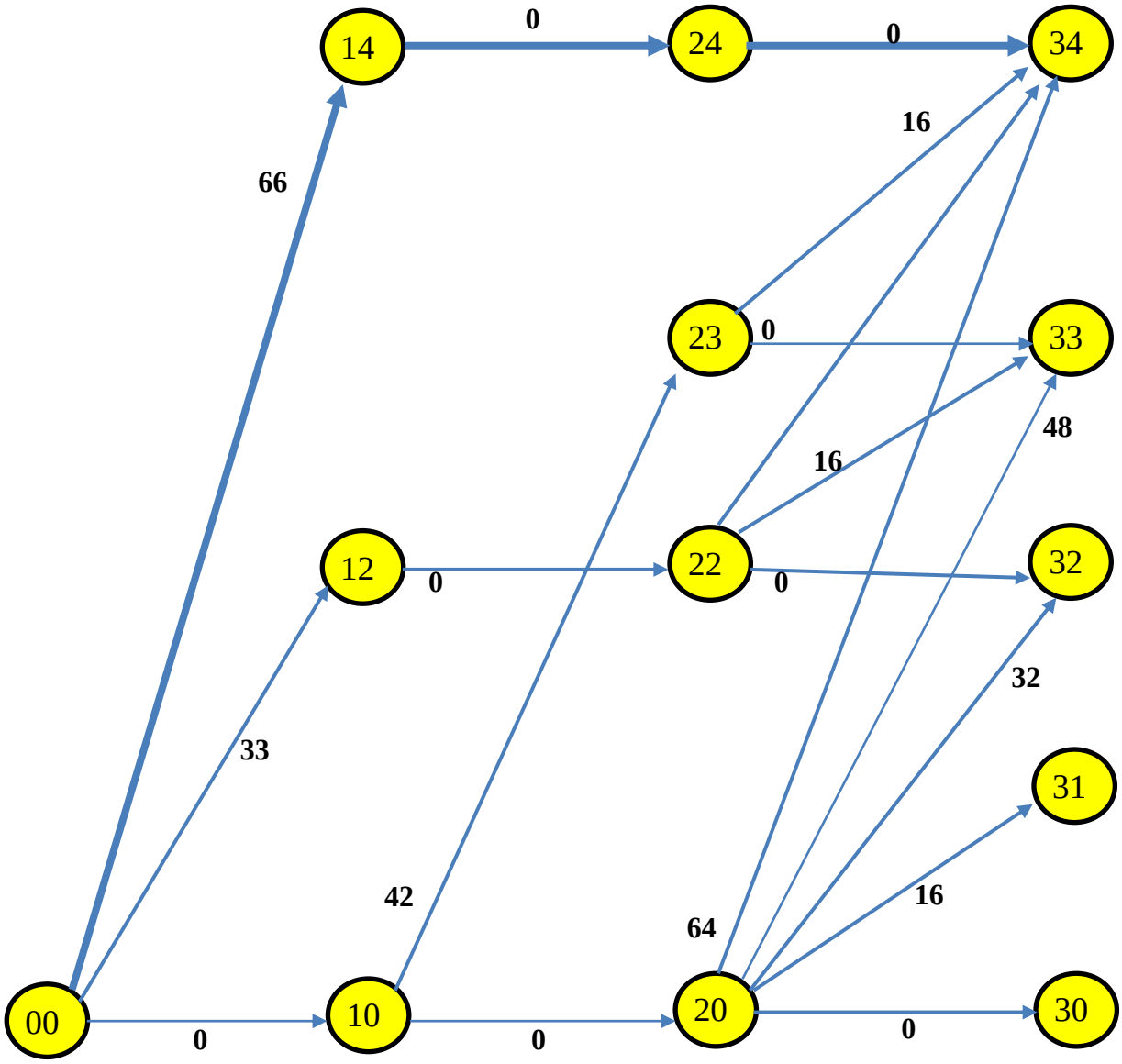


Рисунок 2.3. Граф-модель задачі про завантаження автомобіля

Для даного прикладу максимальну довжину становить маршрут 00, 14, 24, 34. Його позначено товстішою лінією. оптимальна стратегія полягає у завантаженні двох перших предметів, а прибуток становитиме 66 одиниць

Із збільшенням величини *n* тобто місткості автомобіля, граф-модель буде збільшуватися по вертикалі, а із збільшенням видів предметів – по

горизонталі. Зображення стає перевантаженим різними позначеннями. Тоді його візуальний розв’язок стає не очевидним. Побудувати граф-модель у вигляді вагової матриці і знайти оптимальний маршрут можна за допомогою очевидних рекурентних співвідношень.

Висновки до розділу 2

Динамічна система характеризується послідовністю змін її основних властивостей. Методом дискретизації через певні часові проміжки виділяють скінчену впорядковану послідовність станів системи і моделюють систему скінченим антициклічним орієнтованим ваговим графом з фіксованою стартовою вершиною і непорожньою множиною кінцевих вершин. Їх може бути декілька.

Вагова матриця служить основою розгляду оптимізаційних задач. Оптимальний маршрут вибирають зворотнім методом слідування від фінішних вершин до стартової.

Типовими прикладними динамічними задачами є задачі заміни обладнання, оптимального розподілу інвестицій, задача завантаження автомобіля і інші, де необхідно відшукати керування для оптимального переходу від стартового стану системи у фінішний

РОЗДІЛ 3 ВІДШУКАННЯ ОПТИМАЛЬНОГО МАРШРУТУ В ДИНАМІЧНІЙ ЗАДАЧІ КОМІВОЯЖЕРА МЕТОДОМ МОДИФІКОВАНОГО МУРАШИНОГО АЛГОРИТМУ

3.1 Мурашиний алгоритм в задачах логістики

Логістика є ключовим господарським процесом, що передбачає оптимізацію планування та контролю матеріальних і інформаційних потоків. Транспортна логістика, як її критично важливий компонент, безпосередньо пов'язана із завданням вибору оптимальних маршрутів, що є визначальним для мінімізації витрат і часу.

Саме транспорт є критично важливим, визначальним компонентом логістики, виконуючи роль забезпечення переміщення матеріальних потоків між різними ланками логістичного ланцюга. Цей критично важливий компонент логістики включає в себе не лише фізичне перевезення вантажів, але й поняття планування, організації та управління такими перевезеннями, а з оптимізаційної точки зору вибір оптимальних маршрутів і, можливо, вибір видів транспорту.

Ці моменти є ключовими при розробці математичної або як її іноді ще називають економіко-математичної моделі логістичної задачі.

Для формалізації логістичних бізнес-проблем широко застосовується математичне моделювання на основі теорії графів та мережевого аналізу. Ці інструменти дозволяють ефективно розв’язувати оптимізаційні задачі великої розмірності.

Для задач математичної економіки використання математичних методів відкриває широкі можливості для глибокого дослідження логістичних систем, дозволяє створювати такі моделі, які враховують складну взаємодію різного

роду складових у ланцюжку постачання, разом з тим, оптимізують методи розподілу ресурсів, забезпечують якісну оцінку ринкової рівноваги.

В розділі дослідження операцій задача оптимізації логістичних процесів розв’язується за допомогою різного роду інструментів. Одними із найпоширеніших виступають оптимізаційні моделі, що опираються на теорію графів та мережевого аналізу. Їх використовують при відшуканні найкращого рішення для управління логістичними мережами, які забезпечують побудову та оптимізацію маршрутів транспортних перевезень, забезпечуючи при цьому мінімізацію затрат або максимізацію ефективності логістичної задачі.

Одним із перспективних інструментів для розв'язання складних оптимізаційних задач є евристичні алгоритми, зокрема мурашиний алгоритм. Їх перевага полягає у його здатності ефективно оперувати із задачами великої розмірності.

В основі мурашиного алгоритму лежить імітація колективної поведінки мурах, що базується на механізмі стигмергії — непрямої комунікації через модифікацію навколишнього середовища, а саме через відкладення феромону на ребрах графа. Концентрація феромону визначає ймовірність вибору цього ребра іншими мурахами.

Не дивлячись на примітивність поведінки однієї мурахі, дія всієї колонії досить організована базуючись на багатоагентній системі, яка побудована на непрямому спілкуванні. Механізм такої непрямої взаємодії між мурахами базується на залишенні однією мурахою мітки у вигляді спеціальної хімічної речовини - феромону, яка стимулює подальшу активність інших мурах. Концентрація феромону на ділянці маршруту визначає перевагу руху саме по цій ділянці.

У даному розділі задача комівояжера обрана як ключова тестова задача. Мета дослідження полягає в аналізі ефективності мурашиного алгоритму як у статичній (класичній), так і в динамічній постановках.

3.2 Експериментальні дослідження

3.2.1 Аналіз експериментальних досліджень для стаціонарної задачі

Проведемо ряд обчислювальних експериментів з метою демонстрації роботи розробленого програмного продукту, формування технологій визначення оптимального маршруту в задачі комівояжера та виявлення особливих випадків, які можуть проявлятися в роботі алгоритму.

Мінімальний випадок при подорожі чотирьох мурах не є привабливим для таких досліджень.

Тому спочатку запустимо програму для подорожі п’яти мурах. Такий вибір обумовлено ще тим, що всього можна побудувати 4! різних циклічних гамільтонових маршрутів, а враховуючи, що половина з них буде відрізнятися лише зворотнім напрямком руху, то різних з них буде 12 маршрутів і їх легко побудувати та досліджувати без використання мого програмного продукту. Також зручно порівняти з роботою програмного продукту.

Запустивши програму для п’яти мурах, ми одержали таблицю (вагову матрицю), зображену на рисунку 3.1. Програма генерує випадковий набір конкретних числових значень.

Мураха 5. Поточний крок: 4

	V1	V2	V3	V4	V5
V1	x	18	52	50	1
V2	18	x	94	66	25
V3	52	94	x	38	31
V4	50	66	38	x	46
V5	1	25	31	46	x

Рисунок 3.1. Приклад вагової матриці для подорожі п’яти мурах.

За даними цієї таблиці легко визначити, що довжина маршруту

- 1→2→3→4→5→1 становить 197;

- 1→2→3→5→4→1 становить 239;
- 1→2→4→3→5→1 становить 124;
- 1→2→4→5→3→1 становить 241;
- 1→2→5→3→4→1 становить 162 ;
- 1→2→5→4→3→1 становить 159;
- 1→3→2→4→5→1 становить 259;
- 1→3→2→5→4→1 становить 267;
- 1→3→4→2→5→1 становить 182;
- 1→3→4→5→2→1 становить 224;
- 1→4→2→3→5→1 становить 242;
- 1→4→3→2→5→1 становить 208.

Оптимальним серед них є маршрут 1→2→4→3→5→1.

Відвідування ребер

	V1	V2	V3	V4	V5
V1	0	4	1	0	5
V2	4	0	0	5	1
V3	1	0	0	5	4
V4	0	5	5	0	0
V5	5	1	4	0	0

Рисунок 3.2. Приклад відвідування вершин мурахами

Відповідно до вагової матриці (див. рис. 3.1) програмним методом ми одержали таблицю (матрицю відвідувань), що подана на рисунку 3.2, за допомогою якої визначається оптимальний маршрут.

Для визначення найкращого маршруту мурашиним алгоритмом можна обирати будь-яку стартову вершину, адже ми маємо справу з циклічним шляхом. При старті з першої вершини (перший рядок) найбільше відвідувань було у вершину 5. З неї (п’ятий рядок) найбільше відвідувань ведуть у вершину 1, але ми її уже використовували, тому за рейтингом обираємо вершину 3. З третьої вершини (третій рядок) потрапляємо у вершину 4. З неї (рядок

четвертий) є два маршрути з однаковим рейтингом, але у вершині 3 ми уже побували. Тому обираємо вершину 2. Далі залишається завершити цикл у вершині 1. Отже, одержали маршрут оптимальний 1→5→3→4→2→1, який є зворотнім до маршруту 1→2→4→3→5→1 (ми одержали його вище методом всеможливих напрямків).

Цей циклічний шлях з стартом у різних вершинах в одному напрямку або в зворотному пройшли усі мурахи за виключенням першої (див. рис. 3.3), але це не вплинуло на визначення оптимального шляху.

Ускладнимо задачу до подорожі десяти мурах. Всього при такій розмірності задачі можна побудувати аж 9!/2 = 181440 різних маршрутів. Зрозуміло, що «вручну» їх ніхто досліджувати не буде.

Подорож мурах

	Крок 0	Крок 1	Крок 2	Крок 3	Крок 4	Крок 5
M1	V1	V5	V2	V4	V3	V1
▸ M2	V2	V1	V5	V3	V4	V2
M3	V3	V5	V1	V2	V4	V3
M4	V4	V3	V5	V1	V2	V4
• M5	V5	V1	V2	V4	V3	V5

Рисунок 3.3. Таблиця маршрутів, пройдених мурахами

Запустивши програму, одержимо, наприклад, наступну вагову матрицю (див. рис. 3.4).

Мураха 10. Поточний крок: 9

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
▸ V1	x	90	68	36	56	6	53	100	38	1
V2	90	x	86	58	99	65	44	51	91	47
V3	68	86	x	8	20	16	15	80	87	49
V4	36	58	8	x	94	55	75	33	96	97
V5	56	99	20	94	x	7	93	83	52	54
V6	6	65	16	55	7	x	39	17	70	19
V7	53	44	15	75	93	39	x	61	78	45
V8	100	51	80	33	83	17	61	x	32	21
V9	38	91	87	96	52	70	78	32	x	92
V10	1	47	49	97	54	19	45	21	92	x

Рисунок 3.4. Приклад вагової матриці для подорожі десяти мурах.

Програма зафіксувала наступні значення для матриці відвідування (рис. 3.5) та матриці подорожей (рис 3.6).

Відвідування ребер

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
V1	0	1	0	0	0	9	0	0	0	10
V2	1	0	0	5	3	0	5	0	4	2
V3	0	0	0	10	4	0	6	0	0	0
V4	0	5	10	0	0	0	0	5	0	0
V5	0	3	4	0	0	3	1	1	7	1
V6	9	0	0	0	3	0	2	5	0	1
V7	0	5	6	0	1	2	0	0	3	3
V8	0	0	0	5	1	5	0	0	6	3
V9	0	4	0	0	7	0	3	6	0	0
V10	10	2	0	0	1	1	3	3	0	0

Рисунок 3.5. Приклад відвідування вершин десятьма мурахами

Аналогічно до попереднього випадку, пошук оптимального маршруту розпочнемо з вершини 1. З таблиці відвідувань очевидним є шлях у вершину 10. З вершини 10 рівнозначними є два маршрути: у вершину 7 та 8. Отже, наша програма не дає однозначну відповідь.

Може бути три випадки:

1) обидва маршрути є оптимальними. Задача має декілька розв’язків;

2) один із маршрутів оптимальний, а інший близький до оптимального

3) маршрути не «погані», близькі до оптимального, але не є такими.

Подорож мурах

	Крок 0	Крок 1	Крок 2	Крок 3	Крок 4	Крок 5	Крок 6	Крок 7	Крок 8	Крок 9	Крок 10
M1	V1	V10	V6	V5	V3	V4	V8	V9	V7	V2	V1
M2	V2	V7	V3	V4	V8	V6	V1	V10	V5	V9	V2
M3	V3	V4	V8	V6	V1	V10	V7	V2	V9	V5	V3
M4	V4	V3	V7	V6	V1	V10	V8	V9	V5	V2	V4
M5	V5	V6	V1	V10	V8	V9	V7	V3	V4	V2	V5
M6	V6	V1	V10	V8	V9	V5	V3	V4	V2	V7	V6
M7	V7	V3	V4	V8	V6	V1	V10	V2	V9	V5	V7
M8	V8	V6	V1	V10	V7	V3	V4	V2	V9	V5	V8
M9	V9	V8	V6	V1	V10	V7	V3	V4	V2	V5	V9
M10	V10	V1	V6	V5	V3	V4	V8	V9	V7	V2	V10

Рисунок 3.6. Таблиця маршрутів, пройдених десятьма мурахами

Напрошується перевагу надати маршруту 1→10→8, бо в ваговій матриці перехід з вершини 10 у вершину 8 дешевший ніж у вершину 7. При цьому подальший маршрут визначається однозначно. В цьому випадку одержується цикл 1→10→8→9→5→3→4→2→7→6→1 і не важко підрахувати, що його вартість становить 281.

Якщо обрати перший маршрут 1→10→7, то рівноцінними можуть бути три маршрути:

1→10→7→3→4→2→9→5→6→8→1

1→10→7→3→4→8→9→5→2→6→1

1→10→7→3→4→8→9→5→6→2→1

Їх вартості становлять 394, 356 та 348 одиниць відповідно. Як бачимо із чотирьох маршрутів лише один підходить на роль оптимального, а інші близькі до оптимального.

Якщо стартувати із вершини 10, то однозначно визначається маршрут 10→1→6→8→9→5→3→4→2→7→10, вартість якого становить 283. Ми одержали п’ять маршрутів, які можна вважати близькими до оптимального, але жоден із них не є оптимальним, бо, наприклад, вартість маршруту 1→10→7→3→4→2→8→9→5→6→1 становить 275 одиниць.

Проведений аналіз підтверджує, що мурашиний алгоритм, як евристичний метод, ефективно відшуковує високоякісні (близькі до оптимальних) маршрути у статичній задачі комівояжера великої розмірності (10 вершин), коли повний перебір неможливий. Проте, як і більшість евристичних методів, ACO не гарантує знаходження глобального оптимуму, що було підтверджено у випадку неоднозначності вибору ребра (1→10→7 або 1→10→8), де кращий маршрут (275) було знайдено поза межами вибору з найбільшим рейтингом відвідувань. Це підкреслює необхідність модифікації алгоритму для підвищення точності, особливо в динамічних умовах.

3.2.2 Аналіз експериментальних досліджень для динамічної задачі

Тепер проведемо серію обчислювальних експериментів, якщо вагова матриця змінюється з кожним кроком алгоритму.

Спочатку розглянемо той же самий приклад із п’ятьма мурахами, що описано вище і порівняємо результати.

Нехай на кожному кроці три найменші значення зростають на 10 одиниць, а найбільші навпаки – зменшуються на 10. Тобто за повний цикл вагова матриця матиме наступні значення в такій послідовності (див. рисунок 3.7):

	V1	V2	V3	V4	V5
V1	x	18	52	50	1
V2	18	x	94	66	25
V3	52	94	x	38	31
V4	50	66	38	x	46
V5	1	25	31	46	x

а) вагова матриця на старті

	V1	V2	V3	V4	V5
V1	x	28	42	50	11
V2	28	x	84	56	35
V3	42	84	x	38	31
V4	50	56	38	x	46
V5	11	35	31	46	x

б) вагова матриця після першого кроку

	V1	V2	V3	V4	V5
V1	x	38	42	40	21
V2	38	x	74	46	35
V3	42	74	x	38	41
V4	40	46	38	x	46
V5	21	35	41	46	x

в) вагова матриця після другого кроку

	V1	V2	V3	V4	V5
V1	x	48	42	40	31
V2	48	x	64	36	45
V3	42	64	x	48	41
V4	40	36	48	x	36
V5	31	45	41	36	x

г) вагова матриця після третього кроку

	V1	V2	V3	V4	V5
V1	x	38	42	40	41
V2	38	x	54	46	35
V3	42	54	x	38	41
V4	40	46	38	x	46
V5	41	35	41	46	x

д) вагова матриця після четвертого кроку

Рисунок 3.7. Приклад значень вагової матриці у динамічній задачі

Запустивши програму одержимо матрицю відвідування вершин мурахами (див. рис. 3.8) та матрицю подорожей мурах (див. рис. 3.9).

	V1	V2	V3	V4	V5
V1	0	5	0	0	5
V2	5	0	0	5	0
V3	0	0	0	5	5
V4	0	5	5	0	0
V5	5	0	5	0	0

Рисунок 3.8. Матриця відвідування вершин мурахами в динамічній задачі

	Крок 0	Крок 1	Крок 2	Крок 3	Крок 4	Крок 5
M1	V1	V5	V3	V4	V2	V1
M2	V2	V1	V5	V3	V4	V2
M3	V3	V5	V1	V2	V4	V3
M4	V4	V3	V5	V1	V2	V4
M5	V5	V1	V2	V4	V3	V5

Рисунок 3.9. Матриця подорожей мурах в динамічній задачі

Проаналізувавши ці дві матриці, легко встановити, що усі п’ять мурах пройшли циклічний шлях одним і тим же маршрутом, який є для зазначених даних послідовності вагових матриць: 1→2→4→3→5→1. Кожна з них стартувала в іншій початковій точці циклічного маршруту і одні з них рухалися в одному напрямку, а інші – в протилежному.

Обчислимо затрати на ці цикли (довжини маршрутів) для кожної з мурах. Вони розподіляться наступним чином:

- перша мураха рухалась маршрутом 1→5→3→4→2→1. Його довжина становить 1 + 31 + 38 + 36 + 38 = 144 одиниці;
- друга мураха рухалась маршрутом 2→1→5→3→4→2. Його довжина становить 18 + 11 + 41 + 48 + 46 = 164 одиниці;
- третя мураха рухалась маршрутом 3→5→1→2→4→3. Його довжина становить 31 + 11 + 38 + 36 + 38 = 154 одиниці;
- четверта мураха рухалась маршрутом 4→3→5→1→2→4. Його довжина становить 38 + 31 + 21 + 48 + 46 = 184 одиниці;
- п’ята мураха рухалась маршрутом 5→1→2→4→3→5. Його довжина становить 1 + 28 + 46 + 48 + 41 = 164 одиниці.

Проведений експеримент дозволяє зробити цілий ряд висновків.

Перше, що варто зазначити, то в динамічній задачі, на відміну від стаціонарної, дії кожної мурахи не впливають на дії іншої. Привабливий напрямок руху на першому кроці може (в більшості випадків так і буде) не становити інтересу для інших мурах на інших кроках. Отже, тут доцільно використовувати специфіку мурашиного алгоритму для ланцюгових маршрутів.

Також цікаво, що для одного і того ж циклічного маршруту при старті з різних вершин ми одержали різні значення довжин. Отже, одержавши запропонованим мурашиним методом оптимальний циклічний маршрут, потрібно визначати і оптимальну стартову вершину та напрямок руху по ньому (в одну сторону чи протилежну).

Висновки до розділу 3

1. Аналіз АСО в статичній задачі: Проведене експериментальне дослідження Мурашиного алгоритму (АСО) на статичній задачі комівояжера підтвердило, що, як евристичний метод, він ефективно

знаходить близькі до оптимальних маршрути у задачах великої розмірності (10 вершин). Однак, виявлена проблема неоднозначності вибору та відсутність гарантії глобального оптимуму підкреслили необхідність його модифікації для роботи у більш складних умовах.

2. Проблема динамічної TSP: Встановлено, що класичний ACO не є достатньо ефективним для мереж зі змінною пропускною здатністю (динамічною вагою ребер), оскільки механізм оновлення феромонів є занадто інертним і повільно реагує на часові зміни.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці та дослідженню методу оптимізації маршрутів у мережах зі змінною пропускною здатністю на прикладі динамічної задачі комівояжера. У результаті проведеного дослідження були досягнуті поставлені мета та завдання.

З метою теоретичного обґрунтування та моделювання проведено аналіз існуючих графових моделей, який підтвердив актуальність теорії графів для розв'язання прикладних задач логістики та маршрутизації. Обґрунтовано, що для моделювання динамічної задачі комівояжера найбільш адекватним є використання скінченних повних (або умовно повних) орієнтованих вагових графів, які подаються за допомогою вагової матриці.

Проаналізовано методологію динамічного програмування на прикладі прикладних задач (заміна обладнання, розподіл ресурсів). Це забезпечило теоретичну базу для розуміння багатокрокових процесів прийняття рішень, що є ключовим для подальшої розробки адаптивного алгоритму.

Чітко сформульовано математичну модель динамічної задачі комівояжера, в якій ваги ребер графа (пропускна здатність, час або вартість) розглядаються як функції часу. Це стало основою для застосування евристичних методів, здатних швидко адаптуватися до змін у мережі.

Важливо. Проведено експериментальне дослідження Мурашиного алгоритму у статичній постановці, яке підтвердило його здатність знаходити високоякісні (близькі до оптимальних) маршрути, але виявило його обмежену ефективність для динамічних систем через інертність стандартного механізму оновлення феромонів.

Обчислювальні експерименти підтвердили, що в динамічній задачі для циклічного маршруту оптимальний результат залежить не лише від послідовності вершин, а й від оптимальної стартової вершини та напрямку

руху по циклу, що є важливим практичним висновком для планування маршрутів у реальних умовах.

Результати роботи мають високе практичне значення і можуть бути використані для підвищення ефективності планування логістичних та транспортних маршрутів, а також в управлінні динамічними комп’ютерними та комунікаційними мережами.

Таким чином, поставлену мету щодо розробки та дослідження методу оптимізації маршрутів у мережах зі змінною пропускною здатністю було повністю досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бартіш М. Я. Дослідження операцій. Ч. 2. Алгоритми оптимізації на графах, 2007. 168 с.

2. Бартіш М. Я. Методи оптимізації: Навчальний посібник. Львів: Видавничий центр ЛНУ імені Івана Франка, 2006. 120 с.

3. Глибовець М. М. Основи комп'ютерних алгоритмів. Київ: Вид. дім "КМ Академія", 2003. 450 с.

4. Данчук В. Д., Сватко В. В. Оптимізації пошуку шляхів по графу в динамічній задачі комівояжера методом модифікованого мурашиного алгоритму: Наукова стаття. Системні дослідження та інформаційні технології, 2012. № 2. С. 78–86.

5. Зайченко Ю. П. Дослідження операцій: Підручник. Київ: ЗАТ "ВІПОЛ", 2000. 912 с.

6. Катренко А. В. Дослідження операцій: Підручник. Львів: "Магнолія Плюс", 2004. 549 с.

7. Козаченко Д.М., Вернигора Р.В., Малашкін В.В. Основи дослідження операцій у транспортних системах: приклади та задачі: навчальний посібник для ВНЗ. Дніпропетровськ, 2015. 277 с.

8. Машина Н. І. Математичні методи в економіці: Навчальний посібник. Київ: Центр навчальної літератури, 2003. 148 с.

9. Мельник І. В. Теорія оптимізації та її застосування: Підручник. Харків: Фактор, 2010. 320 с.

10. Наконечний С. І., Савіна С. С. Математичне програмування: Навчальний посібник. Київ: КНЕУ, 2003. 452 с.

11. Нікольський Ю. В., Пасічник В. В., Щербина Ю. М. Дискретна математика: Підручник. Київ: Видавнича група BHV, 2007. 368 с.

12. Сеньо П. С. Випадкові процеси: Підручник. Львів: Компакт-ЛВ, 2006. 288 с.

ДОДАТКИ

Додаток А. Код, що реалізує функціональні можливості програми

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace Graphs2
{
   public partial class Form1 : Form
   {
      private List<PointF> vertexPositions = new List<PointF>();
      private float radius;
      private Random random = new Random();
      private int currentStep = 0;
      private int currentAnt = 0;
      private int vertices;
      private int maxDuga = 42;
      private int[,] visitCounts;
      private bool[] visited;
      private int[,] weightMatrix;

      public Form1()
      {
         InitializeComponent();
         maxDuga = 41;
```

```
    panelGraph.Paint += panelGraph_Paint;
    btnCreateMatrix.Enabled = true;
    btnNextStep.Enabled = false;
}
private void btnCreateMatrix_Click(object sender, EventArgs e)
{
    vertices = (int)numVertices.Value;
    if (vertices <= 0) return;
    visitCounts = new int[vertices, vertices];
    weightMatrix = new int[vertices, vertices];
    visited = new bool[vertices];
    visited[0] = true;
    CalculateVertexPositions(vertices);
    UpdateDataGridViews(vertices);
    panelGraph.Invalidate();
    currentAnt = 0;
    currentStep = 0;
    lblStepInfo.Text = $"Мураха {currentAnt + 1}. Поточний крок: 0";

    btnNextStep.Enabled = true;
    btnCreateMatrix.Enabled = true;
}
private void btnNextStep_Click(object sender, EventArgs e)
{
    if (vertices <= 1 || currentAnt >= vertices) return;
    int currentVertex = GetCurrentVertex();
    if (currentVertex == -1) return;
```

```
    int nextVertex = FindNextVertex(currentVertex);
    if (nextVertex != -1)
    {
        UpdatePathAndVisits(currentVertex, nextVertex);
        currentStep++;
        lblStepInfo.Text = $"Мураха {currentAnt + 1}. Поточний крок: {currentStep}";
    }
    else
    {
        MoveToNextAnt();
    }
}
private int GetCurrentVertex()
{
    if (currentStep < 0) return currentAnt;
    string cellValue = dataGridViewPath.Rows[currentAnt].Cells[currentStep].Value?.ToString();
    if (string.IsNullOrEmpty(cellValue)) return currentAnt;
    return int.Parse(cellValue.Substring(1)) - 1;
}
private int FindNextVertex(int currentVertex)
{
    int nextVertex = -1;
    int minWeight = int.MaxValue;
    for (int j = 0; j < vertices; j++)
    {
        if (!visited[j] && j != currentVertex)
        {
```

```
            int weight = weightMatrix[currentVertex, j];
            if (weight < minWeight)
            {
                minWeight = weight;
                nextVertex = j;
            }
        }
    }
    return nextVertex;
}
private void UpdatePathAndVisits(int currentVertex, int nextVertex)
{
    dataGridViewPath.Rows[currentAnt].Cells[currentStep + 1].Value = $"V{nextVertex + 1}";
    visitCounts[currentVertex, nextVertex]++;
    visitCounts[nextVertex, currentVertex]++;
    dataGridViewVisits[currentVertex, nextVertex].Value = visitCounts[currentVertex,
nextVertex];
    dataGridViewVisits[nextVertex, currentVertex].Value = visitCounts[nextVertex,
currentVertex];
    visited[nextVertex] = true;
}
private void MoveToNextAnt()
{
    if (currentAnt < vertices)
    {
        int lastVertex = GetCurrentVertex();
        if (lastVertex != -1 && lastVertex != currentAnt)
        {
```

```
            visitCounts[lastVertex, currentAnt]++;
            visitCounts[currentAnt, lastVertex]++;
            dataGridViewVisits[lastVertex, currentAnt].Value = visitCounts[lastVertex, currentAnt];
            dataGridViewVisits[currentAnt, lastVertex].Value = visitCounts[currentAnt, lastVertex];
        }
    }
    currentAnt++;
    currentStep = 0;
    if (currentAnt < vertices)
    {
        visited = new bool[vertices];
        visited[currentAnt] = true;
        lblStepInfo.Text = $"Мураха {currentAnt + 1}. Поточний крок: 0";
    }
    else
    {
        MessageBox.Show("Усі мурахи завершили свій шлях!");
        btnNextStep.Enabled = false;
    }
}
private void UpdateDataGridViews(int vertices)
{
    dataGridViewMatrix.ColumnCount = vertices;
    dataGridViewMatrix.RowCount = vertices;
    int totalCells = vertices * (vertices - 1) / 2;
    List<int> uniqueWeights = GenerateUniqueRandomNumbers(totalCells, 1, 100);
    int weightIndex = 0;
```

```
for (int i = 0; i < vertices; i++)
{
   dataGridViewMatrix.Columns[i].HeaderText = $"V{i + 1}";
   dataGridViewMatrix.Columns[i].Width = 50;
   dataGridViewMatrix.Rows[i].HeaderCell.Value = $"V{i + 1}";
   dataGridViewMatrix.Rows[i].Cells[i].Style.BackColor = Color.Gold;
   dataGridViewMatrix.Rows[i].Cells[i].Value = "x";
   for (int j = i + 1; j < vertices; j++)
   {
      int weight = uniqueWeights[weightIndex];
      dataGridViewMatrix.Rows[i].Cells[j].Value = weight;
      dataGridViewMatrix.Rows[j].Cells[i].Value = weight;
      weightMatrix[i, j] = weight;
      weightMatrix[j, i] = weight;
      weightIndex++;
   }
}
dataGridViewMatrix.Width = 68 + 50 * vertices;
dataGridViewMatrix.Height = 30 + 29 * vertices;
dataGridViewVisits.ColumnCount = vertices;
dataGridViewVisits.RowCount = vertices;
for (int i = 0; i < vertices; i++)
{
   dataGridViewVisits.Columns[i].HeaderText = $"V{i + 1}";
   dataGridViewVisits.Columns[i].Width = 50;
   dataGridViewVisits.Rows[i].HeaderCell.Value = $"V{i + 1}";
   dataGridViewVisits.Rows[i].Cells[i].Style.BackColor = Color.Gold;
```

```
            for (int j = 0; j < vertices; j++)
            {
                dataGridViewVisits[i, j].Value = visitCounts[i, j];
            }
        }
        dataGridViewVisits.Width = 68 + 50 * vertices;
        dataGridViewVisits.Height = 30 + 29 * vertices;
        dataGridViewPath.ColumnCount = vertices + 1;
        dataGridViewPath.RowCount = vertices;
        for (int i = 0; i < vertices; i++)
        {
            dataGridViewPath.Rows[i].HeaderCell.Value = $"M{i + 1}";
            dataGridViewPath.Rows[i].Cells[0].Value = $"V{i + 1}";
            dataGridViewPath.Rows[i].Cells[0].Style.BackColor = Color.Gold;
            for (int j = 0; j < vertices; j++)
            {
                dataGridViewPath.Columns[j].HeaderText = $"Крок {j}";
                dataGridViewPath.Columns[j].Width = 63;
            }
            dataGridViewPath.Columns[vertices].HeaderText = $"Крок {vertices}";
            dataGridViewPath.Columns[vertices].Width = 63;
            dataGridViewPath.Rows[i].Cells[vertices].Value = $"V{i + 1}";
            dataGridViewPath.Rows[i].Cells[vertices].Style.BackColor = Color.LightGreen;
        }
        dataGridViewPath.Width = 72 + 63 * (vertices + 1);
        dataGridViewPath.Height = 50 + 29 * vertices;
    }
```

```
    private List<int> GenerateUniqueRandomNumbers(int count, int minValue, int maxValue)
    {
        if (maxValue - minValue + 1 < count)
            throw new ArgumentException("Діапазон значень менший за кількість унікальних чисел, які потрібно згенерувати");
        HashSet<int> numbers = new HashSet<int>();
        while (numbers.Count < count)
            numbers.Add(random.Next(minValue, maxValue + 1));
        return numbers.ToList();
    }
    private void panelGraph_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        g.Clear(Color.White);
        for (int i = 0; i < vertices; i++)
        {
            for (int j = i + 1; j < vertices; j++)
            {
                g.DrawLine(Pens.Gray, vertexPositions[i], vertexPositions[j]);
            }
        }
        for (int i = 0; i < vertexPositions.Count; i++)
        {
            DrawVertex(g, vertexPositions[i], i + 1);
        }
    }
    private void DrawVertex(Graphics g, PointF position, int number)
    {
```

```
            float vertexSize = 30;
            float x = position.X - vertexSize / 2;
            float y = position.Y - vertexSize / 2;
            g.FillEllipse(Brushes.LightBlue, x, y, vertexSize, vertexSize);
            g.DrawEllipse(Pens.Black, x, y, vertexSize, vertexSize);
            string vertexText = number.ToString();
            Font font = new Font("Arial", 10);
            SizeF textSize = g.MeasureString(vertexText, font);
            g.DrawString(vertexText, font, Brushes.Black, position.X - textSize.Width / 2, position.Y -
textSize.Height / 2);
        }
        private void CalculateVertexPositions(int vertices)
        {
            vertexPositions.Clear();
            float centerX = panelGraph.Width / 2;
            float centerY = panelGraph.Height / 2;
            radius = (Math.Min(panelGraph.Width, panelGraph.Height) - 50) / 2;
            for (int i = 0; i < vertices; i++)
            {
                float angle = (float)(2 * Math.PI * i / vertices);
                float x = centerX + radius * (float)Math.Cos(angle);
                float y = centerY + radius * (float)Math.Sin(angle);
                vertexPositions.Add(new PointF(x, y));
```

Додаток Б. Код, що відповідає за відображення та взаємодію з користувачем

```
namespace Graphs2
{
   partial class Form1
   {
         private System.ComponentModel.IContainer components = null;
name="disposing
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }
      #region Windows Form Designer generated code
      private void InitializeComponent()
      {
         DataGridViewCellStyle dataGridViewCellStyle1 = new DataGridViewCellStyle();
         DataGridViewCellStyle dataGridViewCellStyle2 = new DataGridViewCellStyle();
         DataGridViewCellStyle dataGridViewCellStyle3 = new DataGridViewCellStyle();
         DataGridViewCellStyle dataGridViewCellStyle4 = new DataGridViewCellStyle();
         DataGridViewCellStyle dataGridViewCellStyle5 = new DataGridViewCellStyle();
         numVertices = new NumericUpDown();
         btnCreateMatrix = new Button();
```

```
dataGridViewMatrix = new DataGridView();
panelGraph = new Panel();
dataGridViewPath = new DataGridView();
dataGridViewVisits = new DataGridView();
label1 = new Label();
btnNextStep = new Button();
lblStepInfo = new Label();
label2 = new Label();
label3 = new Label();
((System.ComponentModel.ISupportInitialize)numVertices).BeginInit();
((System.ComponentModel.ISupportInitialize)dataGridViewMatrix).BeginInit();
((System.ComponentModel.ISupportInitialize)dataGridViewPath).BeginInit();
((System.ComponentModel.ISupportInitialize)dataGridViewVisits).BeginInit();
SuspendLayout();
 // numVertices
 numVertices.Location = new Point(259, 49);
numVertices.Margin = new Padding(3, 4, 3, 4);
numVertices.Maximum = new decimal(new int[] { 10, 0, 0, 0 });
numVertices.Minimum = new decimal(new int[] { 4, 0, 0, 0 });
numVertices.Name = "numVertices";
numVertices.Size = new Size(137, 27);
numVertices.TabIndex = 0;
numVertices.Value = new decimal(new int[] { 10, 0, 0, 0 });

   // btnCreateMatrix
       btnCreateMatrix.Location = new Point(414, 18);
btnCreateMatrix.Margin = new Padding(3, 4, 3, 4);
```

```
        btnCreateMatrix.Name = "btnCreateMatrix";
        btnCreateMatrix.Size = new Size(102, 65);
        btnCreateMatrix.TabIndex = 1;
        btnCreateMatrix.Text = "Створити таблицю";
        btnCreateMatrix.UseVisualStyleBackColor = true;
        btnCreateMatrix.Click += btnCreateMatrix_Click;

        // dataGridViewMatrix
         dataGridViewMatrix.AllowUserToAddRows = false;
        dataGridViewMatrix.AllowUserToDeleteRows = false;
        dataGridViewCellStyle1.Alignment = DataGridViewContentAlignment.TopLeft;
        dataGridViewCellStyle1.BackColor = SystemColors.Control;
        dataGridViewCellStyle1.Font = new Font("Segoe UI", 9F);
        dataGridViewCellStyle1.ForeColor = SystemColors.WindowText;
        dataGridViewCellStyle1.SelectionBackColor = SystemColors.Highlight;
        dataGridViewCellStyle1.SelectionForeColor = SystemColors.HighlightText;
        dataGridViewCellStyle1.WrapMode = DataGridViewTriState.True;
        dataGridViewMatrix.ColumnHeadersDefaultCellStyle = dataGridViewCellStyle1;
        dataGridViewMatrix.ColumnHeadersHeightSizeMode =
DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        dataGridViewMatrix.Location = new Point(3, 155);
        dataGridViewMatrix.Margin = new Padding(3, 4, 3, 4);
        dataGridViewMatrix.Name = "dataGridViewMatrix";
        dataGridViewMatrix.RightToLeft = RightToLeft.No;
        dataGridViewCellStyle2.Alignment = DataGridViewContentAlignment.MiddleLeft;
        dataGridViewCellStyle2.BackColor = SystemColors.Control;
        dataGridViewCellStyle2.Font = new Font("Segoe UI", 9F);
        dataGridViewCellStyle2.ForeColor = SystemColors.WindowText;
```

```
        dataGridViewCellStyle2.SelectionBackColor = SystemColors.Highlight;
        dataGridViewCellStyle2.SelectionForeColor = SystemColors.HighlightText;
        dataGridViewCellStyle2.WrapMode = DataGridViewTriState.True;
        dataGridViewMatrix.RowHeadersDefaultCellStyle = dataGridViewCellStyle2;
        dataGridViewMatrix.RowHeadersWidth = 65;
        dataGridViewCellStyle3.Alignment = DataGridViewContentAlignment.TopCenter;
        dataGridViewCellStyle3.WrapMode = DataGridViewTriState.True;
        dataGridViewMatrix.RowsDefaultCellStyle = dataGridViewCellStyle3;
        dataGridViewMatrix.Size = new Size(431, 331);
        dataGridViewMatrix.TabIndex = 2;

        // panelGraph
        panelGraph.Location = new Point(71, 561);
        panelGraph.Margin = new Padding(3, 4, 3, 4);
        panelGraph.Name = "panelGraph";
        panelGraph.Size = new Size(471, 243);
        panelGraph.TabIndex = 3;
        panelGraph.Paint += panelGraph_Paint;

        // dataGridViewPath
        dataGridViewPath.ColumnHeadersHeightSizeMode =
DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        dataGridViewPath.Location = new Point(1144, 155);
        dataGridViewPath.Margin = new Padding(3, 4, 3, 4);
        dataGridViewPath.Name = "dataGridViewPath";
        dataGridViewPath.RowHeadersWidth = 70;
        dataGridViewCellStyle4.Alignment = DataGridViewContentAlignment.TopCenter;
        dataGridViewPath.RowsDefaultCellStyle = dataGridViewCellStyle4;
```

```
        dataGridViewPath.Size = new Size(749, 331);
        dataGridViewPath.TabIndex = 4;

        // dataGridViewVisits
            dataGridViewVisits.ColumnHeadersHeightSizeMode =
DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        dataGridViewVisits.Location = new Point(574, 155);
        dataGridViewVisits.Margin = new Padding(3, 4, 3, 4);
        dataGridViewVisits.Name = "dataGridViewVisits";
        dataGridViewVisits.RightToLeft = RightToLeft.No;
        dataGridViewVisits.RowHeadersWidth = 65;
        dataGridViewCellStyle5.Alignment = DataGridViewContentAlignment.TopCenter;
        dataGridViewVisits.RowsDefaultCellStyle = dataGridViewCellStyle5;
        dataGridViewVisits.Size = new Size(393, 331);
        dataGridViewVisits.TabIndex = 5;

        // label1
        label1.AutoSize = true;
        label1.Location = new Point(71, 56);
        label1.Name = "label1";
        label1.Size = new Size(182, 20);
        label1.TabIndex = 6;
        label1.Text = "Введіть кількість вершин";
            // btnNextStep
            btnNextStep.Location = new Point(598, 13);
        btnNextStep.Margin = new Padding(3, 4, 3, 4);
        btnNextStep.Name = "btnNextStep";
        btnNextStep.Size = new Size(126, 65);
```

```
        btnNextStep.TabIndex = 7;
        btnNextStep.Text = "Наступний крок";
        btnNextStep.UseVisualStyleBackColor = true;
        btnNextStep.Click += btnNextStep_Click;
            // lblStepInfo

        lblStepInfo.AutoSize = true;
        lblStepInfo.Font = new Font("Times New Roman", 12F, FontStyle.Bold, GraphicsUnit.Point,
204);
        lblStepInfo.ForeColor = SystemColors.HotTrack;
        lblStepInfo.Location = new Point(31, 123);
        lblStepInfo.Name = "lblStepInfo";
        lblStepInfo.Size = new Size(155, 23);
        lblStepInfo.TabIndex = 8;
        lblStepInfo.Text = "Вагова матриця";

        // label2
            label2.AutoSize = true;
        label2.Font = new Font("Times New Roman", 12F, FontStyle.Bold, GraphicsUnit.Point, 204);
        label2.ForeColor = SystemColors.HotTrack;
        label2.Location = new Point(638, 123);
        label2.Name = "label2";
        label2.Size = new Size(185, 23);
        label2.TabIndex = 9;
        label2.Text = "Відвідування ребер";
            // label3
            label3.AutoSize = true;
        label3.Font = new Font("Times New Roman", 12F, FontStyle.Bold, GraphicsUnit.Point, 204);
```

```
label3.ForeColor = SystemColors.HotTrack;
label3.Location = new Point(1215, 123);
label3.Name = "label3";
label3.Size = new Size(148, 23);
label3.TabIndex = 10;
label3.Text = "Подорож мурах";
        // Form1
        AutoScaleDimensions = new SizeF(8F, 20F);
AutoScaleMode = AutoScaleMode.Font;
ClientSize = new Size(1924, 891);
Controls.Add(label3);
Controls.Add(label2);
Controls.Add(lblStepInfo);
Controls.Add(btnNextStep);
Controls.Add(label1);
Controls.Add(dataGridViewVisits);
Controls.Add(dataGridViewPath);
Controls.Add(panelGraph);
Controls.Add(dataGridViewMatrix);
Controls.Add(btnCreateMatrix);
Controls.Add(numVertices);
Margin = new Padding(3, 4, 3, 4);
Name = "Form1";
Text = "Подорож мурах";
((System.ComponentModel.ISupportInitialize)numVertices).EndInit();
((System.ComponentModel.ISupportInitialize)dataGridViewMatrix).EndInit();
((System.ComponentModel.ISupportInitialize)dataGridViewPath).EndInit();
```

```
            ((System.ComponentModel.ISupportInitialize)dataGridViewVisits).EndInit();
            ResumeLayout(false);
            PerformLayout();
        }
        #endregion
        private NumericUpDown numVertices;
        private Button btnCreateMatrix;
        private DataGridView dataGridViewMatrix;
        private Panel panelGraph;
        private DataGridView dataGridViewPath;
        private DataGridView dataGridViewVisits;
        private Label label1;
        private Button btnNextStep;
        private Label lblStepInfo;
        private Label label2;
        private Label label3;
    }
}
```