Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики, факультет інформатики

“Задача комівояжера як задача бінарного ЛП”

Курсова робота

Керівник курсової роботи:
Доцент, Кандидат фіз.-мат. наук
_____(підпис)
“____”_____ 2020 року
Щестюк Наталія Юріївна
Виконав студент 3-го року навчання
Галдецький Андрій Володимирович
113, Прикладна Математика

Київ 2020

План

Вступ

Задача комівояжера - одна з основних задач комбінаторної оптимізації для транспортних шляхів, яка не має більш ефективних аналогів для оптимізації та мінімізації транспортних шляхів і витрат. Вперше згадку задачі комівояжера можна зустріти як "*Icosian Game*" Вільяма Гамільтона в якій він шукав маршрут у графі з 20 вершинами. На сьогоднішній день існує багато методів дискретної оптимізації основаних на задачі комівояжера.

Метою даної курсової роботи є описання шляхів та методів пошуку найвигідніших маршрутів поставок товарів (продукції, сировини) по замкнутому маршруту за методом мурашиної колонії.

Науковим завданням даної курсової роботи є аналіз "задачі комівояжера" як задачі цілочисельно лінійного програмування.

Ця тема є дуже актуальна на сьогоднішній день, оскільки логістика у сучасній світовій економіці та глобальному світі є не менш важливою, ніж виробництво та продажі (реалізація) товарів та продукції.

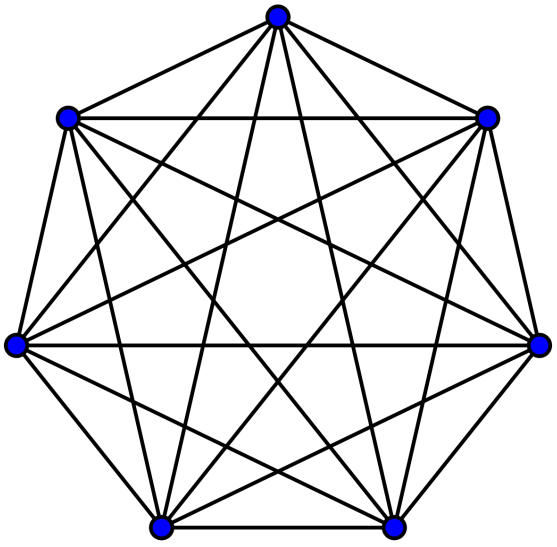
Особливого значення ця тема набуває також у сьогоднішніх умовах всесвітньої пандемії, яка призвела до гальмування світової економіки та руйнування існуючих логістичних схем. Тому, після завершення карантинних заходів, введених всіма країнами у зв'язку з пандемією, виникне гостра потреба у найскорішому відновленні логістичних схем та шляхів доставки сировини, товарів та продукції. Можливо, нинішня пандемія та бурхливе відновлення та зростання світової економіки після зняття карантинних

обмежень, дасть поштовх до активного вирішення задач з оптимізації транспортних шляхів (у тому числі мінімізації людського фактору та зменшення “живих” контактів між бізнесменами та між працівниками транспортно-логістичної галузі).

Також слід зазначити, що оптимізація та мінімізація транспортування товарів є актуальною в зв'язку з обмеженим обсягом світових запасів енергоресурсів та забрудненням атмосфери викидами від використаного палива. Як відомо, транспортування товарів (сировини, продукції) потребує майже стільки ж палива, як перевезення людей. Тому оптимізація та скорочення логістичних шляхів є також нагальною екологічною проблемою.

Постановка задачі

У 1930 році Вільям Гамільтон сформулював “задачу комівояжера”. Постановку задачі можна лаконічно викласти так : припустимо, що існує n міст. Відстань між будь якими двома містами i та j відома і становить Dij де $i = (x_i, y_i)$ та $j = (x_j, y_j)$. Якщо не може існувати прямого маршруту або $i = j$, то $Dij = \infty$. Зручно буде представити задачу у вигляді графу, у якого всі вершини зв’язані між собою.



Неорієнтований граф задачі комівояжера

Отже, можна поставити містам у відповідність вершини графа з малюнку, а дорогам які з’єднують ці міста - ребра цього графу. Отже, ми маємо знайти Гамільтонів цикл. Гамільтонів цикл - цикл, який проходить через усі вершини графу рівно один раз, а довжина циклу визначається як довжина всіх пройдених ребер. В графі n точок, отже наш цикл повинен проходити через всі інші вершини по одному разу - $(n-1)$ ребер, при переході з стану i у стан j $x_{ij} = 1$ - якщо цикл проходить через вершину,

$x_{ij} = 0$ якщо - ні. Отже існує $(n-1)!$ можливих циклів. За означенням відстань між містами симетрична $(Dij = Dji)$.

Подамо граф у вигляді матриці розміром $n \times n$. Кількість можливих шляхів у даному графі - $(n - 1)! = 6! = 720$.

Запишемо у вигляді таблиці(для 7 вершин/міст):

j\i	1	2	3	4	5	6	7
1	∞	Di_2j_1	Di_3j_1	Di_4j_1	Di_5j_1	Di_6j_1	Di_7j_1
2	Di_1j_2	∞	Di_3j_2	Di_4j_2	Di_5j_2	Di_6j_2	Di_7j_2
3	Di_1j_3	Di_2j_3	∞	Di_4j_3	Di_5j_3	Di_6j_3	Di_7j_3
4	Di_1j_4	Di_2j_4	Di_3j_4	∞	Di_5j_4	Di_6j_4	Di_7j_4
5	Di_1j_5	Di_2j_5	Di_3j_5	Di_4j_5	∞	Di_6j_5	Di_7j_5
6	Di_1j_6	Di_2j_6	Di_3j_6	Di_4j_6	Di_5j_6	∞	Di_7j_6
7	Di_1j_7	Di_2j_7	Di_3j_7	Di_4j_7	Di_5j_7	Di_6j_7	∞

Маршрут можна представити у вигляді циклу, де $x_{ij} = 1$ при переході з стану i у стан j, і $x_{ij} = 0$ в протилежному випадку. Довжина $L(R) = \sum_{j=1}^{n} \sum_{i=1}^{n} D_{ij} x_{ij} \rightarrow min, i = (1,n), j = (1,n)\ i \neq j$, де R - маршрут який проходить через усі вершини графу рівно по одному разу.

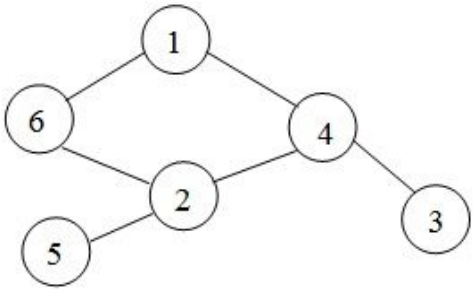
У подальшій роботі буде розглянуто алгоритм мурашиної колонії і його переваги та недоліки в цілому замість порівняння з конкретними алгоритмами вирішення “задачі комівояжера”.

Рішення задачі комівояжера алгоритмом мурашиної колонії(опис)

Алгоритм мурашиної колонії

Мурашиний алгоритм один з найбільш ефективних поліноміальних алгоритмів пошуку наближеного шляху. Алгоритм, який полягає в аналізі соціальної поведінки мурах під час їхнього пошуку ресурсів та їжі для колонії, вперше був запропонований Марком Доріго в 1992 році в своїй роботі “Optimization, Learning and Natural Algorithms”.

Нехай ми маємо деякий граф, в який ми запускаємо мурах для пошуку мінімального або приблизного до мінімального маршруту.



Неорієнтований граф

Припустимо, що вершина з якої мурахи починають шлях - (1), а вершина де вони мають закінчити шлях - (5). В природі, якщо подивитися на поведінку мурах, то можна помітити що вони майже завжди ходять однією лінією, це завдяки шляхам, які були прокладені їхніми попередниками.

Які ж механізми забезпечують настільки складну поведінку мурах? Відповідь на це питання дуже проста - мурахи під час пошуку шляху орієнтуються на феромонні сліди, які є в кожній з вершин і мають деяку

потужність. Під час кожного прогону кожна мураха залишає феромонний слід на своєму шляху.

У феромона є 3 обмеження. Перший - феромон може бути прокладений тільки мурахами, ніякий зовнішній фактор не має залишати феромонів на шляху. Другий - максимальне значення феромонів у вершині - це максимальне значення феромонів серед прилежних до неї ребер графу. Третій - феромони з часом випаровуються, в противному випадку мурахи можуть застрягти на якомусь з шляхів і не зможуть вийти з нього.

У мурахи є два варіанти поведінки. Перший - йти орієнтуючись на максимальний феромонний слід. Другий варіант - випадковим чином вибрати шлях, тобто досліджувати інші шляхи на графі. Кожна мураха може бачити рівно на крок вперед, тобто з вершини (1) мураха може бачити лише вершини (4) і (6) в заданому графі. Як правило перший прогін складається з другого типу поведінки мурах, для того, щоб заповнити вершини феромонами.

Яким чином мурахи прокладають феромони? Покладемо, що в нашому виході значення феромонів - 1, а початок шляху - 0. Кожний раз коли мураха проходить з однієї вершини в іншу вона буде брати значення феромона в цій вершині - P_0 і значення феромона в наступній вершині M з деяким коефіцієнтом згасання $C \in (0,1)$. Далі є два варіанти прокладання феромонів даною мурахою. Перший - беремо максимальне значення з даної і наступної з коефіцієнтом C, $Max(P_0, M \times C)$. Другий підхід - це взяти середнє значення між P_0 і M з деякою константою згасання $C \times M + (1 - C) \times P_0$.

Класичний алгоритм мурашиної колонії в задачі комівояжера

Як було зазначено вище, мурашиний алгоритм моделює багатоагентну систему, де кожна мураха сама вирішує як себе вести. Кожна мураха зберігає в собі деякий “список заборон”(англ. “tabu list”), тобто обираючи кожен наступний крок мураха знає де вона вже була і не повернеться в вершину в якій вона була. З кожним кроком до списку додається нова заборона.

Окрім списку заборон також є привабливість вершин(міст) для мурах. Привабливість залежить від феромонів і відстані між містами. Відстань є константною, а феромони оновлюються з кожним разом прогону алгоритму.

Отже, класичний алгоритм можна виразити так : якщо мураха знаходиться у вершині i, то вона обирає один з шляхів в залежності від свого “списку заборон”, феромонів та відстані між вершинами, припустимо що мураха обрала вершину j одна з доступних вершин для переходу, тобто не знаходиться в “списку заборон” і з'єднана з вершиною i, то можна подати відстань між вершинами як L_{ij}, а феромони у цій вершині як T_{ij}. Тоді ймовірність переходу з одного стану в інший буде:

$$p_{ij} = \frac{T_{ij}^{\alpha} \times \frac{1}{L_{ij}^{\beta}}}{\sum_{l \in S_i} (T_{ij}^{\alpha} \times \frac{1}{L_{ij}^{\beta}})}$$

де параметри α і β відповідають за регулювання важливості відстані і феромонів для мурах. Очевидно, що при $\alpha \to 0$ буде жадібний алгоритм, тобто вибір міста за відстанню, а при $\beta \to 0$ в нас буде швидкий ріст до одного приблизно оптимально рішення, і буде випадок коли мурахи не можуть вийти з одного шляху і покращити його. Від вибору відношення параметрів α і β залежить швидкість знаходження оптимальних шляхів у задачі.

Після проходження мурахою всього маршруту він залишає феромони зворотно пропорційні його пройденному маршруту по довжині, тобто :

$\Delta T_{ij} = (T_{ij} \times C) + \frac{k}{L}$ при $(ij) \in P$ або $\Delta T_{ij} = 0$ при $(ij) \notin P$, де C - коефіцієнт оновлення шляху феромонів, L - довжина пройденого шляху а параметр k - загальне значення феромонів на одну мураху. Тобто на (t + 1) ітерації алгоритму оновлення феромонів має вигляд схожий на другий підхід оновлення феромонів, $T_{ij}(t+1) = C \times T_{ij}(t) + \Delta T_{ij}$.

Іноді використовують мінімальне значення феромонів у вершині для оптимізації алгоритму, наприклад 0.1, для того щоб у агента (мурахи) був стимул переходити в непопулярні стани з мінімальною можливістю. Даний метод використовують в деяких модифікаціях алгоритму, де повинно бути мінімальне значення.

Також існує розбиття схоже на "острівну модель" генетичного алгоритму, тобо розбиття на менші колонії і обмін інформації між ними. Таке розбиття вважається значно ефективнішим за генетичне розбиття.[10]

Модифікації алгоритму мурашиної колонії для задачі комівояжера

ASelite(Elitist AS)

У 1996 році, в своїй роботі [1] М. Доріго, В. Маніезо, А. Колорні описав алгоритм "елітних мурах". Для модифікації і пришвидшення роботи алгоритму в "Елітній мурашиній системі" іноді вводять елітних мурах. Елітна мураха обирає найкращий маршрут знайдений з початку роботи алгоритму і посилює його феромонами. Тобто, суть стратегії полягає в штучному покращенні деяких шляхів. Кількість феромонів яку відкладає елітна мураха на маршрут рівне $\Delta T_{ij} = e \times k/L^{+}$, де L^{+} - довжина обраного маршруту, e - кількість елітних мураха k - загальне значення феромонів на одну мураху.

MAX-MIN AS

У 1997 Т. Штутцлє і Х. Хоос в своїй роботі [3] описали новий алгоритм мурашиної колонії. Даний алгоритм модифікований тим, що феромони зберігаються тільки на найкращих пройдених шляхах. Для уникнення тупикових субоптимальних рішень задачі використовується обмеження на максимальне і мінімальне значення феромонів на ребрах графу.

На етапі початку - всі ребра мають максимальне значення феромонів. З кожною ітерацією феромонний слід залишає лише мураха яка пройшла по найкращому маршруту. Іноді використовують елітних мурах для розповсюдження феромонів. Метод дозволяє провести більш ретельне дослідження області задачі і за меншу кількість кроків.

Ant-Q

У 1995 році М. Доріго, Л. Гамбарделла у роботі [2] описали алгоритму Ant-Q. Алгоритм названий в честь аналогії з методом Q-навчання у машинному навчанні. В Q-навчання також використовуються агенти для навчання системи, і в алгоритмі Ant-Q використовується система навчання з агентним підходом до вирішення задачі.

Алгоритм зберігає деяку Q-таблицю, яка зберігає величину корисності кожного переходу і ця таблиця змінюється з кожною ітерацією за відстанню шляхів в які було включено дане ребро.

ASrank

У 1997 році Б. Булнхаймер, Р. Хартл та К. Штраус у своїй роботі [5] запропонували рангову модифікацію для мурашиного алгоритму. В ранговому методі мурашиної колонії феромони розподіляються відповідно до довжини пройденого шляху. Кожна мураха записується в таблицю, відповідно до

довжини пройденого шляху і отримує відповідну кількість феромонів, яку вона відкладає на своєму шляху.

Для мурах які пройшли по найкоротшому шляху значення залишених феромонів буде мати вигляд $\Delta T_{ij} = W \times \frac{k}{L}$, де W - це коефіцієнт ранжирування по даній таблиці.

Для ретельного дослідження оптимальних шляхів використовують елітних мурах.

Ant Colony System

У 1997 році М. Доріго та Л. Гамбарделла у своїй роботі [4] запропонували модифікацію для мурашиного алгоритму. Даний алгоритм більш схожий до реального середовища, в ньому мурахи залишають феромони не після завершення ітерації, а при кожному кроці. Після кінця ітерації рівень феромонів на найкоротших шляхах збільшується, а на інших - зменшується. Також в алгоритмі змінена логіка пошуку шляху у мурахи - або вона обирає найкращий з деякою ймовірністю, в протилежному випадку - використовує стандартний алгоритм вибору шляху.

Переваги та недоліки використання мурашиного алгоритму

Переваги :

1. Для TSP(задачі комівояжера) ефективний за часом виконання метод.
2. Для великої кількості вершин експоненційний час збіжності до оптимального шляху.
3. Спирається на пам’ять попередників.

Недоліки :

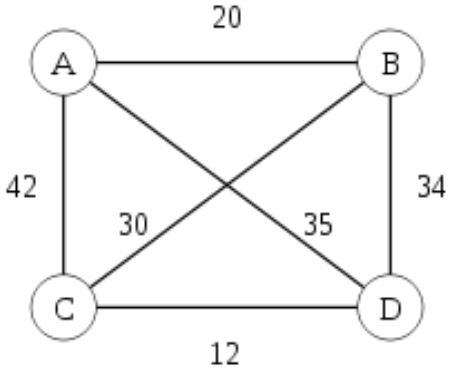
1. Через випадковий вибір шляху алгоритм не завжди дає вірний результат, також можливі відхилення.
2. Теоретичний аналіз алгоритму ускладнений через випадковість пошуку шляху.
3. Алгоритм сильно залежить від обраних параметрів.
    - Параметри обираються експериментальним шляхом.

Рішення задачі комівояжера алгоритмом мурашиної колонії (приклади)

Розглянемо більш конкретний приклад задачі комівояжера з використанням класичного мурашиного алгоритму.

Постанова задачі

Задано неорієнтований граф. Знайти довжину шляху найкоротшого маршруту знайденого за мурашиним алгоритмом за 3 ітерації в задачі комівояжера. За критерій оцінки узяти відстань між вершинами. Початковий стан - A.



1. Запишемо заданий граф у вигляді таблиці :

табл. 1

j\i	A	B	C	D
A	∞	20	42	35
B	20	∞	30	34
C	42	30	∞	12

D	35	34	12	∞

* знак ∞ на діагоналі означає неможливість перейти з стану в цей самий стан.

2. Знайдемо ймовірності переходу для мурах у графі. Випадкове число від 0 до 1.

   $P_1 = 0.7879$

   $P_2 = 0.1836$

   $P_3 = 0.4732$

3. Знайдемо початкові феромони, найкращим чином буде генерувати випадкову кількість для початкового графу. В подальших ітераціях це значення має змінитися відповідно до поведінки мурах.

   $T_{AB} = 3$

   $T_{AC} = 2$

   $T_{AD} = 3$

   $T_{BC} = 1$

   $T_{BD} = 2$

   $T_{CD} = 1$

4. Представимо данні у вигляді таблиці :

табл. 2

Ребро	Довжина	Феромони
A-B	20	3
A-C	42	2
A-D	35	3

B-C	30	1
B-D	34	2
C-D	12	1

Рішення задачі:

Перша ітерація.

Ймовірність переходу мурахи рахується за формулою :

$p_{ij} = \frac{T_{ij}^{\alpha} + \frac{1}{L_{ij}^{\beta}}}{\sum_{l \in S_i}(T_{ij}^{\alpha} + \frac{1}{L_{ij}^{\beta}})}$, де T_{ij} - феромони, L_{ij} - відстань між двома вершинами, а α і β - деякі коефіцієнти регулювання важливості факторів феромонів та відстані.

1. Візьмемо коефіцієнти $\alpha = 0.6$, $\beta = 0.4$ і кількість феромонів на мураху $k = 50$, коефіцієнт $C = 0.9$ та обрахуємо ймовірності переходу в інші стани графу.

$$\sum_{l \in S_i}(T_{iA}^{0.6} + \frac{1}{L_{iA}^{0.4}}) = 1.38941$$

$$P_{AB} = \frac{3^{0.6} \times 1/20^{0.4}}{1.38941} \approx 0.419788$$

$$P_{AC} = \frac{2^{0.6} \times 1/42^{0.4}}{1.38941} \approx 0.244618$$

$$P_{AD} = \frac{3^{0.6} \times 1/35^{0.4}}{1.38941} \approx 0.335594$$

2. Обрахуємо границі секторів переходу :

$p_{AB} = 0.419788$

$p_{AC} = 0.664406$

$p_{AD} = 1$

3. Перейдемо в наступний стан

   У першої мурахи випадкове число рівне 0.7879, отже вона переходить у вершину C.

4. З вершини C залишилося лише 2 можливих переходи - у B і D. Пройдена вершина потрапила до списку заборон.

   $\sum_{l \in S_i} (T_{iC}^{0.6} + \frac{1}{L_{iC}^{0.4}}) = 0.626645$

   $P_{CB} = \frac{1^{0.6} \times 1/30^{0.4}}{4.49301} \approx 0.409383$

   $P_{CD} = \frac{1^{0.6} \times 1/12^{0.4}}{4.49301} \approx 0.590617$

5. Обрахуємо границі секторів переходу для C :

   $p_{CB} = 0.409383$

   $p_{CD} = 1$

6. Перейдемо в наступний стан

   Число рівне 0.7879, отже переходимо до стану D.

7. З вершини D залишається лише один шлях, оскільки всі інші вже потрапили до списку заборон.

Перша мураха пройшла весь маршрут і він - A→C→D→B→A. Загальна довжина пройденого маршруту за першу ітерацію рівна:

$$L_{AC} + L_{CD} + L_{DB} + L_{BA} = 42 + 12 + 34 + 35 = 123$$

8. Оновити феромони у таблиці :

Після проходження мурахою маршруту його феромони розподіляються за формулою : $\Delta T_{ij} = (T_{ij} \times C) + \frac{k}{L}$ на всіх вершинах, де L - шлях пройдений мурахою, T_{ij} - кількість феромонів на ребрі, C - коефіцієнт оновлення шляху а k - це загальна кількість феромонів на мураху.

$$T_{AB} = (3 \times 0.9) + \frac{0}{123} = 2.7$$

$$T_{AC} = (2 \times 0.9) + \frac{50}{123} = 2.2$$

$$T_{AD} = (3 \times 0.9) + \frac{0}{123} = 2.7$$

$$T_{BC} = (1 \times 0.9) + \frac{0}{123} = 0.9$$

$$T_{BD} = (2 \times 0.9) + \frac{50}{123} = 2.2$$

$$T_{CD} = (1 \times 0.9) + \frac{50}{123} = 1.3$$

табл. 3

Ребро	Довжина	Феромони
A-B	20	2.7
A-C	42	2.2
A-D	35	2.7
B-C	30	0.9
B-D	34	2.2
C-D	12	1.3

Друга ітерація

1. Ймовірність переходу у стани графу зі стану A :

$$\sum_{l \in S_i} (T_{iA}^{0.6} + \frac{1}{L_{iA}^{0.4}}) = 1.34512$$

$$P_{AB} = \frac{2.7^{0.6} \times 1/20^{0.4}}{1.38941} \approx 0.407048$$

$$P_{AC} = \frac{2.2^{0.6} \times 1/42^{0.4}}{1.38941} \approx 0.267543$$

$$P_{AD} = \frac{2.7^{0.6} \times 1/35^{0.4}}{1.38941} \approx 0.325409$$

2. Границі переходів другої ітерації :

$$p_{AB} = 0.407048$$

$$p_{AC} = 0.674591$$

$$p_{AD} = 1$$

3. Перехід у наступний стан :

   У другої мурахи випадкове число рівне 0.1836, отже вона переходить у стан B.

4. Ймовірність переходу з вершини B в стани C і D. Стан A потрапляє до списку заборон.

$$\sum_{l \in S_i} (T_{iB}^{0.6} \times \frac{1}{L_{iB}^{0.4}}) = 0.632440$$

$$P_{BC} = \frac{1^{0.6} \times 1/30^{0.4}}{4.49301} \approx 0.380783$$

$$P_{BD} = \frac{1^{0.6} \times 1/12^{0.4}}{4.49301} \approx 0.619217$$

5. Границі секторів переходу для B :

$$p_{CB} = 0.380783$$

$$p_{CD} = 1$$

6. Перехід мурахи у наступний стан :

   $0 < 0.1836 < 0.380783$, отже мураха переходить у стан C.

7. З вершини C залишається лише один шлях, а саме до вершини D.

   Друга мураха пройшла весь маршрут і він - A→B→C→D→A. Загальна довжина пройденого маршруту за першу ітерацію рівна:

   $L_{AB} + L_{BC} + L_{CD} + L_{DA} = 20 + 30 + 12 + 35 = 97$

8. Оновити феромони у таблиці :

   $T_{AB} = (2.7 \times 0.9) + \frac{50}{97} = 2.96$

   $T_{AC} = (2.2 \times 0.9) + \frac{0}{97} = 1.98$

   $T_{AD} = (2.7 \times 0.9) + \frac{50}{97} = 2.96$

   $T_{BC} = (0.9 \times 0.9) + \frac{50}{97} = 1.33$

   $T_{BD} = (2.2 \times 0.9) + \frac{0}{97} = 1.98$

   $T_{CD} = (1.3 \times 0.9) + \frac{50}{97} = 1.69$

табл. 4

Ребро	Довжина	Феромони
A-B	20	2.96
A-C	42	1.98
A-D	35	2.96
B-C	30	1.34
B-D	34	1.98
C-D	12	1.7

Третя ітерація

1. Ймовірність переходу у стани графу зі стану A :

$$\sum_{l \in S_i}(T_{iA}^{0.6} \times \frac{1}{L_{iA}^{0.4}}) = 6.10909$$

$$P_{AB} = \frac{2.96^{0.6} \times 1/20^{0.4}}{6.10909} \approx 0.363292$$

$$P_{AC} = \frac{1.98^{0.6} \times 1/42^{0.4}}{6.10909} \approx 0.283321$$

$$P_{AD} = \frac{2.96^{0.6} \times 1/35^{0.4}}{6.10909} \approx 0.353387$$

2. Границі переходів третьої ітерації зі стану A :

$$p_{AB} = 0.363292$$

$$p_{AC} = 0.646613$$

$$p_{AD} = 1$$

3. Перехід у наступний стан :

   У третьої мурахи випадкове число 0.4732, отже вона переходить у C.

4. Ймовірність переходу зі стану C в інші(за виключенням A у списку задорон) :

$$\sum_{l \in S_i}(T_{iC}^{0.6} \times \frac{1}{L_{iC}^{0.4}}) = 3.19350$$

$$P_{CB} = \frac{1^{0.6} \times 1/30^{0.4}}{4.49301} \approx 0.453578$$

$$P_{CD} = \frac{1^{0.6} \times 1/12^{0.4}}{4.49301} \approx 0.546422$$

5. Границі секторів переходу :

$$p_{CB} = 0.453578$$

$$p_{CD} = 1$$

6. Перехід у наступний стан :

   $0.453578 < 0.4732 < 1$, отже мураха переходить у стан D.

7. З вершини D залишається лише перехід у B, отже мураха пройшла шлях A→C→D→B→A. Загальна довжина пройденого маршруту за першу ітерацію рівна: $L_{AC} + L_{CD} + L_{DB} + L_{BA} = 42 + 12 + 34 + 35 = 123$

8. Оновимо таблицю феромонів :

   $T_{AB} = (2.96 \times 0.9) + \frac{0}{123} = 2.66$

   $T_{AC} = (1.98 \times 0.9) + \frac{50}{123} = 2.18$

   $T_{AD} = (2.96 \times 0.9) + \frac{0}{123} = 2.66$

   $T_{BC} = (1.34 \times 0.9) + \frac{0}{123} = 1.2$

   $T_{BD} = (1.98 \times 0.9) + \frac{50}{123} = 2.18$

   $T_{CD} = (1.7 \times 0.9) + \frac{50}{123} = 1.93$

табл. 4

Ребро	Довжина	Феромони
A-B	20	2.66
A-C	42	2.18
A-D	35	2.66
B-C	30	1.2
B-D	34	2.18
C-D	12	1.93

З даної задачі видно ряд недоліків цього алгоритму, наприклад кількість феромонів на мураху $k = 50$ та коефіцієнт важливості феромонів $\alpha = 0.6$ не зовсім підходить для даної задачі, бо при коефіцієнті - 50 у задачі занадто слабке посилення феромонів і при коефіцієнті важливості 0.6 занадто слабкий вплив вже існуючих феромонних слідів на мурах, але це можна зрозуміти лише експериментальним шляхом.

Висновок

Таким чином тут був проаналізований та запрограмований один з методів пошуку найвигіднішого шляху поставок та транспортування товарів - метод мурашиної колонії, який доказав ефективність свого використання для “задачі комівояжера”.

В сучасній світовій економіці необхідність швидко і ефективно налагоджувати логістику і транспортну систему є не менш важливою проблемою ніж виробництво та продаж товарів та продукції.

Особливо проблема “задачі комівояжера” актуальна у сьогоднішній ситуації коли потрібно змінювати деякі транспортні шляхи у зв'язку з всесвітньою пандемією яка призвела до гальмування світової економіки. Тому після її завершення є потреба в швидкому та ефективному налагодженні транспортних шляхів та логістичних схем.

Список літератури

1. M. Dorigo, V. Maniezzo, A. Colorni, “The Ant System: Optimization by a colony of cooperating agents” / Man, and Cybernetics-Part B, стор. 29-41
2. L. M. Gambardella, M. Dorigo, “Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem” / [журнал] Morgan Kaufmann 1995 p.
3. T. Stützle, H. Hoos, “MAX-MIN Ant System and local search for the traveling salesman problem” / International Conference on Evolutionary Computation, 1997 p.
4. M. Dorigo, S. Member and L. M. Gambardella “Ant colony system: a cooperative learning approach to the TSP” / TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1, APRIL 1997 http://people.idsia.ch/~luca/acs-ec97.pdf
5. Bernd Bullnheimer, Richard F. Hartl, Christine Strauß, “A new rank based version of the Ant System. A computational study” / Adaptive Information Systems and Modelling in Economics and Management Science, 1997 p.
6. Штовба С. Д. “Муравьиные алгоритмы” // [журнал] Exponenta Pro. 2003. № 4. С.70-75 http://www.serhiy-shtovba.narod.ru/doc/Shtovba_Ant_Algorithms_ExponentaPro_2003_3.pdf
7. M. Dorigo, D. Corde, F. Glover “New Ideas in Optimization” McGrav-Hill 1999 p.
8. M. Dorigo, “Optimization, Learning and Natural Algorithms”, [дисертація] 1992p.
9. Ашманов С.А. “Линейное программирование”. М.:Наука, 1981.
10. Ершов Н.М. “Муравьиные алгоритмы” [Лекція] 2011 р. http://hpc-education.ru/files/lectures/2011/ershov/ershov_2011_lectures10.pdf
11. А. Сухарев, А.Тимохов,В. Федоров “Курс Методов Оптимизации”,1986.

Додаток. Реалізація на мові програмування C#

Реалізацію методу як консольного застосування на C#.

Відстань вводяться з файлу matrix.txt(bin\Debug\netcoreapp\matrix.txt) або заповнюються випадковим чином r.Next(5, 50) від 5 до 50. aPhCoef і bDistCoef - коефіцієнти важливості феромонів та дистанції, antPh - кількість феромонів на мураху , colonySize - кількість мурах у колонії

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace TravelingSalesmanProblem
{
   class ACSystemTSP
   {
      public static Random r = new Random();
      public static int colonySize = 4;
      public static int matrixSize = 10;
      public static int aPhCoef = 3;
      public static int bDistCoef = 2;
      public static double antPh = 40;
      public static double dcrsPh = 0.9;
      //public static int[][] distance;
      //public static double[][] ph;

      public static void Main(string[] args)
      {
         //matrix from file
         /*
```

```
         *    0  i2j1 i3j1 ... inj1
         * i1j2     0 i3j2 ... inj2
         * i1j3  i2j3    0 ... inj3
         * ...    ...  ... ...  ...
         * i1jn  i2jn  ... ...    0
         */
        /*-------------------------------------------------------------------------------
        string[] str = System.IO.File.ReadAllLines("matrix.txt");
        int[][] distance = new int[str.Length][];
        matrixSize = str.Length;
        for (int i = 0; i < str.Length; i++)
        {
           //Split separate by ' '
           //Linq select to int
           distance[i] = str[i].Split(default(string[]),
StringSplitOptions.RemoveEmptyEntries).Select(x => int.Parse(x)).ToArray<int>();
           Console.WriteLine(distance[i][0]);
        }
       -------------------------------------------------------------------------------*/



        //generate massive of distance for matrix from 5 to 50
        int[][] distance = new int[matrixSize][];
        for (int i = 0; i < distance.Length; i++)
        {
           distance[i] = new int[matrixSize];
        }
        for (int i = 0; i < distance.Length; i++)
        {
           for (int j = i + 1; j < distance.Length; j++)
           {
```

```
            int dist = r.Next(5, 50);
            distance[i][j] = dist;
            distance[j][i] = dist;
        }
    }

    double result = int.MaxValue;

    //generate massive of ways
    int[][] antsWay = new int[colonySize][];
    for (int i = 0; i < colonySize; i++)
    {
        antsWay[i] = CreateWay(matrixSize);
    }

    //generate massive of pheromon
    double[][] ph = new double[matrixSize][];
    for (int i = 0; i < ph.Length; i++)
    {
        ph[i] = new double[ph.Length];
    }

    for (int i = 0; i < 1000; i++)
    {
        Steps(ph, antsWay, distance);
        Iteration(ph, antsWay, distance);

        int[] newWay = BestWay(antsWay, distance);
        double newLenght = CurrentWay(newWay, distance);
        //match
        if (newLenght < result)
        {
```

```
            result = newLenght;
            Console.WriteLine("----------------------------------------");
            Console.WriteLine("Current best " + result);
        }
    }
    Console.WriteLine("----------------------------------------");
    Console.WriteLine("Best way ants found " + result);
    Console.WriteLine("----------------------------------------");
    Console.WriteLine("Elite ph way " + PheromonWay(ph, distance));
    Console.WriteLine("----------------------------------------");


}

//create massive of ways
public static int[] CreateWay(int matrixSize)
{
    int[] way = new int[matrixSize - 1];
    int[] blacklist = new int[matrixSize - 1];

    for (int i = 1; i < way.Length; i++)
    {
        way[i] = i;
    }

    for (int i = 0; i < way.Length; i++)
    {
        int step = r.Next(i, way.Length);
        bool isPassed = false;
        for (int j = 0; j < blacklist.Length; j++)
        {
            if(blacklist[i] == step)
```

```
            {
                isPassed = true;
                //i--;
            }
        }
        if (isPassed == false)
        {
            way[i] = step;
            blacklist[i] = step;
        }
        else
        {
            i--;
        }
    }

    int res = 0;
    for (int i = 0; i < way.Length; i++)
    {
        if (way[i] == 1)
            res = i;
    }
    int first = way[0];
    way[0] = way[res];
    way[res] = first;

    return way;
}

//sum of the passed array elements
public static double CurrentWay(int[] way, int[][] distance)
{
```

```
    double res = 0.0;
    for (int i = 0; i < way.Length - 1; i++)
    {
        res += distance[way[i]][way[i + 1]];
    }
    return res;
}

public static void Steps(double[][] ph, int[][] antsWay, int[][] distance)
{
    int matrixSize = ph.Length;
    for (int i = 0; i <= antsWay.Length - 1; i++)
    {
        int start = r.Next(1, matrixSize);
        int[] way = CreateWay(ph, distance, start);
        antsWay[i] = way;
    }
}

public static int[] CreateWay(double[][] ph, int[][] distance, int s)
{
    int size = ph.Length;
    int[] ways = new int[size];
    ways[0] = s;
    bool[] blackList = new bool[size];
    blackList[s] = true;
    for (int i = 0; i < size - 1; i++)
    {
        int currentStep = ways[i];
        //analyze pheromones, lenght and select step
        int next = Step(currentStep, blackList, ph, distance);
        ways[i + 1] = next;
```

```
        //blacklist for ants
        blackList[next] = true;
    }
    return ways;
}

public static int Step(int currentStep, bool[] blackList, double[][] ph, int[][] distance)
{
    double[] chances = Chances(blackList, ph, distance, currentStep);

    double[] stepChance = new double[chances.Length + 1];
    for (int i = 0; i <= chances.Length - 1; i++)
    {
        //path selection chances for elements
        stepChance[i + 1] = stepChance[i] + chances[i];
    }

    //chose action
    double antStep = r.NextDouble();

    for (int i = 0; i <= stepChance.Length - 2; i++)
    {
        if (antStep >= stepChance[i] && antStep < stepChance[i + 1])
        {
            return i;
        }
    }
    return 0;
}

public static double[] Chances(bool[] blackList, double[][] ph, int[][] distance, int currStep)
{
```

```
    double[] elemPh = new double[matrixSize];
    double allChances = 0.0;//all from 0.0 to 1.0
    for (int i = 0; i < elemPh.Length; i++)
    {
        if (i != currStep && blackList[i] == false)
        {
            double dist = distance[currStep][i];
            //select way t^a * 1/h^b
            elemPh[i] = Math.Pow(ph[currStep][i], aPhCoef) * Math.Pow(dist, bDistCoef);

            //if ph < 0 that p*coef -> negative inf, cant be 0
            if (elemPh[i] < 0.01)
                elemPh[i] = 0.01;
        }
        else elemPh[i] = 0.0;
        allChances += elemPh[i];
    }

    double[] chances = new double[matrixSize];
    for (int i = 0; i <= chances.Length - 1; i++)
        chances[i] = elemPh[i] / allChances;
    return chances;
}

//update current iteration
public static void Iteration(double[][] ph, int[][] visits, int[][] dists)
{
    //is element [i][j] visited for update
    Boolean[][] way = new Boolean[matrixSize][];
    for (int i = 0; i < way.Length; i++)
    {
        way[i] = new Boolean[matrixSize];
```

```
}
for (int i = 0; i < way.Length; i++)
{
   for (int j = i + 1; j < way.Length; j++)
   {
      for (int s = 0; s < visits.Length; s++)
      {
         if (visits[s][j] == j - 1)
         {
            way[s][j] = true;
         }
         else
         {
            way[i][j] = false;
         }

      }
   }
}

//t(n+1)[i][j]
for (int i = 0; i < ph.Length; i++)
{
   for (int j = i + 1; j < ph[i].Length; j++)
   {
      for (int s = 0; s < visits.Length; s++)
      {

         double curWayL = CurrentWay(visits[s], dists);

         //t(n+1) = t*c + k/L
         if (way[i][j] == true) {
```

```
                ph[i][j] = dcrsPh * ph[i][j] + (antPh / curWayL);
                ph[j][i] = dcrsPh * ph[i][j] + (antPh / curWayL);
            }
            //t(n+1) = t*c + 0/L
            else
            {
                ph[i][j] = dcrsPh * ph[i][j];
                ph[j][i] = dcrsPh * ph[i][j];
            }
            }
        }
    }
}

//best way in matrix
public static int[] BestWay(int[][] antsWay, int[][] distance)
{
    // best way that ants found
    double bLenght = CurrentWay(antsWay[0], distance);
    int res = 0;
    for (int s = 1; s < antsWay.Length; s++)
    {
        double len = CurrentWay(antsWay[s], distance);
        if (len < bLenght)
        {
            bLenght = len;
            res = s;
        }
    }
    int size = antsWay[0].Length;
    int[] bestWay = new int[size];
    bestWay = antsWay[res];
```

```
    return bestWay;
}

//AS Elite for pheromon way on last iteration
public static double PheromonWay(double[][] phWay, int[][] d)
{
    double res = 0.0;
    int[] blackList = new int[d.Length];
    int[][] sortedDist = d;

    int elem = d.Length - 1;
    //Console.WriteLine(elem);
    int currrentElement = 0;
    int listCount = 0;
    double max = 0.0;
    while (elem > 0)
    {
        int nextElement = 0;
        for (int i = 0; i < d.Length; i++)
        {
            if (phWay[currrentElement][i] > max)
            {
                bool isPassed = false;
                for (int s = 0; s < blackList.Length; s++)
                {
                    if (blackList[s] == i)
                    {
                        //Console.WriteLine("pass " + i);
                        isPassed = true;
                    }
                }
                if (isPassed == false)
```

```
                {
                    //Console.WriteLine("www " + phWay[currrentElement][i]);
                    max = phWay[currrentElement][i];
                    nextElement = i;
                }
            }
        }
        //Console.WriteLine("curr - " + currrentElement + ", next - " + nextElement);
        res += d[currrentElement][nextElement];
        currrentElement = nextElement;
        blackList[listCount] = currrentElement;
        listCount++;
        res += phWay[currrentElement][nextElement];
        max = 0.0;
        elem--;
    }

    return res;
}

//most useful class
public void showTable(double[][] table)
{
    for(int i = 0; i < table.Length; i++)
    {
        for(int j = 0; j < table[i].Length; j++)
        {
            Console.WriteLine("[i" + i + "], [j" + j + "] = " + table[i][j]);
        }
    }
}
}}
```