math_2020_031

**Міністерство освіти і науки України**

**Чернівецький національний університет**

**імені Юрія Федьковича**

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

# Розв'язання задач дискретної оптимізації на ПК

**Дипломна робота**

**Рівень вищої освіти - другий (магістерський)**

Виконала:
студентка 6 курсу, групи 607
спеціальності 124 – Системний аналіз_
(назва спеціальності)
____Бляхер Діана Володимирівна____
(прізвище, ім'я та по-батькові)
Керівник к.ф.-м.н., доцент Юрченко І.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

**До захисту допущено:**

**Протокол засідання кафедри № 6**

від „ 3 ” грудня 2019 р.

зав. кафедри _________ доц. Піддубна Л.А.

Чернівці – 2019

## Анотація

У роботі розглядаються класичні задачі дискретної оптимізації та їх розв’язання на комп’ютері засобами мови програмування Python.

## ЗМІСТ


Вступ ............ 3

§1. Постановка задач дискретної оптимізації ............ 5

п.1. Основні типи задач дискретної оптимізації ............ 5

п.2. Задача про пакування рюкзака ............ 6

п.3. Задача оптимального розкрою ............ 8

п.4. Задача про покриття множини системою її підмножин ............ 11

п.5. Задача комівояжера ............ 12

п.6. Оптимізація структури обчислювального кластера ............ 15

§2. Алгоритми розв'язання задач дискретної оптимізації на ПК ............ 22

п.1. Задача про пакування рюкзака ............ 22

п.2. Задача комівояжера. Метод найближчого сусіда ............ 24

п.3. Задача про вісім ферзів ............ 29

п.4. Задача комівояжера. Мурашиний алгоритм ............ 39

Висновки ............ 44

Список використаної літератури ............ 45

Додатки ............ 46

Додаток 1 ............ 46

Додаток 2 ............ 47

Додаток 3 ............ 48

Додаток 4 ............ 50

Додаток 5 ............ 51

Додаток 6 ............ 54

# Вступ

В економічних, виробничих, технологічних процесах різних галузей економіки виникають задачі, подібні за постановкою, що мають ряд спільних ознак та розв’язуються подібними методами. Деякий процес може розвиватися за різними варіантами, кожен з яких має свої переваги та недоліки, причому таких варіантів може бути безліч. Необхідно з усіх можливих варіантів вибрати найкращий. З цією метою використовуються математичні методи. Складність економічних систем (явищ, процесів) як об’єктів досліджень вимагає їх ретельного вивчення з метою з’ясування найважливіших функціональних залежностей, внутрішніх взаємозв’язків між їхніми елементами. В результаті здійснюються можливі спрощення та допущення, що погіршує адекватність побудованих математичних моделей. Не існує загальних рекомендацій щодо процесу моделювання, тому в кожному конкретному разі вимоги до побудови математичної моделі залежать від цілей та умов досліджуваної системи. Одним з класів задач моделювання економічних та інших процесів є задачі дискретної оптимізації.

Задача дискретної оптимізації – це задача знаходження екстремуму функції, заданої на дискретній (найчастіше – скінченій) множині точок. Якщо область визначення функції складається з скінченого числа точок, то задача дискретної оптимізації завжди, в принципі, можна вирішити перебором всієї цієї множини. Однак на практиці ця множина хоч і скінчена, але може бути дуже велика, так що методи перебору не ефективні. Розглянемо кілька загальних підходів до розв’язання задач дискретної оптимізації.

Дана дипломна робота складається зі вступу, двох параграфів, списку літератури, висновків та додатків.

Перший параграф присвячений постановці задач дискретної оптимізації. Наведено основні поняття та типи задач дискретної оптимізації, про які буде йти мова. Розглянені такі основні типи задач дискретної оптимізації: задача про пакування рюкзака, задача оптимального розкрою, задача про покриття

множини системою її підмножин, задача комівояжера та оптимізація структури обчислювального кластера.

Другий параграф містить самостійно розглянуті мною модельні приклади з використанням мови програмування Python. А саме: задача про пакування рюкзака, задача комівояжера, розв’язана за допомогою методу найближчого сусіда та мурашиного алгоритму, задача про вісім ферзів. Тексти програм наведено в додатках.

## §1. Постановка задач дискретної оптимізації

### п.1. Основні типи задач дискретної оптимізації

Задача дискретної оптимізації - це задача пошуку максимуму або мінімуму функції *f*, визначеної на скінченій або зліченій множині *D*:

$$f(x) \rightarrow extr\ ,\ x \in D\ . \tag{1.1}$$

Функція $f$ називається **цільовою** функцією, елементи множини D - допустимими рішеннями. Якщо множина *D* задається системою обмежень

$$g_i(x) \leq 0, i = 1, \dots, m_1, \qquad g_i(x) = 0, i = m_1 + 1, \dots, m,$$

$$x = (x_1, \dots, x_n) \in R^n, x_j \in \Omega_j \subset R, j = 1, \dots, n_1, n_1 \leq n,$$

де кожне $\Omega_j$ – або кінцева множина, що має не менше два елементи, або злічена множина, то задача (1.1) називається **задачею частково дискретного** (або дискретного, якщо $n_1 = n$) **математичного програмування.**

Зазвичай передбачається, що параметри задачі – коефіцієнти цільової функції $c_j, j = 1, \dots, n$, елементи матриці системи обмежень $A = (a_{ij})$ [1] і вектора правих частин $b = b_1, \dots, b_m$ – цілі числа.

Виділяють наступні основні класи задач дискретного програмування: транспортна задача і її варіанти, задачі з неподільностями, екстремальні комбінаторні задачі, задачі на некласичних областях, задачі з розривною цільовою функцією.

*Транспортна задача* лінійного програмування при цілочисельних вихідних даних завжди має цілочисельний оптимальний план. Тому для розв’язання дискретних задач, які можна сформулювати як транспортні задачі, доповнені умовами цілочисельністю змінних, можна застосовувати звичайні методи лінійного програмування. Найвідоміша задача цього класу – задача про призначення.

*Задачі з неподільностями* – це математичні моделі прикладних задач, змінні в яких є фізично неподільними величинами. Такі завдання описують, наприклад, планування випуску неподільних видів продукції або

використання неподільних виробничих факторів, це задачі розподілу ресурсів і капіталовкладень, мережевого планування і управління, виробничого планування і т.п.

У *комбінаторних задачах* оптимізується функція, задана на скінченій множині, елементами якого є вибірки (перестановки) з $n$ об'єктів. З комбінаторних задач, що мають велике прикладне значення, слід виділити *задачу комівояжера* і задачі теорії розкладів. При постановці комбінаторних задач у вигляді задач цілочислового математичного програмування часто вводяться булеві змінні $x_j \in \{0,1\}$, що носять логічний характер: $x_j = 1$, якщо виконується деяка умова і $x_j = 0$ в іншому випадку. Так, наприклад, коли рішення зводиться до вибору одного з можливих варіантів дій, включаючи і варіант відмови від яких би то не було дій.

*Задачі на некласичних областях* є задачами знаходження екстремуму лінійної функції на неопуклій або незв'язній області, що задається, наприклад, з використанням логічних умов виду «або-або». У *задачах з розривними цільовими функціями*, навпаки, допустима множина – опуклий багатогранник, але цільова функція не є неперервною. Наприклад, під неоднорідною розривною лінійною функцією розуміється функція

$$\sum_{j=1}^{n} c_j(x_j), \text{де } c_j(x_j) = \begin{cases} 0, & \text{при } x_j = 0 \\ c_j x_j + d_j, & \text{при } x_j > 0. \end{cases}$$

До появи подібних цільових функцій призводить, зокрема, облік в моделях постійних витрат, які повинні бути зроблені незалежно від обсягу виробництва. Задачі останніх двох класів можуть бути зведені до задач частково цілочисельного лінійного програмування шляхом введення додаткових, як правило, булевих змінних. [1]

Наведемо приклади постановки деяких задач дискретної оптимізації.

### п.2. Задача про пакування рюкзака

Задачею про пакування рюкзака називається задача цілочисельного лінійного програмування з одним обмеженням:

$$\sum_{j=1}^{n} c_j x_j \to max \quad (1.2)$$

$$\sum_{j=1}^{n} a_j x_j \leq b \quad (1.3)$$

$$x_j \geq 0 \in \mathbb{Z}, j = 1, \dots, n \quad (1.4)$$

Числа $c_j, a_j, (j = 1, \dots, n), b$ – додатні і цілі, $a_j \leq b$.

Свою назву задача отримала завдяки наступній її інтерпретації. Є $n$ видів неподільних предметів з вартостями $c_1, \dots, c_n$ і вагами $a_1, \dots, a_n$. Потрібно так упакувати ними рюкзак, щоб його вага не перевищувала $b$, а сумарна вартість упакованих предметів була максимальною. Вводячи змінні $x_j, j = 1, \dots, n$ для кількості упакованих предметів кожного типу, отримуємо задачу (1.2) – (1.4).

Іноді розглядається задача про рюкзак з обмеженням типу рівності

$$\sum_{j=1}^{n} a_j x_j = b,$$

що відповідає вимозі повного завантаження рюкзака вантажопідйомністю $b$.

Задача про рюкзак (1.2) – (1.4) може бути зведена до задачі з обмеженням типу рівності введенням додаткової невід'ємної цілочисельної змінної

$$x_{n+1} = b - \sum_{j=1}^{n} a_j x_j.$$

Якщо вважати, що є рівно по одному предмету кожного типу, то обмеження (1.4) слід замінити умовою

$$x_j \in \{0,1\}, j = 1, \dots, n, \quad (1.5)$$

тобто $x_j = 1$, якщо $j$-й предмет кладеться в рюкзак і $x_j = 0$, якщо ні. Отримана задача (1.2), (1.3), (1.5) називається задачею про бінарний або 0-1 рюкзак. При цьому передбачається додатково, що $\sum_{j=1}^{n} a_j > b$ тобто всі предмети в рюкзак упакувати не можна. Множина $D$ в цій задачі – множина $n$-вимірних бульових векторів з компонентами 0, 1, які відповідають умові (1.3). Очевидно, що $|D| = 2^n$.

Узагальненням задачі про рюкзак є задача про багатовимірний рюкзак

$$\sum_{j=1}^{n} c_j x_j \to max$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, i = 1, \dots, m, \qquad (1.6)$$

$$x_j \in (0,1), j = 1, \dots, n,$$

що має наступну економічну інтерпретацію. Нехай є $n$ проектів, очікуваний прибуток від реалізації яких становить $c_1, \dots, c_n$ . Заданий вектор ресурсів $b = b_1, \dots, b_m, b_i > 0, i = 1, \dots, m$; кількість одиниць ресурсу типу $i$, необхідне для реалізації проекту з номером $j$, рівне $a_{ij} > 0$. Для будь-якого ресурсу $i$ виконується умова $\sum_{j=1}^{n} a_{ij} > b_i$, тобто реалізація всіх проектів неможлива. Потрібно вибрати набір проектів з максимальним сумарним прибутком. Змінна $x_j, j = 1, \dots, n$ приймає значення 1, якщо проект $j$ реалізується і 0 в протилежному випадку. Множина допустимих рішень цієї задачі – це множина булевих векторів $x = (x_1, \dots, x_n)$, що задовольняють умови (1.6). Задача про рюкзак з обмеженнями типу нерівностей завжди мають допустиме рішення (наприклад, нульове). Крім того, допустиме рішення можна отримати, якщо відкинути умову цілочисельності, для задач про бінарний рюкзак – замінити умови (1.5) обмеженнями

$$0 \le x_j \le 1, j = 1, \dots, n,$$

вирішити отриману задачу лінійного програмування і округлити нецілі значення змінних $x_j$ в меншу сторону.

### п.3 Задача оптимального розкрою

**Задача оптимального розкрою** [2] полягає в тому, щоб вибрати один або декілька шляхів розкрою матеріалу й дізнатись, яку кількість матеріалу варто розкроювати, застосовуючи кожний з обраних способів. Завдання такого типу виникають у металургії й машинобудуванні, деревооброблюючій та легкої промисловості.

Виділяють два етапи розв’язання задачі оптимального розкрою. На першому етапі визначаються раціональні способи розкрою матеріалу, на другому – вирішується задача лінійного програмування для визначення інтенсивності використання раціональних способів розкрою.

**Визначення раціональних способів розкрою матеріалу.**

У задачах оптимального розкрою розглядаються раціональні (оптимальні по Парето) способи розкрою. Нехай, з одиниці матеріалу можна виготовити декілька видів заготівлі. Якщо збільшення числа заготівель одного виду можливо тільки за рахунок скорочення числа заготівель іншого виду, то такий спосіб розкрою одиниці матеріалу називається *раціональним (оптимальним по Парето), .*

Нехай $k$ – індекс виду заготівлі, $k = 1, \dots, q$, $i$ – індекс способу розкрою одиниці матеріалу, $i = 1, \dots, p$, $a_{ik}$ – кількість (ціле число) заготівель виду *k,* який ми отримали при розкрої одиниці матеріалу $i$-м способом. Наведене визначення раціонального способу розкрою може бути формалізоване наступним чином.

Спосіб розкрою *v* називається *раціональним (оптимальним по Парето),* якщо для будь-якого іншого способу розкрою *i* з співвідношень $a_{ik} \le a_{vk}, k = 1, \dots, q$, слідують співвідношення $a_{ik} = a_{vk}, k = 1, \dots, q$.

**Визначення інтенсивності використання раціональних способів розкрою.**

Позначення:

$j$ – індекс матеріалу, $j = 1, \dots, n;$

$k$ – індекс виду заготівлі, $k = 1, \dots, q;$

$i$ – індекс способу розкрою одиниці матеріалу, $i = 1, \dots, p;$

$a_{ijk}$– кількість (ціле число) заготівель виду *k,* отриманих при розкрої одиниці $j$-го матеріалу $i$-м способом;

$b_k$ – число заготівель виду $k$ у комплекті, що поставляється замовнику;

$d_j$ – кількість матеріалу $j$-го виду;

$x_{ji}$ – кількість одиниць $j$-го матеріалу, що розкроюються по $i$-му способу (інтенсивність використання способу розкрою);

$c_{ji}$ — величина відходу, отриманого при розкрої одиниці $j$-го матеріалу по $i$-му способу;

*в* — число комплектів заготівель різного виду, що поставляються замовнику.

**Модель *А* розкрою з мінімальною витратою матеріалів:**

$$\sum_{j=1}^{n}\sum_{i=1}^{p} x_{ji} \rightarrow min, \quad (1.7)$$

$$\sum_{j=1}^{n}\sum_{i=1}^{p} a_{jik}x_{ji} \geq b_k, k = 1, \dots, q, \quad (1.8)$$

$$x_{ji} \geq 0, j = 1, \dots, n; i = 1, \dots, p. \quad (1.9)$$

(1.7) – цільова функція (мінімум кількості використовуваних матеріалів);

(1.8) – система обмежень, що визначають кількість заготівель, необхідна для виконання замовлення;

(1.9) – умови незаперечності змінних.

Специфічними для даної області додатка моделі лінійного програмування є обмеження (1.8).

**Модель *В* розкрою з мінімальними відходами:**

$$\sum_{j=1}^{n}\sum_{i=1}^{p} c_{ji}x_{ji} \rightarrow min, \quad (1.10)$$

$$\sum_{j=1}^{n}\sum_{i=1}^{p} a_{jik}x_{ji} = b_k, k = 1, \dots, q, \quad (1.11)$$

$$x_{ji} \geq 0, j = 1, \dots, n; i = 1, \dots, p. \quad (1.12)$$

Тут (1.10) – цільова функція (мінімум відходів при розкрої матеріалів);

(1.11) – система обмежень, що визначають кількість заготівель, необхідна для виконання замовлення;

(1.12) – умови незаперечності змінних.

**Модель *С* розкрою з урахуванням комплектації:**

$$y \rightarrow max, \quad (1.13)$$

$$\sum_{i=1}^{p} x_{ji} \leq d_j,\ j = 1, \dots, n, \tag{1.14}$$

$$\sum_{j=1}^{n} \sum_{i=1}^{p} a_{jik} x_{ji} \geq b_k y,\ k = 1, \dots, q, \tag{1.15}$$

$$y \geq 0,\ x_{ji} \geq 0, j = 1, \dots, n;\ i = 1, \dots, p. \tag{1.16}$$

(1.13) – цільова функція (максимум комплектів, що включають заготівлі різних видів);

(1.14) – обмеження по кількості матеріалів;

(1.15) – система обмежень, що визначають кількість заготівель, необхідна для формування комплектів;

(1.16) – умови незаперечності змінних.

Специфічними для даної області додатка моделі лінійного програмування є обмеження (1.15).

**п.4. Задача про покриття множини системою її підмножин**

Задача про покриття множини системою її підмножин може бути формально визначена в такий спосіб

Нехай є скінчена множина $S = (\sigma_1, \sigma_2, \dots, \sigma_m)$ і система її підмножин $S_j \subset S, j = 1,2, \dots, n$, такі що

$$\bigcup_{j=1}^{n} S_j = S.$$

Кожній з підмножин $S_j$ поставлена у відповідність вага $c_j > 0$, таким чином розглядається задача про зважене покриття. Потрібно знайти мінімальний за кількістю підмножин набір $S_j$, такий, що кожен елемент множини $S$ належить хоча б одній з підмножин цього набору.

Введемо матрицю $A = (a_{ij})_{m \times n}$ наступним чином:

$$a_{ij} = \begin{cases} 1, & \text{якщо } \sigma_i \in S_j, \\ 0, & \text{якщо } \sigma_i \notin S_j. \end{cases}$$

Передбачається, що кожен елемент $\sigma_i$ входить хоча б в одне з підмножин $S_j$. Введемо булеві змінні $x_j, j = 1, \dots, n$:

$$x_j = \begin{cases} 1, & \text{якщо } S_j \text{ входить в покриття,} \\ 0, & \text{якщо } S_j \text{ не входить в покриття.} \end{cases}$$

Тоді задача про покриття множини полягає в знаходженні покриття з мінімальною вартістю за такою математичною моделлю:

$$\sum_{j=1}^{n} c_j x_j \rightarrow min \qquad (1.17)$$

на множині обмежень:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1,\ \ i = 1,2, \dots, m \qquad (1.18)$$

$$x_j \in \{0,1\},\ \ j = 1,2, \dots, n \qquad (1.19)$$

### п.5. Задача комівояжера

Класична задача комівояжера полягає в наступному. Комівояжер, що знаходиться в рідному місті, повинен відвідати кілька інших населених пунктів, побувавши в кожному по одному разу, і повернутися назад так, щоб загальна довжина його шляху була якомога менше [1].

Нехай всі пункти (включаючи початковий) пронумеровані і їх загальна кількість дорівнює $n$. Відстань від пункту $i$ до пункту $j$ позначається $c_{ij}$. Вважаючи $c_{ii} = \infty$ (комівояжеру забороняється залишатися на місці), визначаємо $n \times n$ матрицю $C = (c_{ij})$. Природно вважати, що $c_{ij} \geq 0$.

Будь-який маршрут комівояжера повністю визначається порядком відвіданих пунктів і тому має вигляд $z = (i_1, i_2, \dots, i_n, i_1)$. Довжина маршруту $z$ – сума відповідних елементів матриці $C$, тобто

$$l(z) = \sum_{k=1}^{n} c_{i_{k,} i_{k+1}},$$

де $i_{n+1} = i_1$ Позначивши множину допустимих маршрутів через $D$, отримуємо задачу комбінаторної оптимізації

$$l(z) \rightarrow min, z \in D.$$

Очевидним практичним застосуванням задачі комівояжера є планування маршрутів перевезення вантажів. Крім цього, рішення задачі комівояжера потрібно при розрахунку авіаційних ліній, оптимізації конвеєрного виробництва, складанні розкладів роботи устаткування і т.п. Під «пунктами» розуміються, наприклад, етапи виробничого процесу або різні операції, що виконуються на одному обладнанні.

Відповідно, $c_{ij}$ може являти собою не тільки відстань, але і час, витрати, інший вимірювач, що служить для визначення «вигідності» маршруту. У загальному випадку, $c_{ij}$ – вартість проходження *j* безпосередньо після *i*.

Нехай $G = (V, A)$ – повний граф з множиною вершин $V = \{1, \dots, n\}$ і множиною дуг *A*, довжина дуги (*i,j*) в якому дорівнює $c_{ij}$ . Контур (орієнтований цикл), що включає кожну вершину графа рівно один раз, називається *контуром Гамільтона*. Таким чином, задача комівояжера – це задача пошуку контуру Гамільтона, що має мінімальну довжину.

В силу замкнутості маршруту можна вважати, що початковий пункт $i_1$ заданий, тому кількість допустимих маршрутів равно (*n* – 1)!. Якщо $c_{ji} = c_{ij}$ для всіх $i, j = 1, \dots, n$ (матриця *C* симетрична), граф *G* є неорієнтованим, і говорять про цикли Гамільтона. Оскільки перестановки $(i_2, \dots, i_n)$ і $(i_n, \dots, i_2)$ визначають один маршрут, кількість різних замкнутих маршрутів складе $(n-1)!/2$.

Сформулюємо задачу комівояжера на мові математичного програмування [2]. Нехай $c_{ii} = M$, де $M$ - довільно велике число. Введемо змінні $x_{ij}$, $i, j = 1, \dots, n$, які приймають значення 1, якщо дуга $(i, j)$ міститься в оптимальному маршруті, і 0 в протилежному випадку. Тоді сумарна довжина маршруту має вигляд

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} . \qquad (1.20)$$

Набору змінних $\{x_{ij}\}$ відповідає граф з *n* вершинами, дуга $(i, j)$ в якому існує тоді і тільки тоді, якщо $x_{ij} = 1$. Зрозуміло, що для допустимого рішення

задачі цей граф повинен являти собою цикл, тобто бути зв'язним, і степінь кожної вершини повинна дорівнювати двом.

Виконання другої вимоги гарантують умови одноразового відвідування кожного пункту:

$$\sum_j x_{ij} = 1\,, i = 1, \dots, n, \tag{1.21}$$

$$\sum_i x_{ij} = 1\,, j = 1, \dots, n, \tag{1.22}$$

Один з можливих варіантів забезпечення зв'язності маршруту наступний. Для кожної власної підмножини $S \subset V, \bar{S} = V \backslash S$ вимагатимемо, щоб

$$\sum_{i \in S, j \in \bar{S}} x_{ij} \geq 1,$$

тобто повинна існувати принаймні одна дуга, що йде з $S$ в $\bar{S}$. Всього таких обмежень стільки ж, скільки власних підмножин $V$, тобто $2^n - 2$.

У більш короткому формулюванні кожній вершині $i = 2, \dots, n$ ставиться у відповідність змінна $u_i$ і слід дотримуватися наступних умов:

$$u_i - u_j + n x_{ij} \leq n - 1,\ \ i, j = 2, \dots, n, i \neq j. \tag{1.23}$$

Неважко показати, що обмеження (1.23) виключають наявність циклів довжиною менше $n$ і змінні $u_i$ можна вважати цілочисельними і невід'ємними. Згадуючи, що

$$x_{ij} \in \{0,1\}, i, j = 1, \dots, n, \tag{1.24}$$

отримуємо, що задача комівояжера набуває вигляду задачі цілочислового лінійного програмування, в якій є $n^2 + n - 1$ змінних і $n^2 - n + 2$ обмежень:

Мінімізувати функцію (1.20) при обмеженнях (1.21) – (1.24).

Існує досить багато різновидів і узагальнень задачі комівояжера. Розглянемо деякі з них.

Якщо критерієм оптимальності маршруту комівояжера є не загальна довжина шляху, а протяжність найдовшого переходу, задача комівояжера називається мінімаксною або задачею на вузьке місце. Потрібно знайти контур Гамільтона, що має мінімальну довжину найбільшої дуги:

$$\max c_{i_k, i_{k+1}} \to \min \ , (i_k, i_{k+1}) \in z, z \in D.$$

Знімемо тепер обмеження одноразового відвідування кожної вершини. Задача пошуку контуру найменшої довжини, що включає кожну вершину графа хоча б один раз, називається *спільною задачею комівояжера*. Постановка загальної задачі комівояжера зручна, якщо граф не є повним, тобто заборонено безпосередній перехід з деяких пунктів *i* в деякі пункти *j*. Зазвичай відсутнім дугам графа приписується нескінченна вага: $c_{ij} = \infty$ . Контур Гамільтона в графі може не існувати, тобто задача комівояжера може не мати допустимих рішень. Рішення загальної задачі комівояжера існує, якщо граф сильно-зв'язний (будь-які дві вершини можна з'єднати орієнтованим шляхом).

У *відкритій задачі комівояжера* [1] повернення в початковий пункт не передбачається: комівояжер починає свій шлях в деякому місті та відвідує кожне місто, що залишилось в точності один раз. При цьому кінцеві пункти маршруту можуть бути довільними або фіксованими. Шлях в графі, що проходить через кожну вершину рівно один раз, називається *шляхом Гамільтона*. Таким чином, відкрита задача комівояжера – це задача пошуку шляху Гамільтона, що має мінімальну довжину. Будь-яку відкриту задачу комівояжера можна звести до звичайної (закритої) або навпаки.

Будь-яке допустиме рішення відкритої задачі комівояжера можна ототожнити з перестановкою $z = (i_1, i_2, \dots, i_n)$ чисел $1, \dots, n$ . Довжина маршруту $z$ дорівнює $l(z) = \sum_{k=1}^{n-1} c_{i_k, i_{k+1}}$ . Кількість відкритих маршрутів комівояжера становить, очевидно, $n!$, якщо початковий і кінцевий пункти довільні, $(n-1)!$ або $(n-2)!$, якщо фіксовані один або обидва кінцевих пункти відповідно.

### п.6. Оптимізація структури обчислювального кластера

Підвищення відмовостійкості, надійності і продуктивності розподілених обчислювальних систем, які об'єднують в єдину систему множину окремих кластерів, досягається внаслідок динамічного перерозподілу запитів між ними

з урахуванням змін завантаженості кластерів, відмов і тимчасових відключень їх вузлів.

У розподіленій інфраструктури, що консолідує безліч ресурсів, об'єднаних в кластери, перерозподіл запитів (навантаження) може здійснюватися між вузлами як одного, так і різних кластерів, з'єднаних через мережу. При перерозподілі запитів між кластерами збільшуються витрати на взаємозв'язок через мережу, але зростають можливості балансування завантаження і адаптації до відмов і відключень вузлів, що обумовлює актуальність оптимізації процесу розподілу запитів.

Об'єктом дослідження є розподілена обчислювальна система (рис. 1), що включає $M$ локальних кластерів і загальнодоступний кластер, який об'єднує $m$ серверів.

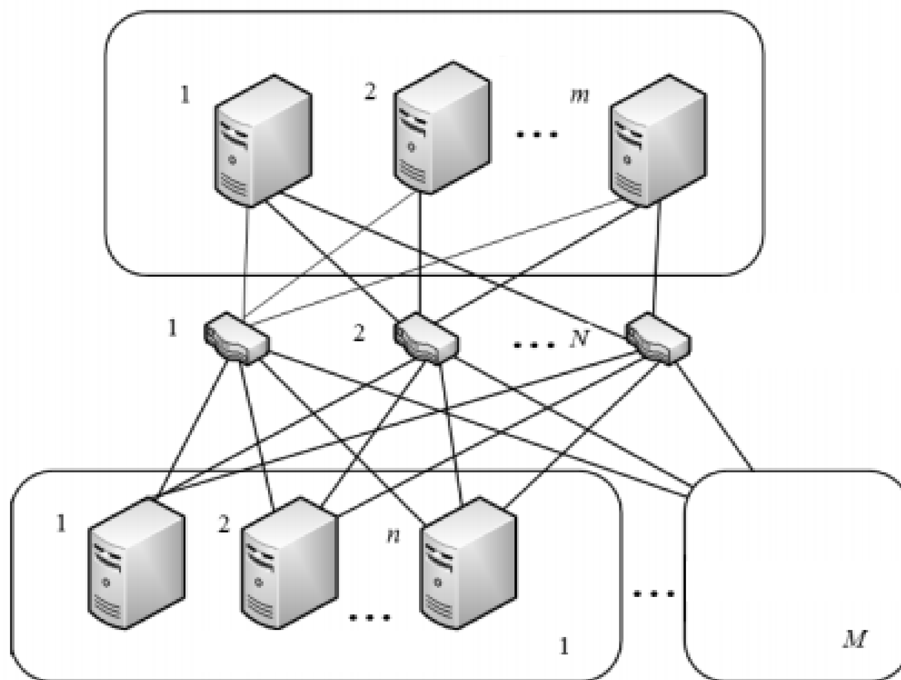



Рис. 1. Структура розподілюючої системи

В результаті перерозподілу запитів від локальних кластерів в загальнодоступний кластер забезпечується збалансованість навантаження вузлів системи і стійкість системи до відмов і перевантажень серверів локальних кластерів. Перерозподіл запитів від деякого локального кластера, що містить у вихідному (до відмов) стані $n$ серверів, в загальнодоступний

кластер здійснюється через $N$ резервованих комутаційних вузлів (маршрутизаторів або комутаторів).

При оптимізації структури визначається число (кратність резервування) серверів в локальних кластерах $n$ і в загальному кластері $m$, а також число комутаційних вузлів $N$, що забезпечують найбільшу надійність системи $P$ при заданих обмеженнях на вартість побудови системи $s$. При оцінці надійності системи враховуються нижні обмеження на кількість вузлів в підсистемах, при яких не виникають перевантаження відповідних кластерів.

При оптимізації процесу розподілу запитів з урахуванням можливості відмов і відключень вузлів загальнодоступного кластера будемо вважати заданими середні часи виконання запитів в серверах кластерів і в комутаційних вузлах $v_0, v_1$, їх інтенсивності відмов $\lambda_0, \lambda_1$, і відновлень $\mu_0, \mu_1$. Будемо вважати відомими ймовірності $r$ знаходження у ввімкненому стані серверів загальнодоступного кластера. Оптимізація проводиться при заданій інтенсивності потоку запитів $\lambda$, що надходить в локальний кластер і при необхідності перерозподіляється через мережу в загальнодоступний кластер, на який від інших кластерів системи через мережу додатково направляється потік запитів з інтенсивністю $\Lambda = \beta\lambda$ [2].

При оптимізації структури будемо вважати вартості серверів локальних і загальнодоступного кластера, а також вартість комутаційних вузлів відповідно рівними $c_0, c_1, c_2$.

В результаті оптимізації процесу розподілу потоку запитів, що надходить в локальний кластер, шукається їх частка, що перерозподіляється через мережу в загальнодоступний кластер, при якій мінімізується середній час перебування запитів $T$.

Визначимо ймовірність працездатності системи для локального кластера з $n$ серверів з урахуванням можливості використання в якості резерву ресурсів $m$ серверів загальнодоступного кластера, зв'язок з яким забезпечується через $N$ комутаційних вузлів.

Припустимо, що пропускна здатність кожного комутаційного вузла достатня, щоб не обмежувати можливості перерозподілу запитів, тобто якщо справний хоча б один комутаційний вузол [1], то запити можуть перерозподілятися в загальнодоступний кластер, але для реалізації такого перерозподілу в локальному кластері повинен бути справний хоча б один обчислювальний вузол. З урахуванням цих умов ймовірність працездатності системи становить

$$P = (1 - P_1)\left[\sum_{i=a}^{n} C_n^i p_0^i (1 - p_0)^{n-i}\right] + P_1 \sum_{j=b}^{n+m} (C_{n+m}^j - d_j C_m^j) p_0^j (1 - p_0)^{n+m-j}$$

де $d_j = 1$, якщо $j \leq m$, інакше *j=0;* $P_1 = \sum_{i=1}^{N} C_N^i p_1^i (1 - p_1)^{N-i}$ – ймовірність справності комутаційної підсистеми, при цьому з міркувань відсутності перевантаження кластерів значення $a$ і $b$ визначаються як найближчі цілі, більші $\lambda v_0$ і $\lambda(1 + \beta) v_0$.

Надійність вузлів визначимо за коефіцієнтами готовності, який обчислюється для серверів і комутаційних вузлів

$$p_0 = \frac{\mu_0}{\lambda_0 + \mu_0};\ p_1 = \frac{\mu_1}{\lambda_1 + \mu_1}.$$

Найпростіша структура дубльованого обчислювального комплексу (ДОК), скомплектована з двох пов'язаних через адаптер сполучення (АС) полукомплексів, що включають процесори (П) і модулі пам'яті (М), представлена на рис. 2, а. Модель надійності ДОК, що допускає можливість спільної роботи процесора і модуля пам'яті різних полукомплексів, зводиться до добре вивченої в теорії надійності моделі мостикової схеми, наведеної на рис. 2, б. Надійність (коефіцієнт готовності) ДОК, відповідно до моделі по рис. 2, б, обчислюється як

$$p_0 = p_a\left(1 - \left(1 - p_p\right)^2\right)\left(1 - (1 - p_M)^2\right) + (1 - p_a)\left(1 - \left(1 - p_p p_M\right)^2\right),$$

де при заданих інтенсивностях відмов $\lambda_p, \lambda_M, \lambda_a$ і відновлень $\mu_p, \mu_M, \mu_a$ процесора, пам'яті і адаптера сполучення відповідно маємо $p_0 = \frac{\mu_p}{\lambda_p+\mu_p}, p_M = \frac{\mu_M}{\lambda_M+\mu_M}, p_a = \frac{\mu_a}{\lambda_a+\mu_a}$.

У разі неможливості спільної роботи процесорів і модулів пам'яті різних полукомплексів надійність ДОК обчислимо як $p_0 = (1 - (1 - p_p p_M)^2)$.

Для ДОК з обмеженим відновленням (одночасний ремонт декількох вузлів неможливий) коефіцієнт готовності визначається як сума ймовірностей працездатних станів, для знаходження яких процес відмов і відновлень представляється процесом Маркова, при цьому складається граф переходів і рівняння Чепмена-Колмогорова, в результаті рішення яких і визначаються шукані ймовірності.

**Оптимізація структури**

При оптимізації структури розглянутої обчислювальної системи шукається число серверів *n* в локальних кластерах, число серверів *m* в загальнодоступному кластері і кратність резервування *N* комутаційних вузлів, що забезпечують максимум надійності системи, $P = \max_{m,n,N,g} P(m,n,N,g,\lambda)$ при обмеженні вартості *s* її реалізації $(Mc_0 n + c_1 N + c_2 m) \le s$ , і умови стаціонарності функціонування вузлів (відсутність перевантаження вузлів).

Пошук максимуму *P* може бути оснований на переборі, що реалізується з використанням засобів системи комп'ютерної математики Matchcad-15 [2].

Метою оптимізації структури може бути мінімізація середнього часу перебування запитів в системі при обмеженні коштів *s* на її побудову, $T = \min_{m,n,N,g} T(m,n,N,g,\lambda)$ при цьому середній час перебування запитів в системі обчислюється як

$$T = g\left(\frac{v_0}{1-g\lambda v_0/n}\right) + (1-g)\left(\frac{2v_1}{1-\frac{((1-g)+\beta)2\lambda v_1}{N}} + \frac{v_0}{1-\frac{((1-g)+\beta)2\lambda v_0}{m}}\right), \qquad (1.26)$$

де *(1 - g)* – середня частка запитів, перерозподілених через мережу від локального кластера в загальнодоступний. При пошуку оптимального $g$

необхідно враховувати умову стаціонарного режиму функціонування вузлів (умова відсутності перевантаження вузлів):

$$\left(\frac{g\lambda v_0}{n} < 1\right) \wedge \left(\frac{\left((1-g)+\beta\right)2\lambda v_1}{N} < 1\right) \wedge \left(\frac{\left((1-g)+\beta\right)2\lambda v_0}{m} < 1\right). \quad (1.27)$$

При необхідності оптимізація може бути проведена по мультиплікативному критерію $r(m,n,N,g,\lambda) = \max_{m,n,N,g}(P(m,n,N)/T(m,n,N,g,\lambda))$.

**Оптимізація процесу перерозподілу запитів**

При заданій структурі системи (сформованої при розглянутої вище структурної оптимізації) проведемо оптимізацію процесу розподілу запитів з урахуванням можливості відмов і відключень справних вузлів загальнодоступного кластера з імовірністю (*1 - r*). оптимізація проводиться при заданій середній інтенсивності потоку запитів $\lambda$, що надходить в локальний кластер і при необхідності перерозподіляється через мережу в загальнодоступний кластер.

В результаті оптимізації процесу розподілу потоку запитів, що надходить в локальний кластер, шукається їх частка, що перерозподіляється через мережу в загальнодоступний кластер [1], при якій мінімізується середній час перебування запитів $T$. $T = \min_{g} T(m,n,N,g,\lambda)$ де при модернізації (1.26) і (1.27) маємо

$$T = g\left(\frac{v_0}{1-g\lambda v_0/n}\right) + (1-g)\left(\frac{2v_1}{1-\frac{((1-g)+\beta)2\lambda v_1}{N_c}} + \frac{v_0}{1-\frac{((1-g)+\beta)2\lambda v_0}{m_c}}\right),$$

$$\left(\frac{g\lambda v_0}{n} < 1\right) \wedge \left(\frac{\left((1-g)+\beta\right)2\lambda v_1}{N_c} < 1\right) \wedge \left(\frac{\left((1-g)+\beta\right)2\lambda v_0}{m_c} < 1\right)$$

при математичних очікуваннях числа комутаційних вузлів $N_c$ і доступних справних серверів загальнодоступного кластера, що обчислюються як

$$N_c = \sum_{i=1}^{N} iC_N^i p_1^i (1-p_1)^{N-i}, \; m_c = \sum_{j=1}^{m} iC_m^j p_2^j (1-p_2)^{m-j},$$

$$\left(\frac{g\lambda v_0}{n} < 1\right) \wedge \left(\frac{((1-g)+\beta)2\lambda v_1}{N_c} < 1\right) \wedge \left(\frac{((1-g)+\beta)2\lambda v_0}{m_c} < 1\right).$$

Для прикладу проведемо оптимізацію процесу розподілу запитів при $n$ = 8 шт., $N$ = 5 шт., $m$ = 23 шт .; $v_0$ = 10 с, $v_1$ = 1 с, $r$ = 0,8; $\lambda_0 = \lambda_2 = 10^{-4}$ 1/год, $\lambda_1$ = 0,5 $\cdot 10^{-4}$ 1/год; $\mu_0 = \mu_1 = \mu_2 = 1$ 1/год. Результати пошуку оптимальної частки (1–$g$), які розподіляються через мережу в загальнодоступний кластер запитів, в залежності від інтенсивності вхідного потоку запитів $\lambda$ 1/с представлені на рис. 3 при $\beta$ = 0,5 і $\beta$ = 1. Зростання частки неперераспределяемих запитів $g$ при незначній інтенсивності $\lambda$ потоку запитів пояснюється впливом додаткових затримок при передачі запитів через мережу, а при значній інтенсивності $\lambda$ – перевантаженням загальнодоступного кластера.

## § 2. Алгоритми розв’язання задач дискретної оптимізації на ПК

### п.1. Задача про пакування рюкзака

Задачу про пакування рюкзака можна вирішити, використовуючи динамічне програмування – $O(nW)$ за часом і пам'яттю алгоритм:

1. Починаємо з пустого рюкзака (дозволена вага дорівнює повній ємності рюкзака – $W$) і всіх предметів (всі $n$ предметів доступні).
2. Якщо найбільша цінність, яку можна отримати, використовуючи всі доступні предмети (індекси не більше заданого), більше цінності, яку можна отримати, не включаючи поточний предмет (з найбільшим індексом), то
    - додаємо поточний предмет в результат
    - зменшуємо дозволену вагу на величину рівну вазі поточного предмета
3. Повторюємо крок №2, виключаючи предмет з найбільшим індексом зі списку доступних предметів, поки все ще залишилися доступні предмети.

Де найбільша цінність визначається рекурентними співвідношеннями:

- якщо немає доступних предметів, то найбільша цінність дорівнює нулю
- якщо вага поточного предмета більше дозволеної ваги, то результат дорівнює найбільшій цінності без цього предмета (з рештою $n-1$ доступними предметами)
- в іншому випадку, вибираємо максимум з двох варіантів:

  (А) варіант, який виключає поточний предмет

  (Б) варіант, який включає поточний предмет. Найбільша цінність в цьому випадку дорівнює сумі цінності від поточного предмета і найбільшої цінності доступних предметів з дозволеною вагою, що

залишилися, зменшеною на вагу поточного предмета (в рюкзаку залишилося менше місця).

Рекурсивне рішення на Python (див. додаток 1).

Результат

[Item(value = 3, weight = 2), Item(value = 3, weight = 4)]

CacheInfo(hits = 9, misses = 21, maxsize = None, currsize = 21)

**Приклад**

$W$=13, $N$=5 W=13, N=5

$w_1$=3, $p_1$=1 w1=3, p1=1

$w_2$=4, $p_2$=6 w2=4, p2=6

$w_3$=5, $p_3$=4 w3=5, p3=4

$w_4$=8, $p_4$=7 w4=8, p4=7

$w_5$=9, $p_5$=6

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| k = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| k = 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| k = 2 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| k = 3 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 11 | 11 |
| k = 4 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |
| k = 5 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |

Числа від 0 до 13 в першому рядку позначають місткість рюкзака.

У першому рядку як тільки місткість рюкзака n⩾3, додаємо в рюкзак 1 предмет.

Розглянемо $k$ = 3, при кожному $s$⩾5 (так як $w$3 = 5) порівнюємо $A$ $[k\text{-}1][s]$ і $A[k\text{-}1][s\text{-}w3]$ + $p$3 і записуємо в $A[k][s]$ вартість або рюкзака без третього предмета, але з такою же вагою, або з третім предметом, тоді вартість дорівнює вартості третього предмета плюс вартість рюкзака з місткістю на $w$3 менше.

Максимальна вартість рюкзака знаходиться в *A* (5,13).

**Відновлення набору предметів, з яких складається максимально дорогий рюкзак.**

Починаючи з *A* (5,13) відновлюємо відповідь. Будемо йти в зворотному порядку по *k*. Червоним тлом позначимо наш шлях

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| k = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| k = 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| k = 2 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| k = 3 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 11 | 11 |
| k = 4 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |
| k = 5 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |

Таким чином, в набір входить 2 і 4 предмет.

Вартість рюкзака: 6 + 7 = 13

Вага рюкзака: 4 + 8 = 12

## п.2. Задача комівояжера методом найближчого сусіда

### Рішення задачі комівояжера методом найближчого сусіда на Python

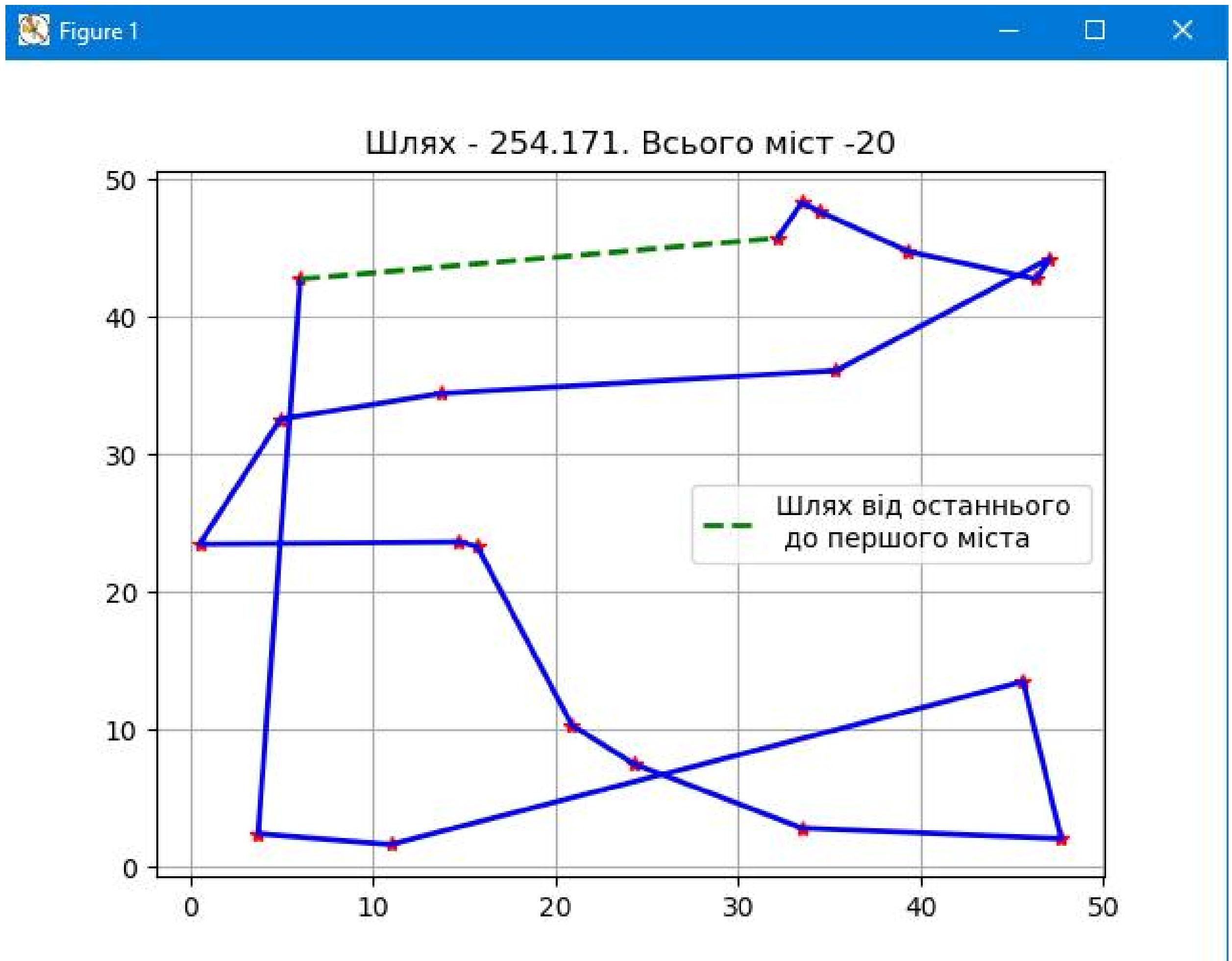
Серед методів розв’язання задачі комівояжера метод найближчого сусіда привертає простотою алгоритму. Метод найближчого сусіда в вихідному формулюванні полягає в знаходженні замкнутої кривої мінімальної довжини, що з'єднує заданий набір точок на площині [9]. Мою увагу привернула найбільш поширена реалізація даного алгоритму в пакеті Mathcad. Сама реалізація не зовсім зручна, наприклад, не можна вивести матрицю відстаней між пунктами або проаналізувати альтернативні маршрути.

Інше рішення – перебір всіх міст (вершин графа) в якості початку маршруту і вибір найліпшого з усіх маршрутів [9]. Однак реалізація останньої пропозиції не приведена. З огляду на всі перераховані обставини, я вирішила

реалізувати наведений алгоритм на Python і при цьому передбачити можливість вибору початкового пункту за критерієм мінімальної довжини маршруту.

Програмний код на мові програмування Python див у додатку 2.

**В результаті роботи програми отримаємо:**




Недоліки методу видно на графіку, про що свідчать петлі. Реалізація алгоритму на Python має більше можливостей, ніж у Mathcad. Наприклад, можна вивести матрицю відстаней між пунктами на друк. Наприклад, для n = 4, отримаємо [10]:

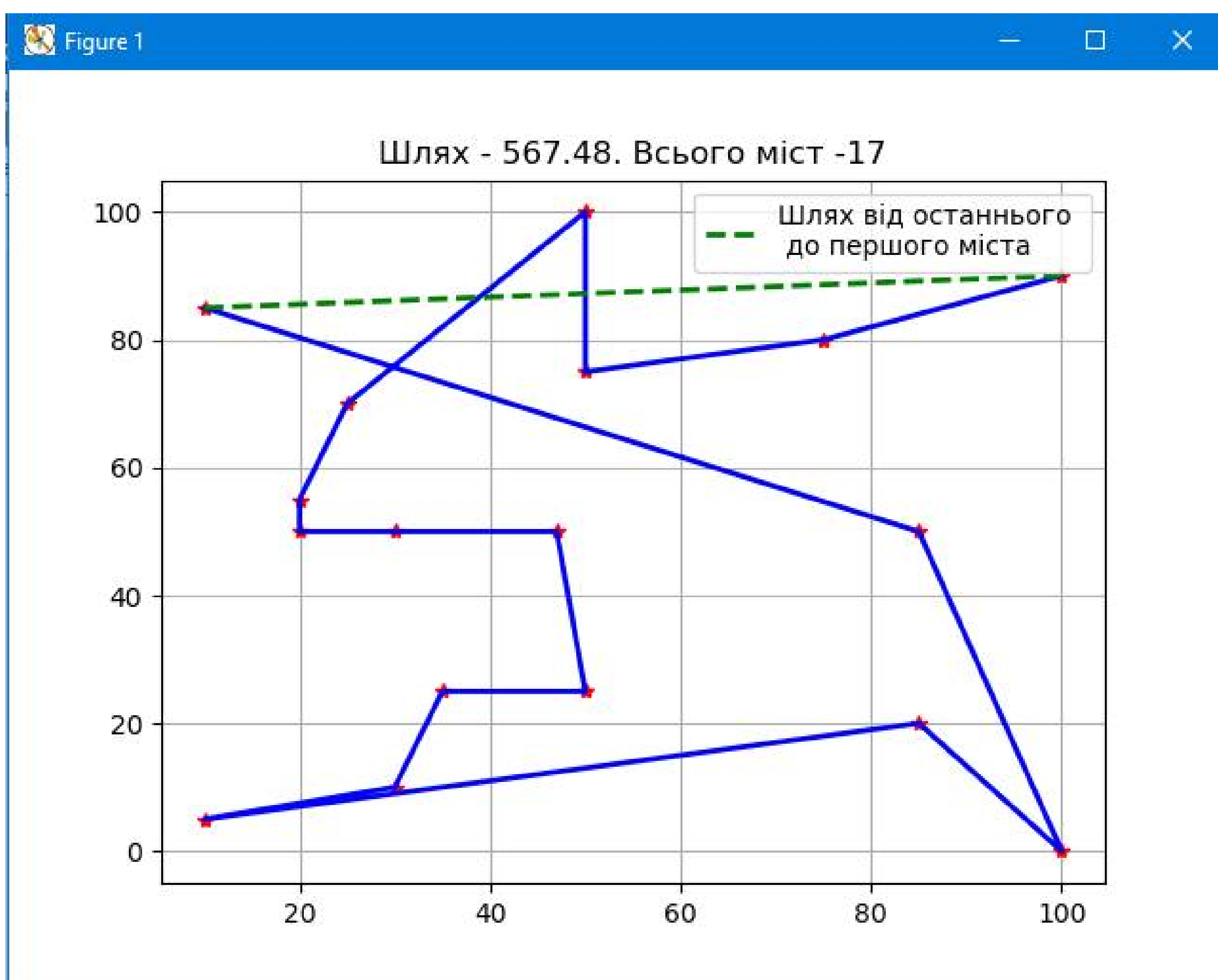
```
[[ inf 43.91676312 48.07577298 22.15545245] [43.91676312 inf 54.31419355 21.7749088 ] [48.07577298 54.31419355 inf 46.92141965] [ 22.15545245 21.7749088 46.92141965 inf]]
```

Для роботи алгоритму на головній діагоналі матриці числові значення встановлюють рівними нескінченності.

Від випадкових координат пунктів перейдемо до заданих. Для роботи програми в режимі заданих координат в наведеному лістингу приберемо коментарі з наступних рядків.

```
X=[10, 10, 100,100 ,30, 20, 20, 50, 50, 85, 85, 75, 35, 25, 30, 47, 50] Y=[5, 85, 0,90,50, 55,50,75 ,25,50,20,80,25,70,10,50,100] n=len(X)
```

**В результаті роботи програми отримаємо:**



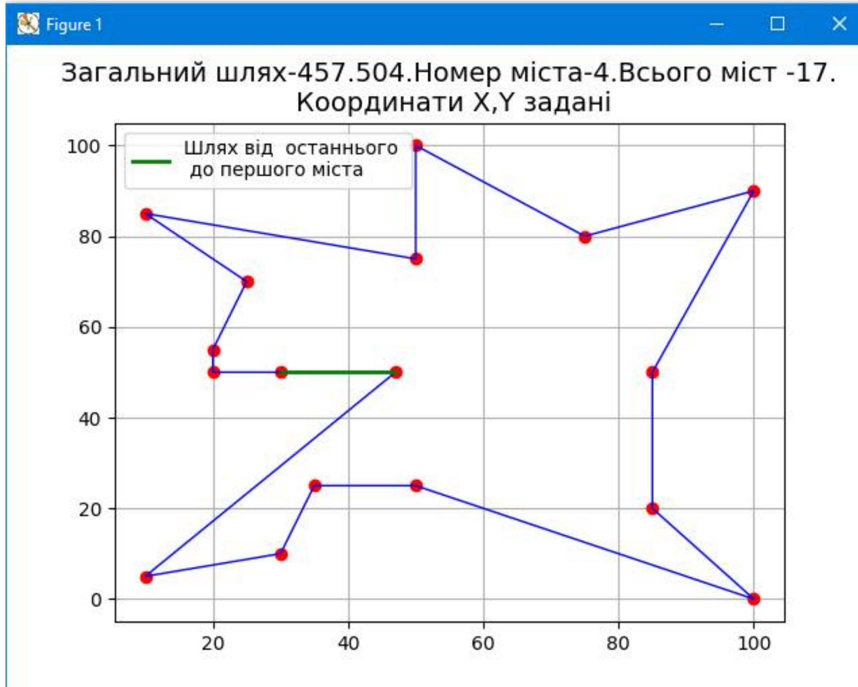

З наведеного графіка видно, що траєкторії переміщення комівояжера перетинаються.

**Удосконалення алгоритму найближчого сусіда:**

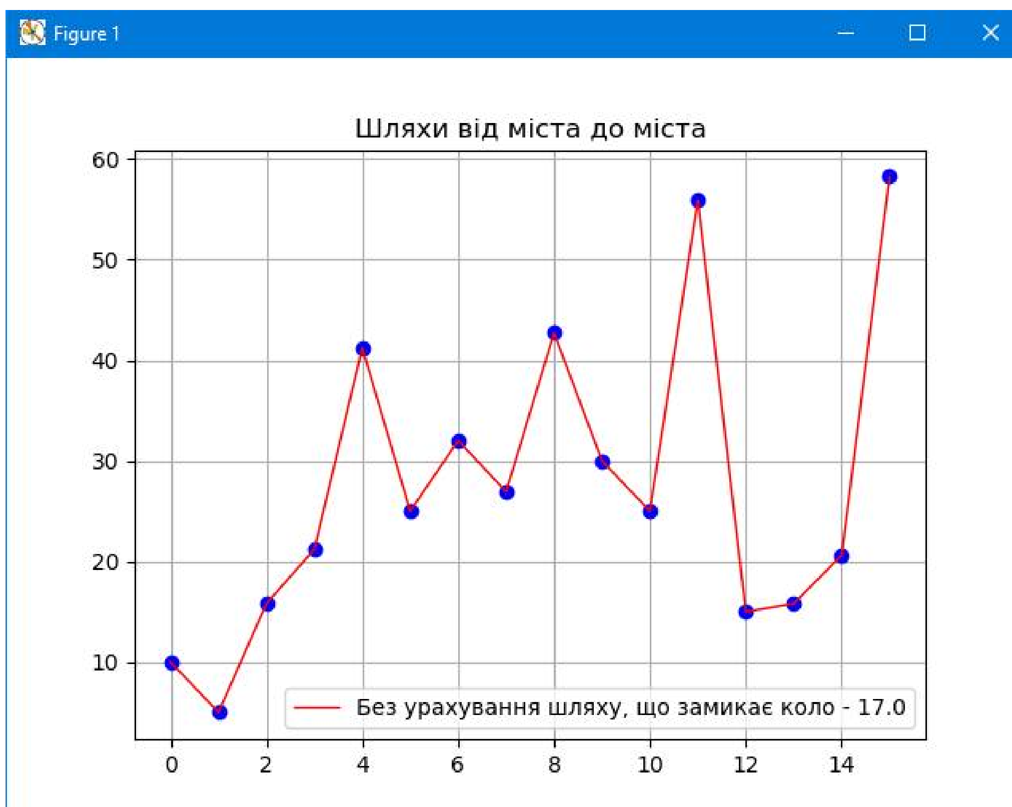
Тепер можна модифікувати мою програму за критерієм мінімальної довжини маршруту.

Код програми (див. додаток 3) для визначення початкового пункту з умови мінімальної довжини маршрута (модифікований алгоритм).

**В результаті роботи програми отримаємо:**




Петель на графіку немає, що свідчить про поліпшення алгоритму.



Довжина маршруту при початку руху з пункту з номером 4 менше, ніж при початку руху з інших пунктів.

```
Довжина маршруту - 458.662, при початку руху з пункту - 0
Довжина маршруту - 463.194, при початку руху з пункту - 1
Довжина маршруту - 560.726, при початку руху з пункту - 2
Довжина маршруту - 567.48, при початку руху з пункту - 3
Довжина маршруту - 457.504, при початку руху з пункту - 4
Довжина маршруту - 465.714, при початку руху з пункту - 5
Довжина маршруту - 471.672, при початку руху з пункту - 6
Довжина маршруту - 460.445, при початку руху з пункту - 7
Довжина маршруту - 533.461, при початку руху з пункту - 8
Довжина маршруту - 532.326, при початку руху з пункту - 9
Довжина маршруту- 564.516, при початку руху з пункту - 10
Довжина маршруту - 565.702, при початку руху з пункту - 11
Довжина маршруту - 535.539, при початку руху з пункту - 12
Довжина маршруту - 463.194, при початку руху з пункту - 13
Довжина маршруту - 458.662, при початку руху з пункту - 14
Довжина маршруту - 457.504, при початку руху з пункту - 15
Довжина маршруту- 508.045, при початку руху з пункту - 16
```

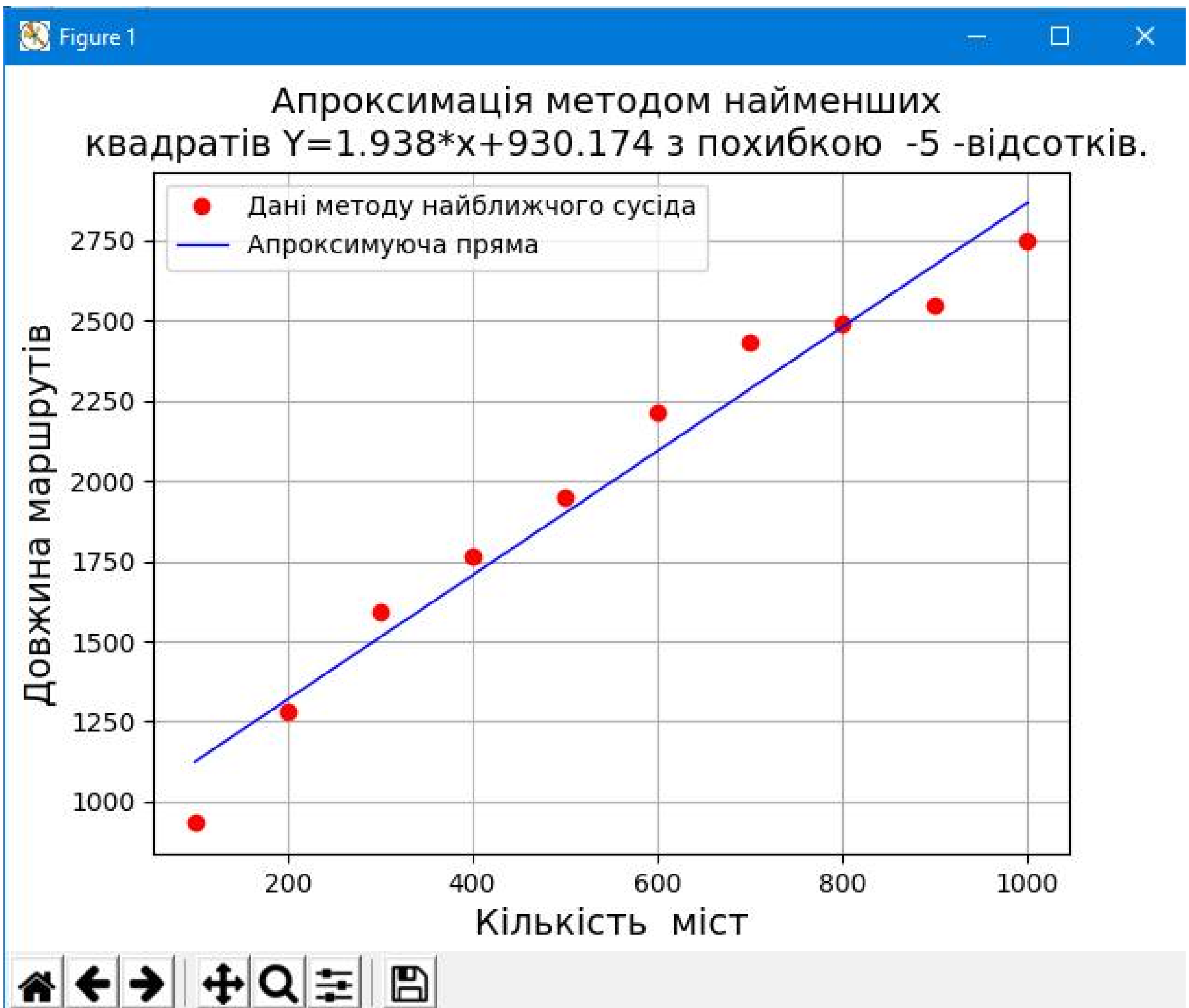
Залежність довжини маршруту від кількості пунктів (міст):

Для цього повернемося до генерації випадкових значень координат. За результатами роботи програми збільшуючи кількість пунктів до 1000 з кроків в 100 складемо таблицю з двох рядків, в однієї довжини маршрутів в іншій кількість пунктів в маршруті.

| 933,516 | 1282,842 | 1590,256 | 1767,327 | 1949,975 | 2212,668 | 2433,955 | 2491,954 |
|---|---|---|---|---|---|---|---|
| 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 |
| 2549,579 | 2748,672 | | | | | | |
| 900 | 1000 | | | | | | |

Апроксимуємо наведені дані за допомогою програми. (див. додаток 4)

Отримаємо:




При використанні методу найближчого сусіда довжина маршруту і число пунктів зв'язані лінійною залежністю в наведеному діапазоні. За наведеними даними кращі результати показали алгоритм Літтла, генетичний алгоритм і модифікація алгоритму «йди в ближній» [10].

### п. 3. Задача про вісім ферзів

У житті ми часто стикаємося з проблемами планування сімейного бюджету (куди необхідно витратити більшу частину грошей), покупки товару за найбільш вигідною ціною (висока якість або необхідні характеристики за прийнятною або мінімальної вартості) і т.д. Всі ці приклади можна звести до оптимізації або пошуку найбільш прийнятного рішення в рамках того чи іншого завдання.

Але такого роду проблеми вирішує не тільки людина, але і природа. Розглянемо такий цікавий і нетривіальний процес, як **відпал металу.** [4] Процес контрольованого і рівномірного зниження температури розплавленого металу призводить до формування, в кінцевому підсумку, міцної кристалічної решітки (структура, яка описує геометричне положення атомів речовини).

Кожному стану і розташуванню атомів, які і складають кристалічну решітку, відповідає певний енергетичний рівень. При цьому може відбуватися перехід атомів з одного енергетичного стану в інший довільним чином в залежності від температури. Чим вище температура, тим вище ймовірність цього переходу. Відповідно, при зниженні температури відбувається поступове зменшення ймовірності переходу атомів з одного енергетичного стану в інший. На що варто звернути увагу: атоми в процесі відпалу можуть зайняти невигідні (з точки зору оптимальності структури) положення тільки для того, щоб в подальшому оптимальним чином перебудувати кристалічну решітку.

Таким чином, при рівномірному охолодженні фізична система буде прагнути досягти стану з мінімальною енергією, що призведе до формування найбільш якісної структури металу. Необхідно визначити три найважливіших компонента, які виробляють динамічні перетворення стану системи: температура (при зниженні процеси перебудови зменшуються), енергетичний стан (поточний показник фізичного стану), зміна системи (момент переходу в новий стан).

Якщо розширити горизонти абстракції, то модель імітації відпалу можна успішно застосувати в ряді оптимізаційних задач, які на практиці пов'язані з магічним пошуком глобального екстремуму (найбільшого або найменшого значення функції). Дійсно, якщо за поточне рішення або значення функції прийняти енергетичний стан, а зміну системи за новий спосіб обчислити це рішення (наприклад, випадковим чином переставити потенційні рішення) при певній температурі. В таких умовах, температура буде регулювати ймовірність

прийняття обчисленого рішення (чим температура нижче, тим ймовірність прийняти рішення навіть не дуже вдале нижче і відповідно навпаки).

З одного боку, врахувати невідповідне рішення це як крок назад – відбувається навмисне прийняття неправильного рішення (в цьому і є головна відмінність від «жадібних» алгоритмів, в яких приймаються тільки найбільш підходящі критерієм рішення). З іншого боку, знижується ймовірність «провалу» в локальний мінімум або зациклення на одному з хороших вирішень.

Постановка задачі:

Необхідно знайти глобальний екстемум функції (може бути нерівною тобто недиференційованою) на заданій області визначення із заданою точністю за прийнятний час.

Лінгвістичний опис алгоритму:

Як зазначалося вище – необхідно перенести аналогію відпалу металу на алгоритм пошуку глобального екстремуму. Саму цільову функцію (яку оптимізуємо) приймемо за енергію системи – E. Тобто значення енергії буде відповідати значенню цільової функції. [3] Для переходу в новий стан (зміна системи) необхідно ввести функцію нового стану, як правило, така функція виробляє випадкове «перетасування», отриманих рішень від яких залежить значення цільової функції (енергія системи) – S. Температура (T) буде визначати тривалість роботи алгоритму і регулювати ймовірність прийняття рішення на основі обчислених значень функції нового стану (переходу в новий стан).

Після синхронізації моделі відпалу із завданням оптимізації, опишемо роботу алгоритму. [3]

Крок 1. Ініціалізується початкова температура ($t_{max}$ – початок відпалу) і мінімальна температура ($t_{min}$ – кінець відпалу).

Крок 2. Визначається функція нового стану і задається довільний початковий стан системи – $S_0$;

Крок 3. Порівнюється поточне значення цільової функції (енергія системи до переходу в новий стан – E ($S_0$)) і значення після прийняття нового стану (енергія після переходу в новий стан - E ($S_i$) - на i - кроці);

Крок 4. Якщо обчислене значення цільової функції після прийняття нового стану виявилося менше (для завдання пошуку глобального мінімуму), то зберігаємо його в якості поточного значення цільової функції. E ($S_i$)) < E ($S_0$)) => $S_0$ = $S_i$;

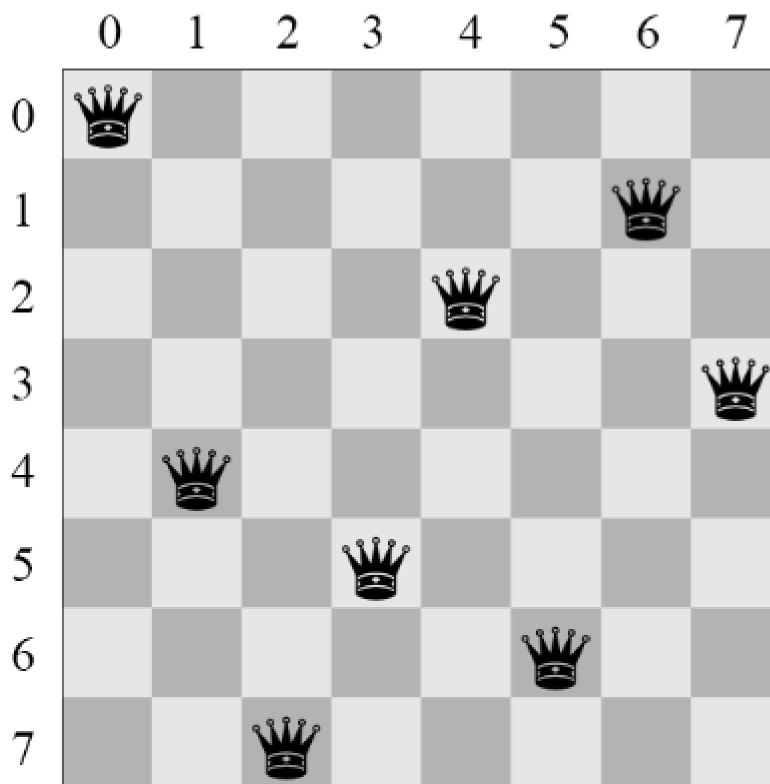
Крок 5. В іншому випадку, обчислюється ймовірність прийняття поточного значення навіть якщо воно більше (не задовольняє умовам задачі). Ймовірність дорівнює $e^{-\Delta E/T_i}$, де ΔE - різниця поточного значення цільової функції до і після прийняття нового стану E ($S_0$)) – E ($S_i$)) , $T_i$ поточна температура. Якщо така ймовірність більше певного значення (наприклад, 0.5), то зберігаємо новий стан системи $S_0$ = $S_i$;

Крок 6. Обчислюємо нове значення температури $T_i$ (температура може знижуватися відповідно за певною закономірністю, наприклад лінійною, $T_i$ = $T_{i-1}$ * α / i, де i – крок на певній ітерації і α – настроюється коефіцієнт, зазвичай приймається за значення в інтервалі 0.8 <= α <= 0.99).

Крок 7. Якщо $T_i$ <= $T_{min}$. Алгоритм завершується, в іншому разі перехід на Крок 2. [3]

**Реалізація:**

Розглянемо задачу про шахових ферзів. Пропонується розмістити певну кількість ферзів на шахівниці так, щоб жоден з ферзів не попадав під удар будь-якого іншого – тобто не існувало шахового конфлікту між усіма фігурами. приклад:




Дана задача відноситься до класу оптимізаційних задач тому в підсумку потрібно мінімізувати кількість конфліктних ситуацій між ферзями при заданих обмеженнях: кількість фігур, розмір шахівниці. [4]

При використанні методу імітації відпалу необхідно визначити три головні компоненти при його реалізації - функцію енергетичного стану E ($S_i$), функцію переходу $S_i$ і температуру – $T_i$ . За енергетичний стан можна прийняти число конфліктних ситуацій при поточному розташуванні фігур, а за функцію переходу прийняти нове розташування ферзів на шахівниці. [4] (див. додаток 5)

Результат роботи програми:

```
Приклад вирішення задачі розміщення ферзів на шахматному полі без конфліктів
методом відпалу

Розмір поля: 20
Початкова температура: 30.0
Кінцева температура: 0.5
Коефіцієнт зміни температури: 0.99
Кількість иерацій перед зміною температури: 100

NOTE: При обчисленні відображається поточна ітерація, температура,
енергія(кіл. конфліктів) и кіл. принятих гірших рішень
1: t: 30.00 Енергія: 10 Прийнято гірших: 39
2: t: 29.70 Енергія: 8 Прийнято гірших: 39
3: t: 29.40 Енергія: 8 Прийнято гірших: 42
4: t: 29.11 Енергія: 8 Прийнято гірших: 45
5: t: 28.82 Енергія: 8 Прийнято гірших: 45
6: t: 28.53 Енергія: 8 Прийнято гірших: 35
7: t: 28.24 Енергія: 8 Прийнято гірших: 42
8: t: 27.96 Енергія: 6 Прийнято гірших: 33
9: t: 27.68 Енергія: 6 Прийнято гірших: 41
10: t: 27.41 Енергія: 6 Прийнято гірших: 39
11: t: 27.13 Енергія: 6 Прийнято гірших: 44
12: t: 26.86 Енергія: 6 Прийнято гірших: 42
13: t: 26.59 Енергія: 4 Прийнято гірших: 38
14: t: 26.33 Енергія: 4 Прийнято гірших: 40
15: t: 26.06 Енергія: 4 Прийнято гірших: 43
16: t: 25.80 Енергія: 4 Прийнято гірших: 40
17: t: 25.54 Енергія: 4 Прийнято гірших: 43
18: t: 25.29 Енергія: 4 Прийнято гірших: 41
19: t: 25.04 Енергія: 4 Прийнято гірших: 40
20: t: 24.79 Енергія: 4 Прийнято гірших: 37
21: t: 24.54 Енергія: 4 Прийнято гірших: 39
22: t: 24.29 Енергія: 4 Прийнято гірших: 42
23: t: 24.05 Енергія: 4 Прийнято гірших: 32
24: t: 23.81 Енергія: 4 Прийнято гірших: 39
25: t: 23.57 Енергія: 4 Прийнято гірших: 42
26: t: 23.33 Енергія: 4 Прийнято гірших: 42
27: t: 23.10 Енергія: 4 Прийнято гірших: 40
28: t: 22.87 Енергія: 4 Прийнято гірших: 34
29: t: 22.64 Енергія: 4 Прийнято гірших: 36
30: t: 22.42 Енергія: 4 Прийнято гірших: 38
31: t: 22.19 Енергія: 4 Прийнято гірших: 34
32: t: 21.97 Енергія: 4 Прийнято гірших: 42
33: t: 21.75 Енергія: 4 Прийнято гірших: 40
34: t: 21.53 Енергія: 4 Прийнято гірших: 40
35: t: 21.32 Енергія: 4 Прийнято гірших: 43
36: t: 21.10 Енергія: 4 Прийнято гірших: 43
37: t: 20.89 Енергія: 4 Прийнято гірших: 43
38: t: 20.68 Енергія: 4 Прийнято гірших: 40
39: t: 20.48 Енергія: 4 Прийнято гірших: 46
40: t: 20.27 Енергія: 4 Прийнято гірших: 36
41: t: 20.07 Енергія: 4 Прийнято гірших: 40
42: t: 19.87 Енергія: 4 Прийнято гірших: 40
43: t: 19.67 Енергія: 4 Прийнято гірших: 35
44: t: 19.47 Енергія: 4 Прийнято гірших: 36
45: t: 19.28 Енергія: 4 Прийнято гірших: 34
46: t: 19.09 Енергія: 4 Прийнято гірших: 38
47: t: 18.89 Енергія: 4 Прийнято гірших: 35
48: t: 18.71 Енергія: 4 Прийнято гірших: 39
49: t: 18.52 Енергія: 4 Прийнято гірших: 35
50: t: 18.33 Енергія: 4 Прийнято гірших: 35
51: t: 18.15 Енергія: 4 Прийнято гірших: 40
52: t: 17.97 Енергія: 4 Прийнято гірших: 43
53: t: 17.79 Енергія: 4 Прийнято гірших: 45
54: t: 17.61 Енергія: 4 Прийнято гірших: 41
55: t: 17.43 Енергія: 4 Прийнято гірших: 39
56: t: 17.26 Енергія: 4 Прийнято гірших: 34
57: t: 17.09 Енергія: 4 Прийнято гірших: 36
58: t: 16.92 Енергія: 4 Прийнято гірших: 43
59: t: 16.75 Енергія: 4 Прийнято гірших: 39
60: t: 16.58 Енергія: 4 Прийнято гірших: 35
61: t: 16.41 Енергія: 4 Прийнято гірших: 43
```

```
62: t: 16.25 Енергія: 4 Прийнято гірших: 37
63: t: 16.09 Енергія: 4 Прийнято гірших: 33
64: t: 15.93 Енергія: 4 Прийнято гірших: 37
65: t: 15.77 Енергія: 4 Прийнято гірших: 35
66: t: 15.61 Енергія: 4 Прийнято гірших: 31
67: t: 15.45 Енергія: 2 Прийнято гірших: 38
68: t: 15.30 Енергія: 2 Прийнято гірших: 39
69: t: 15.15 Енергія: 2 Прийнято гірших: 35
70: t: 15.00 Енергія: 2 Прийнято гірших: 38
71: t: 14.85 Енергія: 2 Прийнято гірших: 39
72: t: 14.70 Енергія: 2 Прийнято гірших: 35
73: t: 14.55 Енергія: 2 Прийнято гірших: 42
74: t: 14.40 Енергія: 2 Прийнято гірших: 39
75: t: 14.26 Енергія: 2 Прийнято гірших: 39
76: t: 14.12 Енергія: 2 Прийнято гірших: 38
77: t: 13.98 Енергія: 2 Прийнято гірших: 39
78: t: 13.84 Енергія: 2 Прийнято гірших: 33
79: t: 13.70 Енергія: 2 Прийнято гірших: 33
80: t: 13.56 Енергія: 2 Прийнято гірших: 36
81: t: 13.43 Енергія: 2 Прийнято гірших: 36
82: t: 13.29 Енергія: 2 Прийнято гірших: 32
83: t: 13.16 Енергія: 2 Прийнято гірших: 37
84: t: 13.03 Енергія: 2 Прийнято гірших: 31
85: t: 12.90 Енергія: 2 Прийнято гірших: 37
86: t: 12.77 Енергія: 2 Прийнято гірших: 35
87: t: 12.64 Енергія: 2 Прийнято гірших: 34
88: t: 12.51 Енергія: 2 Прийнято гірших: 33
89: t: 12.39 Енергія: 2 Прийнято гірших: 35
90: t: 12.26 Енергія: 2 Прийнято гірших: 36
91: t: 12.14 Енергія: 2 Прийнято гірших: 36
92: t: 12.02 Енергія: 2 Прийнято гірших: 36
93: t: 11.90 Енергія: 2 Прийнято гірших: 37
94: t: 11.78 Енергія: 2 Прийнято гірших: 39
95: t: 11.66 Енергія: 2 Прийнято гірших: 32
96: t: 11.55 Енергія: 2 Прийнято гірших: 35
97: t: 11.43 Енергія: 2 Прийнято гірших: 38
98: t: 11.32 Енергія: 2 Прийнято гірших: 35
99: t: 11.20 Енергія: 2 Прийнято гірших: 30
100: t: 11.09 Енергія: 2 Прийнято гірших: 32
101: t: 10.98 Енергія: 2 Прийнято гірших: 40
102: t: 10.87 Енергія: 2 Прийнято гірших: 38
103: t: 10.76 Енергія: 2 Прийнято гірших: 32
104: t: 10.65 Енергія: 2 Прийнято гірших: 37
105: t: 10.55 Енергія: 2 Прийнято гірших: 31
106: t: 10.44 Енергія: 2 Прийнято гірших: 35
107: t: 10.34 Енергія: 2 Прийнято гірших: 31
108: t: 10.23 Енергія: 2 Прийнято гірших: 40
109: t: 10.13 Енергія: 2 Прийнято гірших: 35
110: t: 10.03 Енергія: 2 Прийнято гірших: 35
111: t: 9.93 Енергія: 2 Прийнято гірших: 31
112: t: 9.83 Енергія: 2 Прийнято гірших: 39
113: t: 9.73 Енергія: 2 Прийнято гірших: 31
114: t: 9.64 Енергія: 2 Прийнято гірших: 31
115: t: 9.54 Енергія: 2 Прийнято гірших: 30
116: t: 9.44 Енергія: 2 Прийнято гірших: 35
117: t: 9.35 Енергія: 2 Прийнято гірших: 32
118: t: 9.26 Енергія: 2 Прийнято гірших: 36
119: t: 9.16 Енергія: 2 Прийнято гірших: 30
120: t: 9.07 Енергія: 2 Прийнято гірших: 28
121: t: 8.98 Енергія: 2 Прийнято гірших: 34
122: t: 8.89 Енергія: 2 Прийнято гірших: 41
123: t: 8.80 Енергія: 2 Прийнято гірших: 31
124: t: 8.71 Енергія: 2 Прийнято гірших: 33
125: t: 8.63 Енергія: 2 Прийнято гірших: 31
126: t: 8.54 Енергія: 2 Прийнято гірших: 30
127: t: 8.46 Енергія: 2 Прийнято гірших: 36
128: t: 8.37 Енергія: 2 Прийнято гірших: 32
129: t: 8.29 Енергія: 2 Прийнято гірших: 39
130: t: 8.20 Енергія: 2 Прийнято гірших: 34
131: t: 8.12 Енергія: 2 Прийнято гірших: 33
```

```
132: t: 8.04 Енергія: 2 Прийнято гірших: 32
133: t: 7.96 Енергія: 2 Прийнято гірших: 30
134: t: 7.88 Енергія: 2 Прийнято гірших: 36
135: t: 7.80 Енергія: 2 Прийнято гірших: 35
136: t: 7.72 Енергія: 2 Прийнято гірших: 33
137: t: 7.65 Енергія: 2 Прийнято гірших: 30
138: t: 7.57 Енергія: 2 Прийнято гірших: 39
139: t: 7.50 Енергія: 2 Прийнято гірших: 24
140: t: 7.42 Енергія: 2 Прийнято гірших: 31
141: t: 7.35 Енергія: 2 Прийнято гірших: 33
142: t: 7.27 Енергія: 2 Прийнято гірших: 31
143: t: 7.20 Енергія: 2 Прийнято гірших: 29
144: t: 7.13 Енергія: 2 Прийнято гірших: 32
145: t: 7.06 Енергія: 2 Прийнято гірших: 32
146: t: 6.99 Енергія: 2 Прийнято гірших: 32
147: t: 6.92 Енергія: 2 Прийнято гірших: 29
148: t: 6.85 Енергія: 2 Прийнято гірших: 34
149: t: 6.78 Енергія: 2 Прийнято гірших: 32
150: t: 6.71 Енергія: 2 Прийнято гірших: 30
151: t: 6.64 Енергія: 2 Прийнято гірших: 29
152: t: 6.58 Енергія: 2 Прийнято гірших: 33
153: t: 6.51 Енергія: 2 Прийнято гірших: 27
154: t: 6.45 Енергія: 2 Прийнято гірших: 26
155: t: 6.38 Енергія: 2 Прийнято гірших: 36
156: t: 6.32 Енергія: 2 Прийнято гірших: 32
157: t: 6.25 Енергія: 2 Прийнято гірших: 30
158: t: 6.19 Енергія: 2 Прийнято гірших: 27
159: t: 6.13 Енергія: 2 Прийнято гірших: 33
160: t: 6.07 Енергія: 2 Прийнято гірших: 33
161: t: 6.01 Енергія: 2 Прийнято гірших: 31
162: t: 5.95 Енергія: 2 Прийнято гірших: 30
163: t: 5.89 Енергія: 2 Прийнято гірших: 27
164: t: 5.83 Енергія: 2 Прийнято гірших: 34
165: t: 5.77 Енергія: 2 Прийнято гірших: 26
166: t: 5.71 Енергія: 2 Прийнято гірших: 29
167: t: 5.66 Енергія: 2 Прийнято гірших: 28
168: t: 5.60 Енергія: 2 Прийнято гірших: 27
169: t: 5.54 Енергія: 2 Прийнято гірших: 27
170: t: 5.49 Енергія: 2 Прийнято гірших: 28
171: t: 5.43 Енергія: 2 Прийнято гірших: 23
172: t: 5.38 Енергія: 2 Прийнято гірших: 32
173: t: 5.33 Енергія: 2 Прийнято гірших: 30
174: t: 5.27 Енергія: 2 Прийнято гірших: 30
175: t: 5.22 Енергія: 2 Прийнято гірших: 35
176: t: 5.17 Енергія: 2 Прийнято гірших: 28
177: t: 5.12 Енергія: 2 Прийнято гірших: 33
178: t: 5.06 Енергія: 2 Прийнято гірших: 28
179: t: 5.01 Енергія: 2 Прийнято гірших: 29
180: t: 4.96 Енергія: 2 Прийнято гірших: 21
181: t: 4.91 Енергія: 2 Прийнято гірших: 30
182: t: 4.87 Енергія: 2 Прийнято гірших: 24
183: t: 4.82 Енергія: 2 Прийнято гірших: 25
184: t: 4.77 Енергія: 2 Прийнято гірших: 31
185: t: 4.72 Енергія: 2 Прийнято гірших: 29
186: t: 4.67 Енергія: 2 Прийнято гірших: 32
187: t: 4.63 Енергія: 2 Прийнято гірших: 26
188: t: 4.58 Енергія: 2 Прийнято гірших: 31
189: t: 4.53 Енергія: 2 Прийнято гірших: 32
190: t: 4.49 Енергія: 2 Прийнято гірших: 21
191: t: 4.44 Енергія: 2 Прийнято гірших: 27
192: t: 4.40 Енергія: 2 Прийнято гірших: 29
193: t: 4.36 Енергія: 2 Прийнято гірших: 21
194: t: 4.31 Енергія: 2 Прийнято гірших: 30
195: t: 4.27 Енергія: 2 Прийнято гірших: 24
196: t: 4.23 Енергія: 2 Прийнято гірших: 21
197: t: 4.18 Енергія: 2 Прийнято гірших: 30
198: t: 4.14 Енергія: 2 Прийнято гірших: 23
199: t: 4.10 Енергія: 2 Прийнято гірших: 26
200: t: 4.06 Енергія: 2 Прийнято гірших: 24
201: t: 4.02 Енергія: 2 Прийнято гірших: 23
```

```
202: t: 3.98 Енергія: 2 Прийнято гірших: 26
203: t: 3.94 Енергія: 2 Прийнято гірших: 26
204: t: 3.90 Енергія: 2 Прийнято гірших: 26
205: t: 3.86 Енергія: 2 Прийнято гірших: 26
206: t: 3.82 Енергія: 2 Прийнято гірших: 21
207: t: 3.78 Енергія: 2 Прийнято гірших: 23
208: t: 3.75 Енергія: 2 Прийнято гірших: 25
209: t: 3.71 Енергія: 2 Прийнято гірших: 21
210: t: 3.67 Енергія: 2 Прийнято гірших: 20
211: t: 3.64 Енергія: 2 Прийнято гірших: 24
212: t: 3.60 Енергія: 2 Прийнято гірших: 28
213: t: 3.56 Енергія: 2 Прийнято гірших: 22
214: t: 3.53 Енергія: 2 Прийнято гірших: 22
215: t: 3.49 Енергія: 2 Прийнято гірших: 18
216: t: 3.46 Енергія: 2 Прийнято гірших: 23
217: t: 3.42 Енергія: 2 Прийнято гірших: 20
218: t: 3.39 Енергія: 2 Прийнято гірших: 23
219: t: 3.35 Енергія: 2 Прийнято гірших: 27
220: t: 3.32 Енергія: 2 Прийнято гірших: 19
221: t: 3.29 Енергія: 2 Прийнято гірших: 19
222: t: 3.25 Енергія: 2 Прийнято гірших: 12
223: t: 3.22 Енергія: 2 Прийнято гірших: 25
224: t: 3.19 Енергія: 2 Прийнято гірших: 22
225: t: 3.16 Енергія: 2 Прийнято гірших: 19
226: t: 3.13 Енергія: 2 Прийнято гірших: 23
227: t: 3.10 Енергія: 2 Прийнято гірших: 20
228: t: 3.06 Енергія: 2 Прийнято гірших: 22
229: t: 3.03 Енергія: 2 Прийнято гірших: 18
230: t: 3.00 Енергія: 2 Прийнято гірших: 19
231: t: 2.97 Енергія: 2 Прийнято гірших: 16
232: t: 2.94 Енергія: 2 Прийнято гірших: 19
233: t: 2.91 Енергія: 2 Прийнято гірших: 21
234: t: 2.88 Енергія: 2 Прийнято гірших: 18
235: t: 2.86 Енергія: 2 Прийнято гірших: 21
236: t: 2.83 Енергія: 2 Прийнято гірших: 19
```

236: t: 2.83 Енергія: 2 Прийнято гірших: 19

```
237: t: 2.80 Енергія: 2 Прийнято гірших: 18
238: t: 2.77 Енергія: 2 Прийнято гірших: 20
239: t: 2.74 Енергія: 2 Прийнято гірших: 16
240: t: 2.72 Енергія: 2 Прийнято гірших: 16
241: t: 2.69 Енергія: 2 Прийнято гірших: 15
242: t: 2.66 Енергія: 2 Прийнято гірших: 20
243: t: 2.64 Енергія: 2 Прийнято гірших: 17
244: t: 2.61 Енергія: 2 Прийнято гірших: 11
245: t: 2.58 Енергія: 2 Прийнято гірших: 16
246: t: 2.56 Енергія: 2 Прийнято гірших: 12
247: t: 2.53 Енергія: 2 Прийнято гірших: 21
248: t: 2.51 Енергія: 2 Прийнято гірших: 14
249: t: 2.48 Енергія: 2 Прийнято гірших: 15
250: t: 2.46 Енергія: 2 Прийнято гірших: 15
251: t: 2.43 Енергія: 2 Прийнято гірших: 13
252: t: 2.41 Енергія: 2 Прийнято гірших: 15
253: t: 2.38 Енергія: 2 Прийнято гірших: 15
254: t: 2.36 Енергія: 2 Прийнято гірших: 16
255: t: 2.34 Енергія: 2 Прийнято гірших: 20
256: t: 2.31 Енергія: 2 Прийнято гірших: 11
257: t: 2.29 Енергія: 2 Прийнято гірших: 19
258: t: 2.27 Енергія: 2 Прийнято гірших: 20
259: t: 2.24 Енергія: 2 Прийнято гірших: 12
260: t: 2.22 Енергія: 2 Прийнято гірших: 15
261: t: 2.20 Енергія: 2 Прийнято гірших: 16
262: t: 2.18 Енергія: 2 Прийнято гірших: 12
263: t: 2.16 Енергія: 2 Прийнято гірших: 16
264: t: 2.13 Енергія: 2 Прийнято гірших: 13
265: t: 2.11 Енергія: 2 Прийнято гірших: 11
266: t: 2.09 Енергія: 2 Прийнято гірших: 12
267: t: 2.07 Енергія: 2 Прийнято гірших: 12
268: t: 2.05 Енергія: 2 Прийнято гірших: 15
269: t: 2.03 Енергія: 2 Прийнято гірших: 14
270: t: 2.01 Енергія: 2 Прийнято гірших: 16
271: t: 1.99 Енергія: 2 Прийнято гірших: 16
```

```
271: t: 1.99 Енергія: 2 Прийнято гірших: 16
272: t: 1.97 Енергія: 2 Прийнято гірших: 15
273: t: 1.95 Енергія: 2 Прийнято гірших: 12
274: t: 1.93 Енергія: 2 Прийнято гірших: 16
275: t: 1.91 Енергія: 2 Прийнято гірших: 12
276: t: 1.89 Енергія: 2 Прийнято гірших: 15
277: t: 1.87 Енергія: 2 Прийнято гірших: 8
278: t: 1.85 Енергія: 2 Прийнято гірших: 18
279: t: 1.84 Енергія: 2 Прийнято гірших: 10
280: t: 1.82 Енергія: 2 Прийнято гірших: 8
281: t: 1.80 Енергія: 2 Прийнято гірших: 13
282: t: 1.78 Енергія: 2 Прийнято гірших: 12
283: t: 1.76 Енергія: 2 Прийнято гірших: 12
284: t: 1.75 Енергія: 2 Прийнято гірших: 12
285: t: 1.73 Енергія: 2 Прийнято гірших: 14
286: t: 1.71 Енергія: 2 Прийнято гірших: 9
287: t: 1.69 Енергія: 2 Прийнято гірших: 9
288: t: 1.68 Енергія: 2 Прийнято гірших: 7
289: t: 1.66 Енергія: 2 Прийнято гірших: 8
290: t: 1.64 Енергія: 2 Прийнято гірших: 7
291: t: 1.63 Енергія: 2 Прийнято гірших: 12
292: t: 1.61 Енергія: 2 Прийнято гірших: 9
293: t: 1.59 Енергія: 2 Прийнято гірших: 7
294: t: 1.58 Енергія: 2 Прийнято гірших: 10
295: t: 1.56 Енергія: 2 Прийнято гірших: 10
296: t: 1.55 Енергія: 2 Прийнято гірших: 7
297: t: 1.53 Енергія: 2 Прийнято гірших: 9
298: t: 1.52 Енергія: 2 Прийнято гірших: 2
299: t: 1.50 Енергія: 2 Прийнято гірших: 9
300: t: 1.49 Енергія: 2 Прийнято гірших: 6
301: t: 1.47 Енергія: 2 Прийнято гірших: 13
302: t: 1.46 Енергія: 2 Прийнято гірших: 13
303: t: 1.44 Енергія: 2 Прийнято гірших: 7
304: t: 1.43 Енергія: 2 Прийнято гірших: 8
305: t: 1.41 Енергія: 2 Прийнято гірших: 11
306: t: 1.40 Енергія: 2 Прийнято гірших: 10
307: t: 1.39 Енергія: 2 Прийнято гірших: 5
308: t: 1.37 Енергія: 2 Прийнято гірших: 14
309: t: 1.36 Енергія: 2 Прийнято гірших: 6
310: t: 1.34 Енергія: 2 Прийнято гірших: 6
311: t: 1.33 Енергія: 2 Прийнято гірших: 7
312: t: 1.32 Енергія: 2 Прийнято гірших: 6
313: t: 1.30 Енергія: 2 Прийнято гірших: 9
314: t: 1.29 Енергія: 2 Прийнято гірших: 7
315: t: 1.28 Енергія: 2 Прийнято гірших: 10
316: t: 1.27 Енергія: 2 Прийнято гірших: 11
317: t: 1.25 Енергія: 2 Прийнято гірших: 9
318: t: 1.24 Енергія: 2 Прийнято гірших: 5
319: t: 1.23 Енергія: 2 Прийнято гірших: 4
320: t: 1.22 Енергія: 2 Прийнято гірших: 3
321: t: 1.20 Енергія: 2 Прийнято гірших: 7
322: t: 1.19 Енергія: 2 Прийнято гірших: 4
323: t: 1.18 Енергія: 2 Прийнято гірших: 2
324: t: 1.17 Енергія: 2 Прийнято гірших: 10
325: t: 1.16 Енергія: 2 Прийнято гірших: 2
326: t: 1.14 Енергія: 2 Прийнято гірших: 7
327: t: 1.13 Енергія: 2 Прийнято гірших: 4
328: t: 1.12 Енергія: 2 Прийнято гірших: 3
329: t: 1.11 Енергія: 2 Прийнято гірших: 6
330: t: 1.10 Енергія: 2 Прийнято гірших: 3
331: t: 1.09 Енергія: 2 Прийнято гірших: 4
332: t: 1.08 Енергія: 2 Прийнято гірших: 3
333: t: 1.07 Енергія: 2 Прийнято гірших: 2
334: t: 1.06 Енергія: 2 Прийнято гірших: 5
335: t: 1.05 Енергія: 2 Прийнято гірших: 4
336: t: 1.03 Енергія: 2 Прийнято гірших: 0
337: t: 1.02 Енергія: 2 Прийнято гірших: 5
338: t: 1.01 Енергія: 2 Прийнято гірших: 4
339: t: 1.00 Енергія: 2 Прийнято гірших: 6
340: t: 0.99 Енергія: 2 Прийнято гірших: 2
341: t: 0.98 Енергія: 2 Прийнято гірших: 4
```

```
342: t: 0.97 Енергія: 2 Прийнято гірших: 4
343: t: 0.96 Енергія: 2 Прийнято гірших: 4
344: t: 0.95 Енергія: 2 Прийнято гірших: 2
345: t: 0.95 Енергія: 2 Прийнято гірших: 2
346: t: 0.94 Енергія: 2 Прийнято гірших: 5
347: t: 0.93 Енергія: 2 Прийнято гірших: 2
348: t: 0.92 Енергія: 2 Прийнято гірших: 0
349: t: 0.91 Енергія: 2 Прийнято гірших: 1
350: t: 0.90 Енергія: 2 Прийнято гірших: 4
351: t: 0.89 Енергія: 2 Прийнято гірших: 2
352: t: 0.88 Енергія: 2 Прийнято гірших: 0
353: t: 0.87 Енергія: 2 Прийнято гірших: 4
354: t: 0.86 Енергія: 2 Прийнято гірших: 0
355: t: 0.86 Енергія: 0 Прийнято гірших: 0
. . . . . o . . . . . . . . . . . . . .
. . . . . . . . o . . . . . . . . . . .
. . . . . . . . . . . . . o . . . . . .
. o . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . o . .
. . . . . . . . . . . . . . . . . . . o
. . . . . . . . . . . . o . . . . . . .
. . . . . . . . . o . . . . . . . . . .
o . . . . . . . . . . . . . . . . . . .
. . . . o . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . o .
. . . . . . . . . . . . . . . o . . . .
. . o . . . . . . . . . . . . . . . . .
. . . . . . . o . . . . . . . . . . . .
. . . . . . . . . . . o . . . . . . . .
. . . . . . . . . . . . . . . . o . . .
. . . o . . . . . . . . . . . . . . . .
. . . . . . o . . . . . . . . . . . . .
. . . . . . . . . . . . . . o . . . . .
. . . . . . . . . . o . . . . . . . . .
Енергія(кіл. конфліктів): 0
```

**п. 4. Задача комівояжера. Мурашиний алгоритм**

Мураху не можна назвати кмітливим. Окремий мураха не в змозі прийняти жодного рішення. Справа в тому, що він влаштований вкрай примітивно: всі його дії зводяться до елементарних реакцій на навколишнє оточення і своїх побратимів. Мураха не здатний аналізувати, робити висновки і шукати рішення.

Ці факти, однак, ніяк не узгоджуються з успішністю мурах як виду. Вони існують на планеті понад 100 мільйонів років, будують величезні житла, забезпечують їх усім необхідним і навіть ведуть справжні війни. У порівнянні з повною безпорадністю окремих особин, досягнення мурах здаються немислимими.

Домогтися таких успіхів мурахи здатні завдяки своїй соціальності. Вони живуть тільки в колективах – колоніях. Всі мурахи колонії формують так званий ройовий інтелект. Особи, що становлять колонію, не повинні бути розумними: вони повинні лише взаємодіяти по певним – вкрай простим – правилам, і тоді колонія цілком буде ефективна.

У колонії немає домінуючих особин, немає начальників і підлеглих, немає лідерів, які роздають вказівки і координують дії. Колонія повністю самоорганізовується. Кожен з мурах володіє інформацією тільки про локальну обстановку, не один з них не має уявлення про всю ситуації в цілому – тільки про те, що дізнався сам або від своїх родичів, явно чи неявно. На неявних взаємодіях мурах, що називаються стігмергіями, засновані механізми пошуку найкоротшого шляху від мурашника до джерела їжі.

Кожен раз проходячи від мурашника до їжі і назад, мурахи залишають за собою доріжку феромонів. Інші мурахи, відчувши такі сліди на землі, будуть інстинктивно прагнути до нього. Оскільки ці мурахи теж залишають за собою доріжки феромонів, то чим більше мурах проходить певним шляхом, тим привабливішим він стає для їх родичів. При цьому, чим коротше шлях до джерела їжі, тим менше часу потрібно мурашкам на нього – а отже, тим швидше залишені на ньому сліди стають помітними. [5]

У 1992 році в своїй дисертації Марко Дориго (Marco Dorigo) запропонував запозичувати описаний природний механізм для розв’язання задач оптимізації [5]. Імітуючи поведінку колонії мурах в природі, мурашині алгоритми використовують багатоагентні системи, агенти яких функціонують по вкрай простим правилам. Вони вкрай ефективні при вирішенні складних комбінаторних задач – таких, наприклад, як задача комівояжера, перша з вирішених з використанням даного типу алгоритмів.

**Застосування мурашиних алгоритмів для задачі комівояжера.**

Завдача формулюється як задача пошуку мінімального за вартістю замкнутого маршруту по всіх вершинах без повторень на повному зваженому графі з $n$ вершинами. Змістовно вершини графа є містами, які повинен відвідати комівояжер, а ваги ребер відображають відстані (довжини) або вартості проїзду. Це задача є NP-важкою, і точний переборний алгоритм її рішення має факторіальну складність. [5]

Моделювання поведінки мурах пов'язано з розподілом феромону на стежці – ребрі графа в задачі комівояжера. При цьому ймовірність включення ребра в маршрут окремого мурашки пропорційна кількості феромону на цьому ребрі, а кількість феромона, що відкладали мурахи, пропорційна довжині маршруту. Чим коротше маршрут, тим більше феромона буде відкладено на його ребрах, отже, більша кількість мурах буде включати його в синтез власних маршрутів. Моделювання такого підходу, що використовує тільки позитивний зворотний зв'язок, призводить до передчасної збіжності – більшість мурашок рухається локально-оптимальним маршрутом. Уникнути цього можна, моделюючи негативний зворотний зв'язок у вигляді випаровування феромону. При цьому якщо феромон випаровується швидко, то це призводить до втрати пам'яті колонії і забування хороших рішень, з іншого боку, великий час випаровування може привести до отримання стійкого локального оптимального рішення. [5]

Тепер з урахуванням особливостей задачі комівояжера, ми можемо описати локальні правила поведінки мурах при виборі шляху.

1. Мурахи мають власну «пам'ять». Оскільки кожне місто може бути відвідане тільки один раз, то у кожного мурашки є список вже відвіданих міст – список заборон. Позначимо через $j$ $i$, $k$ список міст, які необхідно відвідати мурашки $k$, що знаходиться в місті $i$.

2. Мурахи володіють «зором» – видимість, евристичне бажання відвідати місто $j$, якщо мураха знаходиться в місті $i$. Будемо вважати, що видимість обернено пропорційна відстані між містами

$$n_{ij} = \frac{1}{D_{ij}}$$

3. Мурахи володіють «нюхом» – вони можуть вловлювати слід феромону, що підтверджує бажання відвідати місто $j$ з міста $i$ на підставі досвіду інших мурах. Кількість феромону на ребрі *(i, j)* в момент часу $t$ позначимо через $\tau_{ij}(t)$. [5]

На цій підставі ми можемо сформулювати ймовірнісно-пропорційне правило, що визначає ймовірність переходу $k$-ого мурашки з міста $i$ в місто $j$:

$$\begin{cases} P_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \epsilon J_{ik}} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}}, \ j \epsilon J_{i,k} \\ P_{ij}(t) = 0, j \notin J_{i,k} \end{cases} \quad (1)$$

Де $\alpha$, $\beta$ – параметри, що задають ваги сліду феромону. При $\alpha = 0$ алгоритм вироджується до жадного алгоритму (буде обране найближче місто). Зауважимо, що вибір міста є імовірнісним, правило (1) лише визначає ширину зони міста $j$; в загальну зону всіх міст кидається випадкове число, яке і визначає вибір мурашки. Правило (1) не змінюється в ході алгоритму, але у двох різних мурах значення ймовірності переходу будуть відрізнятися, тому що вони мають різний список дозволених міст.

5. Пройшовши ребро ($i$, $j$), мураха відкладає на ньому деяку кількість феромону, яке повинно бути пов'язано з оптимальністю зробленого вибору. Нехай $T_k(t)$ є маршрут, пройдений мурахою $k$ до моменту часу $t$, $L_k(t)$ – довжина цього маршруту, а $Q$ – параметр, що має значення порядку довжини оптимального шляху. Тоді кількість феромону, що відкладається може бути задано у вигляді [7]

$$\triangle \tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, (i,j) \in T_k(t) \\ 0, (i,j) \notin T_k(t) \end{cases}$$

Правила зовнішнього середовища визначають, в першу чергу, випаровування феромону. Нехай $\rho \in [0,1]$ коефіцієнт випаровування, тоді правило випаровування має вигляд [6]

$$\tau_{ij}(t+!) = (1-\rho) * \tau_{ij}(t) + \Delta\tau_{ij}(t);$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}(t), \quad (2)$$

де $m$ - кількість мурах в колонії.

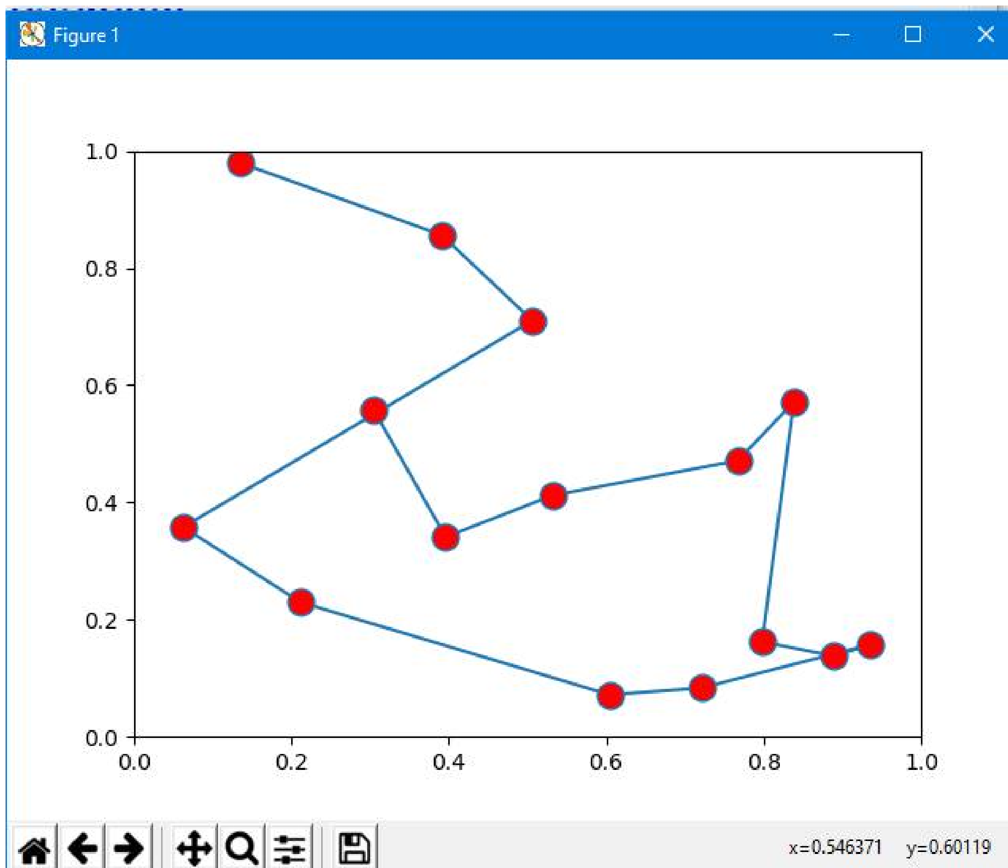
На початку алгоритму кількість феромону на ребрах приймається рівною невеликому позитивному числу. Загальна кількість мурах залишається

постійною і рівною кількості міст, кожен мураха починає маршрут зі свого міста.

Додаткова модифікація алгоритму може складатися у веденні так званих «елітних» мурах, які підсилюють ребра найкращого маршруту, знайденого з початку роботи алгоритму. Позначимо через $T^*$ найкращий поточний маршрут, через $L^*$ – його довжину. Тоді якщо в колонії є $e$ елітних мурах, то ребра маршруту отримають додаткову кількість феромону

$$\Delta\tau_e = e^* \frac{Q}{L^*}$$

Програмна реалізація мурашиного алгоритму (див. додаток 6)



## Висновки

Задачі дискретної оптимізації  знаходять широке застосування в різних областях, де використовуються математичні методи для аналізу процесів, що там відбуваються. Необхідність розв’язання таких задач призводить до того, що дискретна оптимізація стає важливим елементом освіти фахівців, пов'язаних з її застосуванням при вирішенні задач, що виникають. Тому технологія розв’язання задач дискретного програмування повинна стати однією з важливих складових частин сучасної математичної освіти для фахівців з прикладної математики. В даний час розроблені сучасні методи і алгоритми розв’язання задач дискретного програмування. Розроблено пакети прикладних програм, що дозволяють вирішувати ряд стандартних задач дискретного програмування. Теоретичні знання алгоритмів і технологій їх реалізації дозволяє більш ефективно використовувати розроблені пакети. При виникненні нових нестандартних задач реалізація алгоритмів їх розв’язання вимагає інформації про технології розв’язання задач дискретної оптимізації. Для вивчення матеріалу необхідні знання основ математичного аналізу, лінійної алгебри, лінійного програмування і основ теорії графів.

У роботі розглянуті деякі задачі дискретної оптимізації (задача пакування рюкзака, задача комівояжера, задача оптимального розкрою, задача про покриття множини системою її підмножин). Був проведений аналіз ефективності таких алгоритмів розв’язання задачі дискретної оптимізації – задачі коммівояжера, як метод найближчого сусіда та мурашиний алгоритм. Також були розроблені деякі модифікації даних алгоритмів. Усі програмні реалізації алгоритмів розроблені на мові програмування Python.

## Список використаної літератури

1. Наконечний С. І., Савіна С. С. Математичне програмування: Навч. посіб.– К.: КНЕУ, 2003.– 452 с.

2. Попов Ю.Д., Тюптя В.І., Шевченко В.І. Методи оптимізації. Навчальний електронний посібник.− Київ: Електр. видання. Ел. бібліотека факультету кібернетики КНУТШ, 2003.− 215 с.

3. М. Тим Джонс. Программирование искусственного интеллекта в приложениях. – ДМК Пресс, 2011 – 387 с.

4. Стивен С. Скиена Алгоритмы. Руководство по разработке— 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург. 2011. — 720 с.

5. Штовба С. Д. Муравьиные алгоритмы. Математика в приложениях, 2003, №4, стр. 70-75.

6. МакКоннелл Дж. Основы современных алгоритмов. – М.: Техносфера, 2004. – 368 с.

7. Столлингс В. Современные компьютерные сети. 2-е изд. - СПб: Питер, 2003. – 783с.

8. Yurchenko I.V., Blyacher D.V. Research of discrete optimization problems on computer // International conference "The Future of Mankind in the Results of Today's Scientific Research '2019" (Ukraine, Odessa, 11-12 November, 2019).– [Електронний ресурс] https://sworld.education/index.php/ru/secciiua15-6/31768

9. J. Bang-Jensen, G. Gutin and A. Yeo, When the greedy algorithm fails. Discrete Optimization 1 (2004), 121–127 с.

10. G. Bendall and F. Margot, Greedy Type Resistance of Combinatorial Problems, Discrete Optimization 3 (2006), 288–298 с.

11. Юрченко І.В., Бляхер Д.В. Дослідження задач дискретної оптимізації на комп'ютері // Научный взгляд в будущее.– Вып.15. Том 1.– Одесса: Куприенко СВ, 2019.– С. 82-88. DOI: 10.30888/2415-7538.2019-15-01-005 https://www.scilook.eu/index.php/slif/article/view/slif15-005

# Додатки

## Додаток 1

```
from collection import namedtuple
from functools import 1ru_cache
items = namedtuple('Item', 'value weight')
items = Item(4,5), Item(3,4), Item(3,2), Item(2,1)
capacity = 6 #max weight we can put into the knapsack
@1ru_cache(maxsize=None) # cache all calls
def best_value(nitems, weight_limit):
    if nitems == 0: # no items
        return 0
    elif items[nitems - 1].weight > weight_limit:
        # new item is heavier than the current weight limit
        return best_value(nitems – 1, weight_limit) # don`t include new item
else:
        return max( # max of with and without the new item
                best_value(nitems – 1, weight_limit), # without
                best_value(nitems – 1, weight_limit – items[nitems - 1], weight)
                + items[nitems -1].value) # with the new item
result = []
weight_limit = capacity
for i in reversed(range(len(items))):
        if best_value(i+1, weight_limit) > best_value(i, weight_limit):
        # better with the i-th item
        result append(items[i]) # include it in the result
        weight_limit - = items[i].weight
print(result)
print(best_value.cache_info())
```

## Додаток 2

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import exp,sqrt
n=50;m=100;ib=3;way=[];a=0
X=np.random.uniform(a,m,n)
Y=np.random.uniform(a,m,n)
M = np.zeros([n,n]) # шаблон матриці відносних відстаней між пунктами
for i in np.arange(0,n,1):
for j in np.arange(0,n,1):
if i!=j:
            M[i,j]=sqrt((X[i]-X[j])**2+(Y[i]-Y[j])**2)# заповнення матриці
        else:
            M[i,j]=float('inf')# заповнення головної діагоналі матриці
way.append(ib)
for i in np.arange(1,n,1):
s=[]
for j in np.arange(0,n,1):
        s.append(M[way[i-1],j])
        way.append(s.index(min(s)))# індекси пунктів ближчих міст сусідів
for j in np.arange(0,i,1):
        M[way[i],way[j]]=float('inf')
        M[way[i],way[j]]=float('inf')
S=sum([sqrt((X[way[i]]-X[way[i+1]])**2+(Y[way[i]]-Y[way[i+1]])**2)
for i in np.arange(0,n-1,1)])+ sqrt((X[way[n-1]]-X[way[0]])**2+(Y[way[n-1]]-Y[way[0]])**2)
plt.title('Загальний шлях-%s.Номер міста-%i.Всього міст -%i.\n Координати X,Y
випадкові числа від %i до %i'%(round(S,3),ib,n,a,m), size=14)
X1=[X[way[i]] for i in np.arange(0,n,1)]
Y1=[Y[way[i]] for i in np.arange(0,n,1)]
plt.plot(X1, Y1, color='r', linestyle=' ', marker='o')
plt.plot(X1, Y1, color='b', linewidth=1)
X2=[X[way[n-1]],X[way[0]]]
Y2=[Y[way[n-1]],Y[way[0]]]
plt.plot(X2, Y2, color='g', linewidth=2, linestyle='-', label=Шлях від останнього \n до
першого міста')
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

## Додаток 3

```
import matplotlib.pyplot as plt
import numpy as np from numpy
import exp,sqrt
n=50;m=100;way=[];a=0
X=np.random.uniform(a,m,n)
Y=np.random.uniform(a,m,n)
X=[10, 10, 100,100 ,30, 20, 20, 50, 50, 85, 85, 75, 35, 25, 30, 47, 50]
Y=[5, 85, 0,90,50, 55,50,75 ,25,50,20,80,25,70,10,50,100]
n=len(X)
RS=[]; RW=[]; RIB=[]
s=[]
for ib in np.arange(0,n,1):
    M = np.zeros([n,n])
    for i in np.arange(0,n,1):
        for j in np.arange(0,n,1):
            if i!=j:
                M[i,j]=sqrt((X[i]-X[j])**2+(Y[i]-Y[j])**2) else:
                M[i,j]=float('inf')
    way=[]
    way.append(ib)
    for i in np.arange(1,n,1):
        s=[]
        for j in np.arange(0,n,1):
            s.append(M[way[i-1],j])
            way.append(s.index(min(s)))
                for j in np.arange(0,i,1):
                    M[way[i],way[j]]=float('inf')
                    M[way[i],way[j]]=float('inf')
    S=sum([sqrt((X[way[i]]-X[way[i+1]])**2+(Y[way[i]]-Y[way[i+1]])**2)  for  i  in
np.arange(0,n-1,1)])+ sqrt((X[way[n-1]]-X[way[0]])**2+(Y[way[n-1]]-Y[way[0]])**2)
    RS.append(S)
    RW.append(way)
    RIB.append(ib)
S=min(RS)
way=RW[RS.index(min(RS))]
ib=RIB[RS.index(min(RS))]
X1=[X[way[i]] for i in np.arange(0,n,1)]
Y1=[Y[way[i]] for i in np.arange(0,n,1)]
plt.title(Загальний шлях-%s.Номер міста-%i.Всього міст -%i.\n Координати X,Y
задані'%(round(S,3),ib,n), size=14)
plt.plot(X1, Y1, color='r', linestyle=' ', marker='o')
plt.plot(X1, Y1, color='b', linewidth=1)
X2=[X[way[n-1]],X[way[0]]]
Y2=[Y[way[n-1]],Y[way[0]]]
```

```
plt.plot(X2, Y2, color='g', linewidth=2, linestyle='-', label=Шлях від останнього \n до
першого міста)
plt.legend(loc='best')
plt.grid(True)
plt.show()
Z=sqrt((X[way[n-1]]-X[way[0]])**2+(Y[way[n-1]]-Y[way[0]])**2)
Y3=[sqrt((X[way[i+1]]-X[way[i]])**2+(Y[way[i+1]]-Y[way[i]])**2) for i in np.arange(0,n-1,1)]
X3=[i for i in np.arange(0,n-1,1)]
plt.title(Шляхи від міста до міста)
plt.plot(X3, Y3, color='b', linestyle=' ', marker='o')
plt.plot(X3, Y3, color='r', linewidth=1, linestyle='-', label=Без врахування шляху, що
замикає коло - %s'%str(round(Z,3)))
plt.legend(loc='best')
plt.grid(True)
plt.show ()
```

**Додаток 4**

```
import matplotlib.pyplot as plt
def mnkLIN(x,y):
        a=round((len(x)*sum([x[i]*y[i]
        for i in range(0,len(x))])-sum(x)*sum(y))/(len(x)*sum([x[i]**2
        for i in range(0,len(x))])-sum(x)**2),3)
        b=round((sum(y)-a*sum(x))/len(x) ,3)
        y1=[round(a*w+b ,3) for w in x]
        s=[round((y1[i]-y[i])**2,3)
        for i in range(0,len(x))]
        sko=round((sum(s)/(len(x)-1))**0.5,3)
        p=(sko*len(x)*100)/sum(y1)
        plt.title('Апроксимація методом найменших \n квадратів Y=%s*x+%s с
похибкою -%i -відсотків'%(str(a),str(b),int(p)), size=14)
        plt.xlabel('Кількість міст, size=14)
        plt.ylabel('Довжина маршрутів', size=14)
        plt.plot(x, y, color='r', linestyle=' ', marker='o', label='Дані методу найближчого
сусіда')
        plt.plot(x, y1, color='b',linewidth=1, label='Апроксимуюча пряма')
        plt.legend(loc='best')
        plt.grid(True)
        plt.show()
y=[933.516, 1282.842, 1590.256, 1767.327 ,1949.975, 2212.668, 2433.955, 2491.954,
2549.579, 2748.672]
x=[100,200,300,400,500,600, 700,800, 900,1000]
mnkLIN(x,y)
```

**Додаток 5**

```
# -*- coding: utf8 -*-
#
# Алгоритм відпалу
# Задача про розстановку ферзів
#
# Simulated Annealing Implementation for the n-Queens Problem
from copy import copy
from math import exp
import random

MAX_LENGTH = 20  # Розмір поля
INITIAL_TEMPERATURE = 30.  # Початкова температура
FINAL_TEMPERATURE = 0.5  # Кінцева температура
ALPHA = 0.99  # Коефіцієнт зміни температури
STEPS_PER_CHANGE = 100  # Кількість інтерацій перед зміною температури
class Board(list):
    """
    Класс поля
    """
    def __init__(self, solution=None):
        super(Board, self).__init__()
        for i in range(MAX_LENGTH):
            self.append([0] * MAX_LENGTH)
        if solution:
            self.apply(solution)
    def apply(self, solution):
        for x in range(MAX_LENGTH):
            y = solution[x]
            self[x][y] = 1
    def draw(self):
        for row in self:
            for i in row:
                if i:
                    print('o', end=" ")
                else:
                    print('.', end=" ")
            print
class Solution(list):
    """
    Варіант розв'язку. Список довжиною MAX_LENGTH.
    Кожен елемент вказує строку розміщення ферзя в і-й колонці.
    """
    dx = (-1, 1, -1, 1)
    dy = (-1, 1, 1, -1)
```

```
    def __init__(self):
        self.energy = 0
        super(Solution, self).__init__(range(MAX_LENGTH))
        random.shuffle(self)
        self.calculate_energy()
    def tweak(self):
        """
        Міняємо місцями дві колонки для отримання нового розв'язку.
        """
        i1 = random.randint(0, MAX_LENGTH - 1)
        i2 = random.randint(0, MAX_LENGTH - 1)
        while i1 == i2:
            i2 = random.randint(0, MAX_LENGTH - 1)
        self[i1], self[i2] = self[i2], self[i1]
        self.calculate_energy()
    def calculate_energy(self):
        board = Board(self)
        conflicts = 0
        for x in range(MAX_LENGTH):
            y = self[x]
            # Провіряємо діагоналі
            for i in range(4):
                tx, ty, dx, dy = x, y, self.dx[i], self.dy[i]
                while True:
                    tx += dx
                    ty += dy
                    if tx < 0 or ty < 0 or tx >= MAX_LENGTH or ty >= MAX_LENGTH:
                        break
                    if board[tx][ty]:
                        conflicts += 1
        self.energy = conflicts
if __name__ == "__main__":
    print(u'Приклад розв’язання задачі розміщення ферзів на шахматному полі без конфліктів')
    print(u'методом відпалу')
    print()
    print(u'Розмір поля:', MAX_LENGTH)
    print(u'Початкова температура:', INITIAL_TEMPERATURE)
    print(u'Кінцева температура:', FINAL_TEMPERATURE)
    print(u'Коефіцієнт зміни температури:', ALPHA)
    print(u'Кількість иерацій перед зміною температури:', STEPS_PER_CHANGE)
    print()
    print(u'NOTE: При обчисленні відображається поточна ітерація, температура, ')
```

```
        print(u'енергія(кіл. конфліктів) и кіл. принятих гірших рішень')
        #raw_input(u'<ENTER>')
        best = Solution()
        temperature = INITIAL_TEMPERATURE
        timer = 0
        while temperature > FINAL_TEMPERATURE:
            timer += 1
            accepted = 0  # Кіл. принятих гірших рішень
            current = copy(best)
            working = copy(current)
            for i in range(STEPS_PER_CHANGE):
                use_new = False
                working.tweak()
                if working.energy <= current.energy:
                    use_new = True
                    if working.energy < best.energy:
                        best = copy(working)
                else:
                    delta = working.energy - current.energy
                    if random.random() < exp(-delta / temperature):
                        accepted += 1
                        use_new = True
                if use_new:
                    current = copy(working)
                else:
                    working = copy(current)
            print( '%s: t: %.02f Енергія: %s Прийнято гірших: %s' % (timer, temperature,
best.energy, accepted))
            if best.energy == 0:
                break
            temperature *= ALPHA
        board = Board(best)
        board.draw()
        print( 'Енергія(кіл. конфліктів): %s' % best.energy)
```

## Додаток 6

```
# -*- coding: utf8 -*-
#
# Ant Algorithm Demonstration
#
# Задача комівояжера
# Мурашиний алгоритм

from math import sqrt
from random import random
import matplotlib.lines as lines
import matplotlib.pyplot as plt
import numpy as np

MAX_CITIES = 15
MAX_DISTANCE = 1
MAX_TIME = 500 * MAX_CITIES
INIT_PHEROMONE = 1.0 / MAX_CITIES

MAX_ANTS = MAX_CITIES * MAX_CITIES
ALPHA = 1  # вага ферменту
BETA = 5  # коефіцієнт эвристики, вплив апріорних знань(1/d, де d - відстань)
RHO = .5  # Інтенсивність. Коеф. випаровування рівен 1 - RHO. По результатам
тестів краще використовувати >= 0.5
QVAL = 100  # Кіл. феромонів на один прохід

class City(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

class Ant(object):

    def __init__(self, start_city):
        self.cur_city = start_city
        self.path = [start_city]
        self.tour_length = 0.

    def move_to_city(self, city):
        global DISTANCE, MAX_CITIES
        self.path.append(city)
        self.tour_length += DISTANCE[self.cur_city][city]
        if len(self.path) == MAX_CITIES:
            self.tour_length += DISTANCE[self.path[-1]][self.path[0]]
```

```
        self.cur_city = city

    def can_move(self):
        global MAX_CITIES
        return len(self.path) < MAX_CITIES

    def reset(self, city):
        self.cur_city = city
        self.path = [city]
        self.tour_length = 0.

def get_random(l):
    r = random()
    cur_probability = 0
    cur_val = None

    for val, probability in l:
        cur_val = val
        cur_probability += probability
        if r <= cur_probability:
            break

    return cur_val


ANTS = []  # [MAX_ANTS]
CITIES = []  # [MAX_CITIES]
DISTANCE = []  # [MAX_CITIES][MAX_CITIES]
PHEROMONE = []  # [MAX_CITIES][MAX_CITIES]
BEST = MAX_CITIES * MAX_DISTANCE
BEST_ANT = None


def init():
    global DISTANCE, PHEROMONE, CITIES, ANTS

    for i in range(MAX_CITIES):
        DISTANCE.append([0.] * MAX_CITIES)
        PHEROMONE.append([INIT_PHEROMONE] * MAX_CITIES)

    for i in range(MAX_CITIES):
        CITIES.append(City(random() * MAX_DISTANCE, random() * MAX_DISTANCE))

    # calculate distance
    for i in range(MAX_CITIES):
```

```
        for j in range(MAX_CITIES):
            if i != j and DISTANCE[i][j] == 0.:
                xd = CITIES[i].x - CITIES[j].x
                yd = CITIES[i].y - CITIES[j].y
                distance = sqrt(xd * xd + yd * yd)
                DISTANCE[i][j], DISTANCE[j][i] = distance, distance

    # create ants
    to = 0
    for i in range(MAX_ANTS):
        ANTS.append(Ant(to))
        to += 1
        to = to % MAX_CITIES

def ant_product(from_city, to_city, ph=None):
    global DISTANCE, PHEROMONE, ALPHA, BETA
    ph = ph or PHEROMONE[from_city][to_city]
    return (ph ** ALPHA) * \
         ((1. / DISTANCE[from_city][to_city]) ** BETA)

def select_next_city(ant):
    global MAX_CITIES, PHEROMONE, DISTANCE
    denom = 0.
    not_visited = []

    for to in range(MAX_CITIES):
        if to not in ant.path:
            ap = ant_product(ant.cur_city, to)
            not_visited.append((to, ap))
            denom += ap

    assert not_visited
    not_visited = [(val, ap / denom) for (val, ap) in not_visited]
    to = get_random(not_visited)
    return to
    i = 0
    while True:
        to, ap = not_visited[i]
        p = ap / denom
        if random() < p:
            break
        i += 1
        i = i % len(not_visited)

    if False and len(not_visited) == MAX_CITIES - 1:
```

```
        for to_city, ap in not_visited:
            print ('%i %.03f %.01f %.02f' % (to_city, PHEROMONE[ant.cur_city][to_city],
DISTANCE[ant.cur_city][to_city], ap / denom))
        print() #to
        raw_input()
    assert ant.cur_city != to
    return to


def simulate_ants():
    global ANTS, MAX_CITIES
    moving = 0

    for ant in ANTS:
        if ant.can_move():
            ant.move_to_city(select_next_city(ant))
            moving += 1

    return moving


def update_trails():
    global MAX_CITIES, PHEROMONE, RHO, INIT_PHEROMONE, ANTS

    # add new pheromone
    for ant in ANTS:
        pheromove_amount = QVAL / ant.tour_length

        for i in range(MAX_CITIES):
            if i == MAX_CITIES - 1:
                from_city = ant.path[i]
                to_city = ant.path[0]
            else:
                from_city = ant.path[i]
                to_city = ant.path[i + 1]
            assert from_city != to_city
            PHEROMONE[from_city][to_city] = PHEROMONE[from_city][to_city] * (1 - RHO)
+ pheromove_amount
            PHEROMONE[to_city][from_city] = PHEROMONE[from_city][to_city]


def restart_ants():
    global ANTS, BEST, BEST_ANT, MAX_CITIES
    to = 0
```

```
    for ant in ANTS:
        if ant.tour_length < BEST:
            BEST = ant.tour_length
            BEST_ANT = ant

        ant.reset(to)
        to += 1
        to = to % MAX_CITIES

if __name__ == '__main__':
    init()
    cur_time = 0
    while cur_time < MAX_TIME:
        cur_time += 1
        if cur_time % 100 == 0:
            print('time:', cur_time, 'of', MAX_TIME)

        if simulate_ants() == 0:
            update_trails()
            cur_time != MAX_TIME and restart_ants()

    import pprint
    pprint.pprint(PHEROMONE)

    x, y = [], []
    for i in BEST_ANT.path:
        city = CITIES[i]
        x.append(city.x / float(MAX_DISTANCE))
        y.append(city.y / float(MAX_DISTANCE))

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    x = np.array(x)
    y = np.array(y)
    line = lines.Line2D(x, y, mfc='red', ms=12, marker='o')
    ax.add_line(line)
    plt.show()
```