

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»  
Навчально-науковий інститут прикладного системного аналізу  
Кафедра системного проєктування**

**Звіт  
про виконання лабораторної роботи №4  
з дисципліни «Методи оптимізації»  
“Дослідження методів вирішення задачі комівояжера”**

**Виконав:  
студент 2 курсу, групи КН-32  
Сафонов Дмитро  
Володимирович**

**Київ – 2025**

## Варіант 20

**Мета роботи:** Ознайомитись і дослідити методи геометричної комбінаторної оптимізації NP-повних задач на прикладі задачі комівояжера. Набути навичок практичної реалізації евристичних та метаевристичних методів, провести експерименти та порівняння реалізованих алгоритмів.

**Хід роботи:**

### 1. Підготувати вхідні дані:

```
import random
import matplotlib.pyplot as plt
import csv

class Town:
    def __init__(self, x, y):
        self.location = [x, y]

    def print(self):
        print(f"location: [{self.location[0]}, {self.location[1]}]")

X_RANGE = (-5000, 5000)
Y_RANGE = (-5000, 5000)

towns = []

for i in range(5000):
    x = random.randint(*X_RANGE)
    y = random.randint(*Y_RANGE)

    town = Town(x, y)
    towns.append(town)

# візуалізація
x_vals = [t.location[0] for t in towns]
y_vals = [t.location[1] for t in towns]

plt.scatter(x_vals, y_vals, s=1)
plt.title("Генеровані міста")
plt.xlabel("X координата")
plt.ylabel("Y координата")
plt.grid(True)
plt.show()

# Запис до файлу
with open("cities1.csv", mode="w", newline="") as file:
    writer = csv.writer(file, delimiter=';')
    writer.writerow(["x", "y"])
    for town in towns:
        writer.writerow([town.location[0], town.location[1]])
```

Генеруємо та зберігаємо мапу, на якій розміщено 5000 міст із рандомними координатами у межах від -5000 до 5000.

## **2. Розв'язати задачу комівояжера різними методами. Розв'язком вважати довжину найкоротшого шляху між містами.**

### **2.2. Брутфорс**

Ідея полягає у тому, щоб просто генерувати різні маршрути рандомно та серед них обрати найкращий.

```
import random
import matplotlib.pyplot as plt
import csv
import time

class Town:
    def __init__(self, x, y):
        self.location = [x, y]

    def print(self):
        print(f"location: [{self.location[0]}, {self.location[1]}]")

towns = []

with open("cities1.csv", newline='') as file:
    reader = csv.reader(file, delimiter=';')
    next(reader)
    for row in reader:
        x = int(row[0])
        y = int(row[1])
        towns.append((x, y))

towns20 = towns[:20]
towns50 = towns[:50]
towns100 = towns[:100]
towns1000 = towns[:1000]

def find_distance(city1, city2):
    dx = city1[0] - city2[0]
    dy = city1[1] - city2[1]
    dist = (dx ** 2 + dy ** 2) ** (1 / 2)
    return dist

def find_E(route):
    E = 0

    for i in range(len(route) - 1):
        E += find_distance(route[i], route[i + 1])

    E += find_distance(route[len(route) - 1], route[0])

    return E

def random_permutation(towns):
    route = towns[:]
    random.shuffle(route)
    return route
```

```

def brute_force(towns, duration):
    best_route = None
    best_E = float('inf')
    start_time = time.time()

    time_points = []
    E_points = []

    plt.ion()
    fig = plt.figure()

    while time.time() - start_time < duration:
        route = random_permutation(towns)
        E = find_E(route)

        if E < best_E:
            best_E = E
            best_route = route[:]
            current_time = time.time() - start_time
            time_points.append(current_time)
            E_points.append(best_E)

        show_route_live(best_route)

    print(f'Best length: {best_E}')
    plt.ioff()
    plt.show()

    build_time_improvements_graphic(time_points, E_points)

    return best_route, best_E

def show_route_live(route, color='green', size=20):
    plt.clf()

    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label="Маршрут")
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5)
    plt.title("Поточний найкращий маршрут")
    plt.xlabel("X координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.pause(0.5)

def build_time_improvements_graphic(time_points, E_points):
    plt.figure()
    plt.plot(time_points, E_points, marker='o')
    plt.title("Залежність довжини найкращого маршруту від часу")
    plt.xlabel("Час (секунди)")
    plt.ylabel("Довжина маршруту")
    plt.grid(True)
    plt.show()

route, E = brute_force(towns, 60)

```

Окрім алгоритму брутфорсу, у коді реалізовано візуалізацію змін маршруту у “прямому ефірі” та побудову графіку залежності загальної довжини маршруту в залежності від секунди виконання.

### 2.3. Метод найближчого сусіда

```
import random
import matplotlib.pyplot as plt
import csv
import time

class Town:
    def __init__(self, x, y):
        self.location = [x, y]

    def print(self):
        print(f"location: [{self.location[0]}, {self.location[1]}]")

cities = []

with open("cities1.csv", newline='') as file:
    reader = csv.reader(file, delimiter=';')
    next(reader)
    for row in reader:
        x = int(row[0])
        y = int(row[1])
        cities.append((x, y))

cities20 = cities[:20]
cities50 = cities[:50]
cities100 = cities[:100]
cities1000 = cities[:1000]

def find_distance(city1, city2):
    dx = city1[0] - city2[0]
    dy = city1[1] - city2[1]
    dist = (dx ** 2 + dy ** 2) ** (1 / 2)
    return dist

def find_E(route):
    E = 0

    for i in range(len(route) - 1):
        E += find_distance(route[i], route[i + 1])

    E += find_distance(route[len(route) - 1], route[0])

    return E

def nearest_neighbour(cities):
    unvisited = cities[:]
    route = []

    current_city = unvisited.pop(0)
    route.append(current_city)
```

```

while unvisited:
    nearest = None
    best_dist = float('inf')

    for city in unvisited:
        dist = find_distance(current_city, city)
        if dist < best_dist:
            best_dist = dist
            nearest = city

    route.append(nearest)
    unvisited.remove(nearest)
    current_city = nearest

return route

def show_route(route, color='blue', size=20):

    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')
    plt.scatter(x_vals[0], y_vals[0], s=size*2, c='green', zorder=6,
label='Початкове місто')
    plt.scatter(x_vals[-2], y_vals[-2], s=size*2, c='red', zorder=6,
label='Останнє місто')

    plt.xlabel("X координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.title("Візуалізація маршруту (найближчий сусід)")
    plt.show()

route = nearest_neighbour(cities50)
show_route(route)
E = find_E(route)
print(f'Best length of route: {E}')

```

## 2.4. 2-opt алгоритм

```

import random
import matplotlib.pyplot as plt
import csv
import time

class Town:
    def __init__(self, x, y):
        self.location = [x, y]

    def print(self):
        print(f"location: [{self.location[0]}, {self.location[1]}]")

cities = []

```

```

with open("cities1.csv", newline='') as file:
    reader = csv.reader(file, delimiter=';')
    next(reader)
    for row in reader:
        x = int(row[0])
        y = int(row[1])
        cities.append((x, y))

cities20 = cities[:20]
cities50 = cities[:50]
cities100 = cities[:100]
cities1000 = cities[:1000]

def find_distance(city1, city2):
    dx = city1[0] - city2[0]
    dy = city1[1] - city2[1]
    dist = (dx ** 2 + dy ** 2) ** (1 / 2)
    return dist

def find_E(route):
    E = 0

    for i in range(len(route) - 1):
        E += find_distance(route[i], route[i + 1])

    E += find_distance(route[len(route) - 1], route[0])

    return E

def random_permutation(towns):
    route = towns[:]
    random.shuffle(route)
    return route

def nearest_neighbour(cities):
    unvisited = cities[:]
    route = []

    current_city = unvisited.pop(0)
    route.append(current_city)

    while unvisited:
        nearest = None
        best_dist = float('inf')

        for city in unvisited:
            dist = find_distance(current_city, city)
            if dist < best_dist:
                best_dist = dist
                nearest = city

        route.append(nearest)
        unvisited.remove(nearest)
        current_city = nearest

    return route

# Рандомний вибір індексів

def two_opt_optimization_random(route):
    best_route = route[:]
    best_E = find_E(best_route)

    # Використовуємо цикл з умовою break
    # для того, щоб обмежити кількість перевірок
    # до 10000
    for _ in range(10000):
        # Вибираємо випадкову пару місць
        # в роуті
        i = random.randint(0, len(route) - 2)
        j = random.randint(i + 1, len(route) - 1)

        # Виконуємо обмежені зміни в роуті
        # та обчислюємо нову його довжину
        new_route = route[:i] + route[j:i:-1] + route[j:]
        new_E = find_E(new_route)

        # Якщо новий роут коротший, ніж
        # попередній, то заміняємо його
        if new_E < best_E:
            best_route = new_route
            best_E = new_E

    return best_route

```

```

n = len(route)

i = random.randint(1, n - 3)
j = random.randint(i + 1, n - 2)

A = best_route[0:i + 1]
B = best_route[i + 1:j + 1]
C = best_route[j + 1:]

candidate = A + B[::-1] + C
candidate_E = find_E(candidate)

if candidate_E < best_E:
    best_route = candidate
    best_E = candidate_E

return best_route, best_E

def two_opt_algorithm_random(route, time_limit):
    best_route = route[:]
    best_E = find_E(best_route)
    not_improved_counter = 0

    time_points = []
    E_points = []

    start_time = time.time()

    plt.ion()
    fig = plt.figure()

    while time.time() - start_time < time_limit and not_improved_counter < 1000:
        candidate, candidate_E = two_opt_optimization_random(best_route)

        if candidate_E < best_E:
            not_improved_counter = 0
            best_route = candidate
            best_E = candidate_E

            current_time = time.time() - start_time
            time_points.append(current_time)
            E_points.append(best_E)

            show_route_live(best_route)
        else:
            not_improved_counter += 1

    plt.ioff()
    plt.show()
    build_time_improvements_graphic(time_points, E_points)

    return best_route

# Повний перебір

def two_opt_algorithm_enum(route, time_limit):
    best_route = route[:]
    best_E = find_E(best_route)
    without_impr = 0

```

```

improved = True
n = len(route)

time_points = []
E_points = []

plt.ion()
fig = plt.figure()

start_time = time.time()

while time.time() - start_time < time_limit and without_impr < 10:
    improved = False
    for i in range(1, n - 2):
        for j in range(i + 1, n - 1):
            A = best_route[:i]
            B = best_route[i:j + 1]
            C = best_route[j + 1:]

            candidate = A + B[::-1] + C
            candidate_E = find_E(candidate)

            if candidate_E < best_E:
                best_route = candidate
                best_E = candidate_E
                show_route_live(best_route)
                improved = True
                without_impr = 0

            current_time = time.time() - start_time
            time_points.append(current_time)
            E_points.append(best_E)

            break

        if improved:
            break
    if not improved:
        without_impr += 1

plt.ioff()
plt.show()
build_time_improvements_graphic(time_points, E_points)

return best_route


def build_time_improvements_graphic(time_points, E_points):
    plt.figure()
    plt.plot(time_points, E_points, marker='o')
    plt.title("Залежність довжини найкращого маршруту від часу")
    plt.xlabel("Час (секунди)")
    plt.ylabel("Довжина маршруту")
    plt.grid(True)
    plt.show()

def show_route_live(route, color='blue', size=20):
    plt.clf()

    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')

```

```

    plt.scatter(x_vals[0], y_vals[0], s=size * 2, c='green', zorder=6,
label='Початкове місто')
    plt.scatter(x_vals[-2], y_vals[-2], s=size * 2, c='red', zorder=6,
label='Останнє місто')

    plt.title("Поточний найкращий маршрут")
    plt.xlabel("X координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.pause(0.5)

def show_route(route, color='blue', size=20):
    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')
    plt.scatter(x_vals[0], y_vals[0], s=size * 2, c='green', zorder=6,
label='Початкове місто')
    plt.scatter(x_vals[-2], y_vals[-2], s=size * 2, c='red', zorder=6,
label='Останнє місто')

    plt.title("Візуалізація маршруту (найближчий сусід)")
    plt.xlabel("X координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.show()

start_route = random_permutation(cities50)
route = two_opt_algorithm_enum(start_route, 60)
# show_route(route)

E_start = find_E(start_route)
E = find_E(route)
print(f'Start length of route: {E_start}')
print(f'Best length of route: {E}')

```

У коді реалізовано два підходи для вибору індексів  $i$  та  $j$ . У першому варіанті алгоритму індекси обираються випадково, а у другому виконуємо повний перебір. Під час тестування перевіримо, який працюватиме краще. Також початковий маршрут можна обирати випадковим чином або попередньо викликати функцію, яка буде повертати маршрут утворений за допомогою методу найближчого сусіда.

## 2.5. Алгоритм імітації відпалу

```
import random
import matplotlib.pyplot as plt
import csv
import time
import math


class Town:
    def __init__(self, x, y):
        self.location = [x, y]

    def print(self):
        print(f"location: [{self.location[0]}, {self.location[1]}]")


cities = []

with open("cities1.csv", newline='') as file:
    reader = csv.reader(file, delimiter=';')
    next(reader)
    for row in reader:
        x = int(row[0])
        y = int(row[1])
        cities.append((x, y))

cities20 = cities[:20]
cities50 = cities[:50]
cities100 = cities[:100]
cities1000 = cities[:1000]


def find_distance(city1, city2):
    dx = city1[0] - city2[0]
    dy = city1[1] - city2[1]
    dist = (dx ** 2 + dy ** 2) ** (1 / 2)
    return dist


def find_E(route):
    E = 0

    for i in range(len(route) - 1):
        E += find_distance(route[i], route[i + 1])

    E += find_distance(route[len(route) - 1], route[0])

    return E


def random_permutation(towns):
    route = towns[:]
    random.shuffle(route)
    return route


def nearest_neighbour(cities):
    unvisited = cities[:]
    route = []

    current_city = unvisited.pop(0)
    route.append(current_city)

    while len(unvisited) > 0:
        min_dist = float('inf')
        next_city = None
        for city in unvisited:
            dist = find_distance(route[-1], city)
            if dist < min_dist:
                min_dist = dist
                next_city = city
        route.append(next_city)
        unvisited.remove(next_city)
```

```

while unvisited:
    nearest = None
    best_dist = float('inf')

    for city in unvisited:
        dist = find_distance(current_city, city)
        if dist < best_dist:
            best_dist = dist
            nearest = city

    route.append(nearest)
    unvisited.remove(nearest)
    current_city = nearest

return route

# randomний вибір індексів

def two_opt_optimization_random(route):
    n = len(route)

    i = random.randint(1, n - 3)
    j = random.randint(i + 1, n - 2)

    A = route[0:i + 1]
    B = route[i + 1:j + 1]
    C = route[j + 1:]

    candidate = A + B[::-1] + C

    return candidate

# алгоритм імітації відпалу

def annealing(route, time_limit, T, T_end, a):
    best_route = route[:]
    best_E = find_E(best_route)

    start_time = time.time()

    time_points = []
    E_points = []

    plt.ion()
    fig = plt.figure()

    while time.time() - start_time < time_limit and T > T_end:

        candidate = two_opt_optimization_random(best_route)
        candidate_E = find_E(candidate)

        dE = candidate_E - best_E

        if dE < 0:
            best_route = candidate
            best_E = candidate_E
        else:
            P = math.exp(-dE / T) # спробувати прибрати можливість вибору
гіршого шляху
            if random.uniform(0, 1) < P:
                best_route = candidate
                best_E = candidate_E

```

```

        current_time = time.time() - start_time
        time_points.append(current_time)
        E_points.append(best_E)

        show_route_live(best_route)

        T = T * a

plt.ioff()
plt.show()
# build_time_improvements_graphic(time_points, E_points)

return best_route, time_points, E_points

# Повний перебір

def two_opt_algorithm_enum(route, time_limit):
    best_route = route[:]
    best_E = find_E(best_route)
    without_impr = 0
    improved = True
    n = len(route)

    time_points = []
    E_points = []

    plt.ion()
    fig = plt.figure()

    start_time = time.time()

    while time.time() - start_time < time_limit and without_impr < 10:
        improved = False
        for i in range(1, n - 2):
            for j in range(i + 1, n - 1):
                A = best_route[:i]
                B = best_route[i:j + 1]
                C = best_route[j + 1:]

                candidate = A + B[::-1] + C
                candidate_E = find_E(candidate)

                if candidate_E < best_E:
                    best_route = candidate
                    best_E = candidate_E
                    show_route_live(best_route)
                    improved = True
                    without_impr = 0

                    current_time = time.time() - start_time
                    time_points.append(current_time)
                    E_points.append(best_E)

                    break

                if improved:
                    break
            if not improved:
                without_impr += 1

    plt.ioff()
    plt.show()

```

```

# build_time_improvements_graphic(time_points, E_points)

    return best_route, time_points, E_points

# комбінація 2-opt та імітації відпалу

def two_opt_plus_annealing(route, time_limit, T, T_end, a):

    sa_time = time_limit * 0.8
    opt_time = time_limit * 0.2

    route_after_sa, sa_time_points, sa_E_points = annealing(route, sa_time,
T, T_end, a)
    time_shift = sa_time_points[-1] if sa_time_points else 0

    final_route, opt_time_points, opt_E_points =
two_opt_algorithm_enum(route_after_sa, opt_time)
    opt_time_points = [t + time_shift for t in opt_time_points]

    time_points = sa_time_points + opt_time_points
    E_points = sa_E_points + opt_E_points

    build_time_improvements_graphic(time_points, E_points)

    return final_route


def build_time_improvements_graphic(time_points, E_points):
    plt.figure()
    plt.plot(time_points, E_points, marker='o')
    plt.title("Залежність довжини найкращого маршруту від часу")
    plt.xlabel("Час (секунди)")
    plt.ylabel("Довжина маршруту")
    plt.grid(True)
    plt.show()


def show_route_live(route, color='blue', size=20):
    plt.clf()

    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')
    plt.scatter(x_vals[0], y_vals[0], s=size * 2, c='green', zorder=6,
label='Початкове місто')
    plt.scatter(x_vals[-2], y_vals[-2], s=size * 2, c='red', zorder=6,
label='Останнє місто')

    plt.title("Поточний найкращий маршрут")
    plt.xlabel("Х координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.pause(0.05)


def show_route(route, color='blue', size=20):
    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

```

```

plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')
plt.scatter(x_vals[0], y_vals[0], s=size * 2, c='green', zorder=6,
label='Початкове місто')
plt.scatter(x_vals[-2], y_vals[-2], s=size * 2, c='red', zorder=6,
label='Останнє місто')

plt.title("Візуалізація маршруту (найближчий сусід)")
plt.xlabel("X координата")
plt.ylabel("Y координата")
plt.grid(True)
plt.axis("equal")
plt.legend()
plt.show()

start_route = random_permutation(cities20)
route = two_opt_plus_annealing(start_route, 30, 10, 0, 0.99)
# show_route(route)

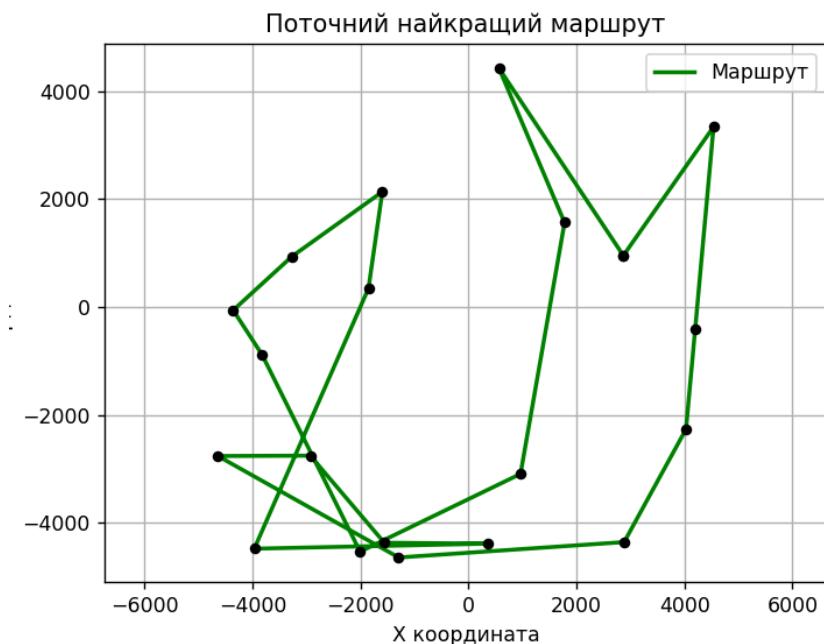
E_start = find_E(start_route)
E = find_E(route)
print(f'Start length of route: {E_start}')
print(f'Best length of route: {E}')

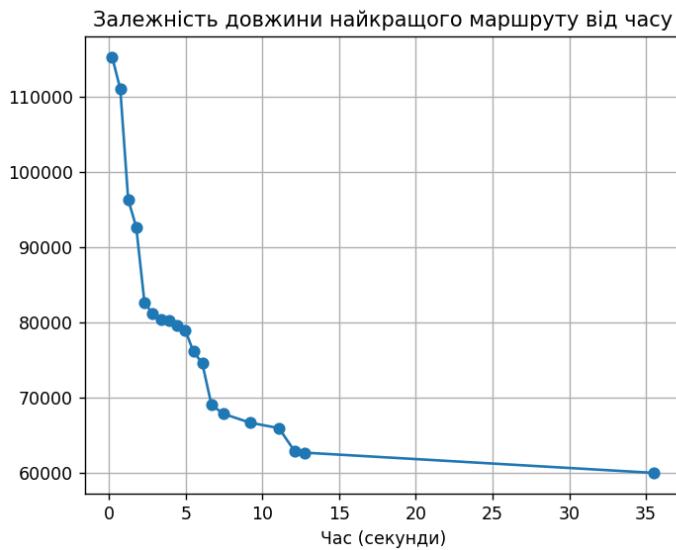
```

## Тестування та порівняння методів

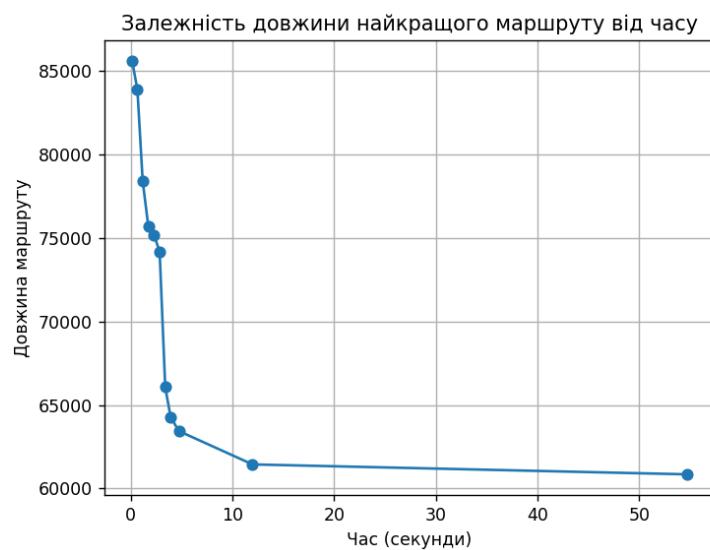
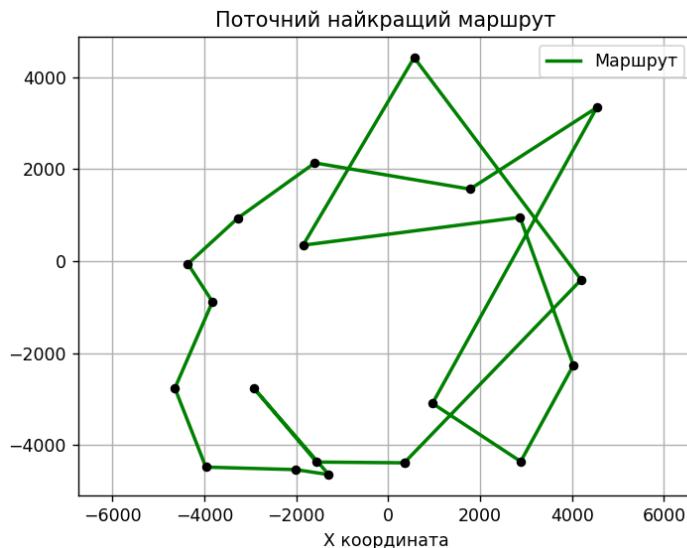
Спочатку протестуємо метод грубої сили на мапах різного розміру:

20 міст, час роботи 1 хвилина:



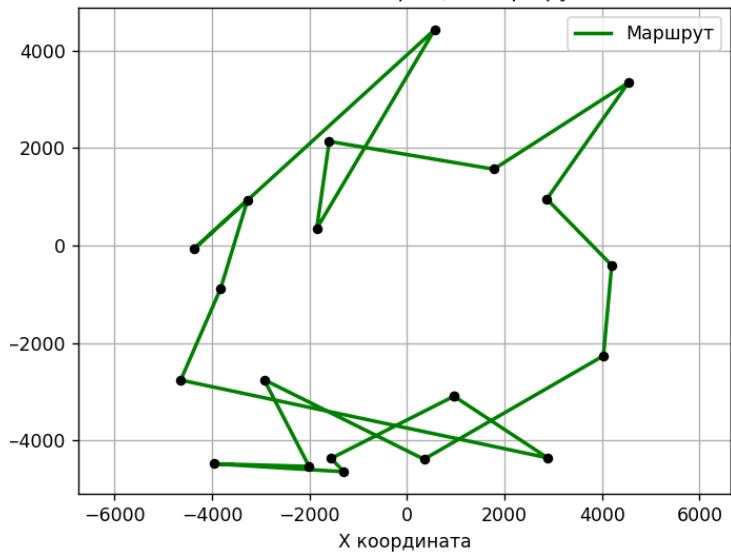


Довжина: Best length: **59954.17459457059.**

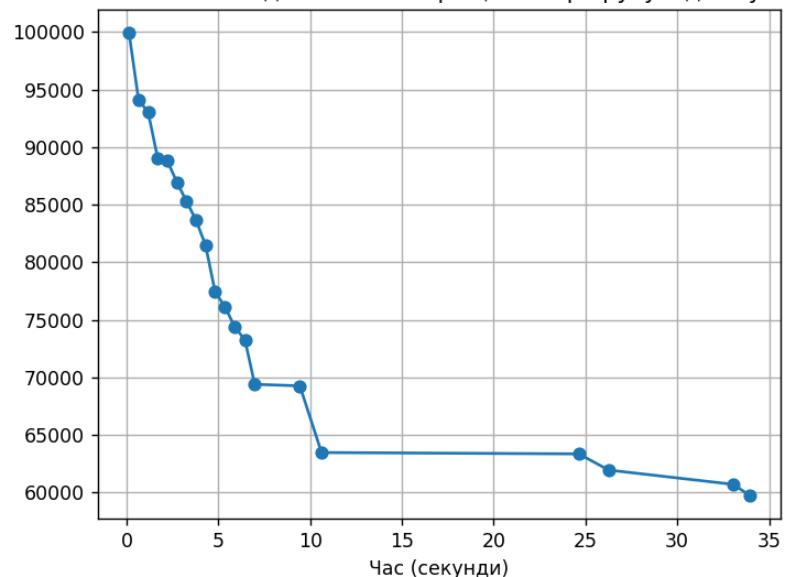


Довжина: Best length: **60844.24694359182**

Поточний найкращий маршрут

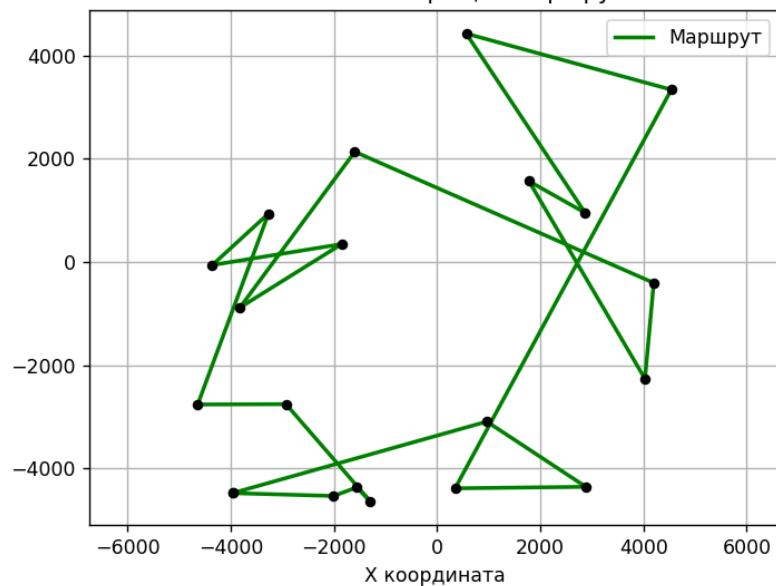


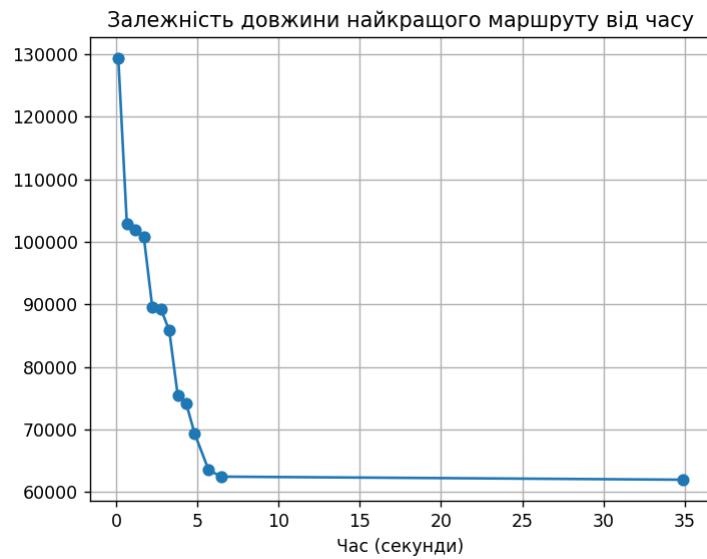
Залежність довжини найкращого маршруту від часу



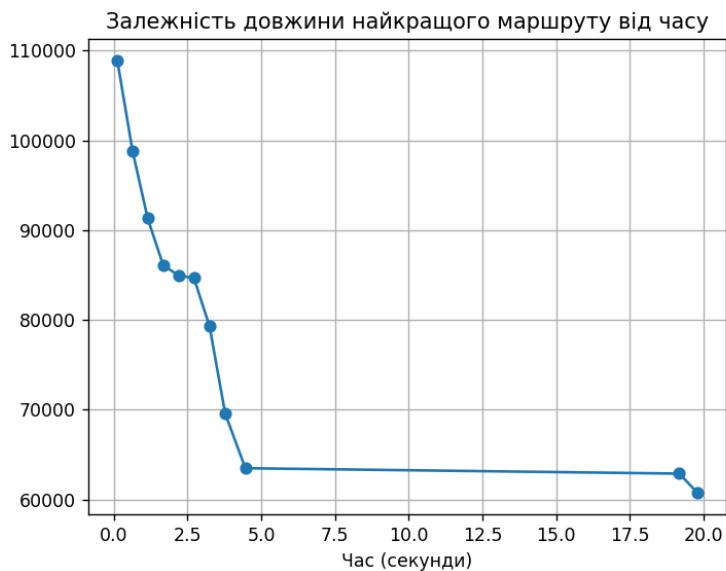
**Best length: 59751.633921505854**

Поточний найкращий маршрут



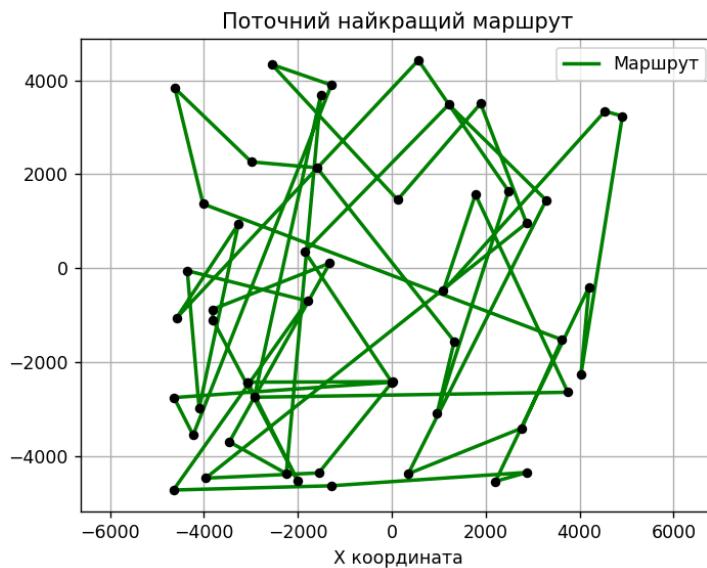


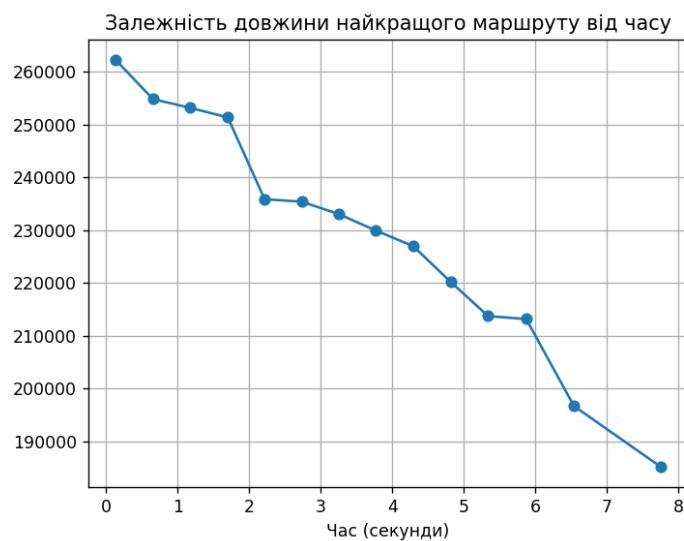
**Best length: 61956.42784432854**



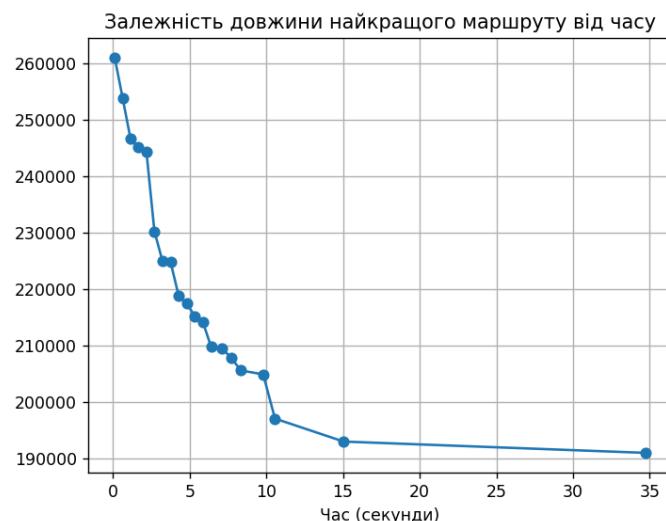
**Best length: 60778.69833983264**

Збільшимо розмір мапи до 50:

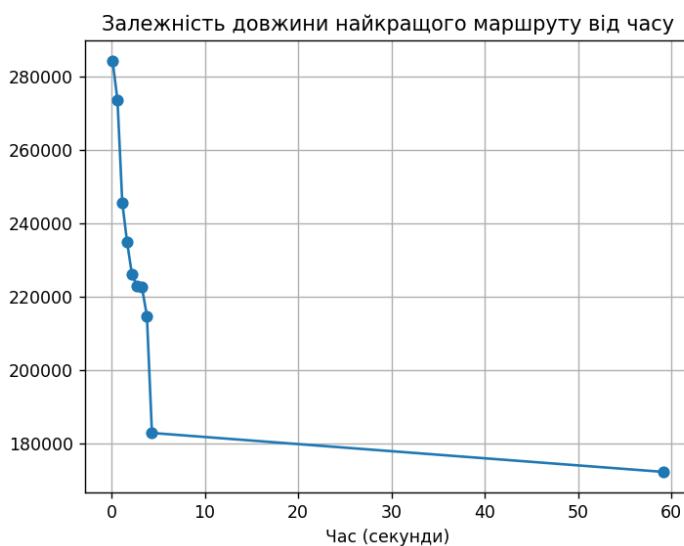




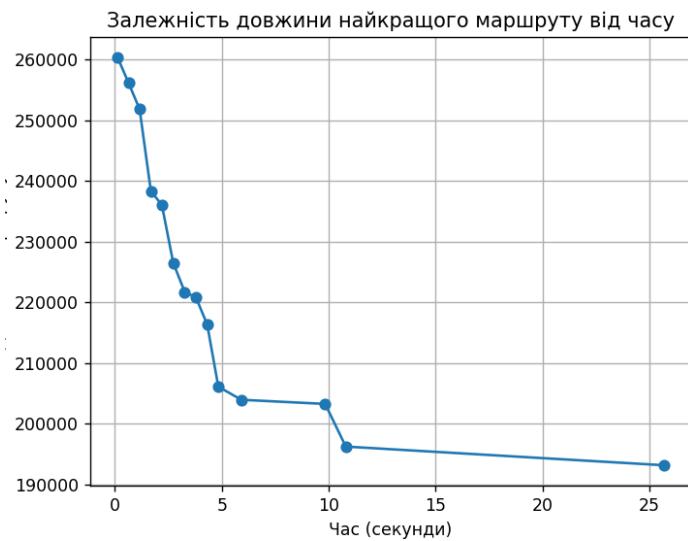
**Best length: 185149.45712659645**



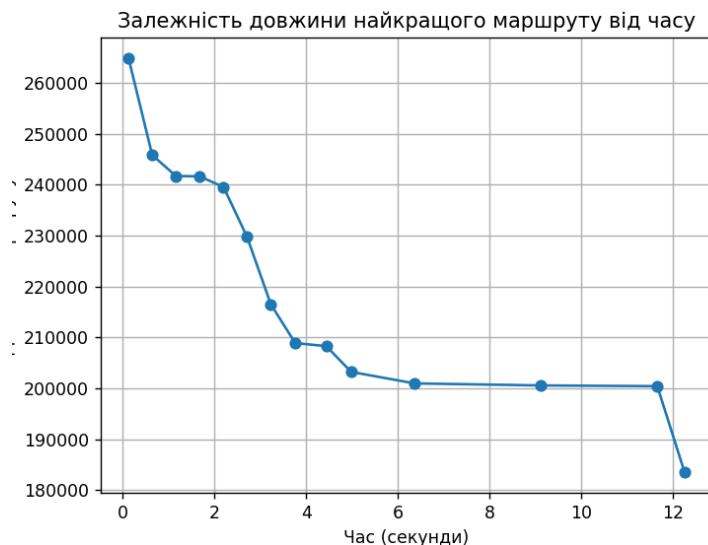
**Best length: 191062.71041988858**



**Best length: 172254.54344348202**

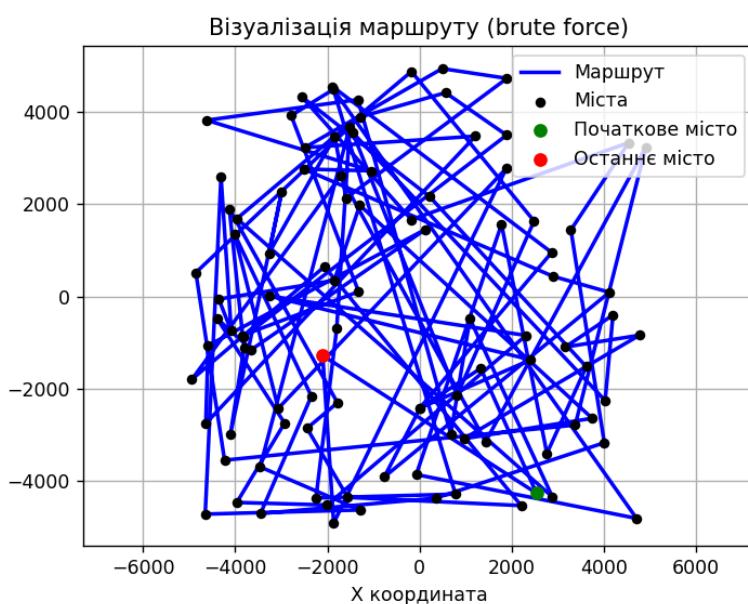


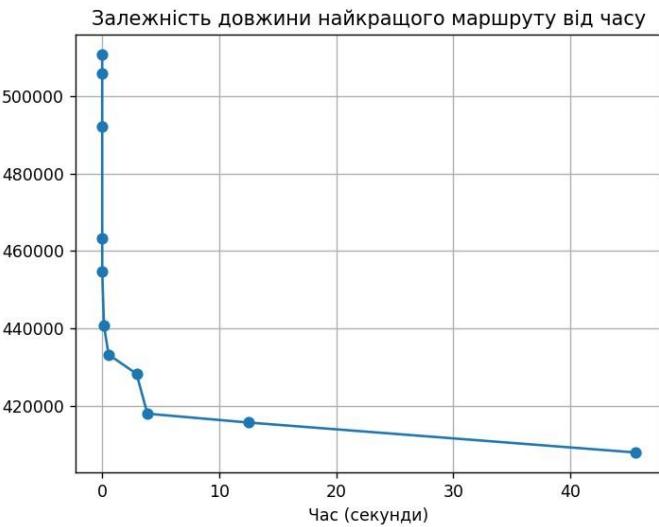
**Best length: 193131.47401473133**



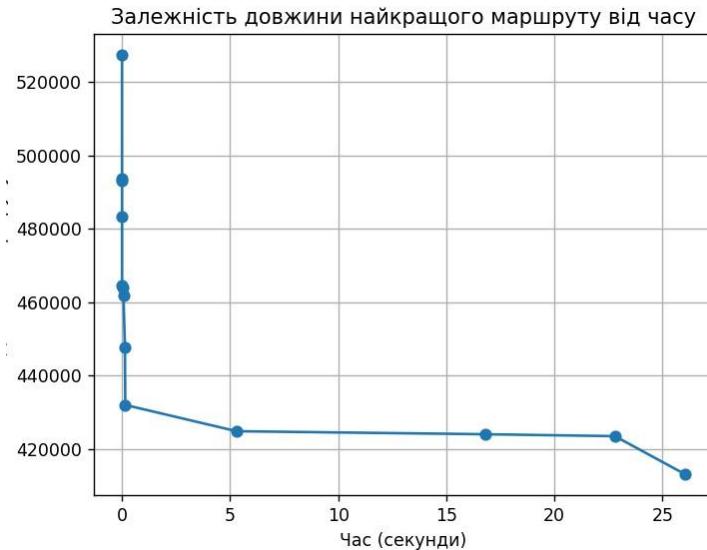
**Best length: 183563.08032681447**

**Збільшимо мапу до 100 міст:**

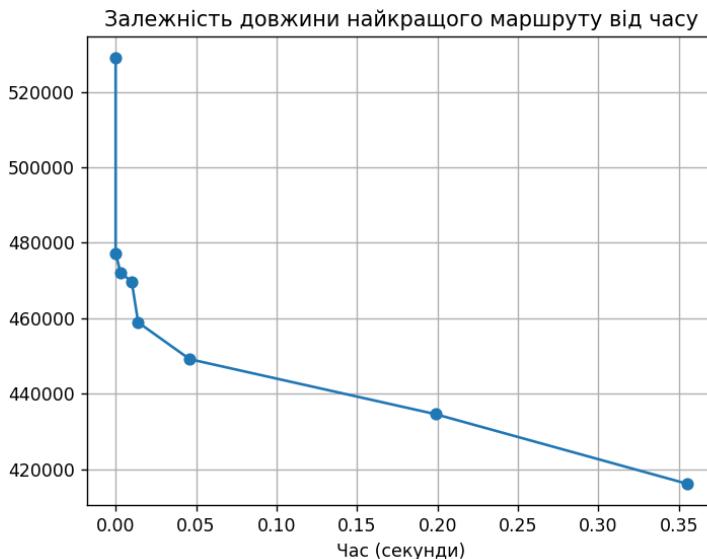




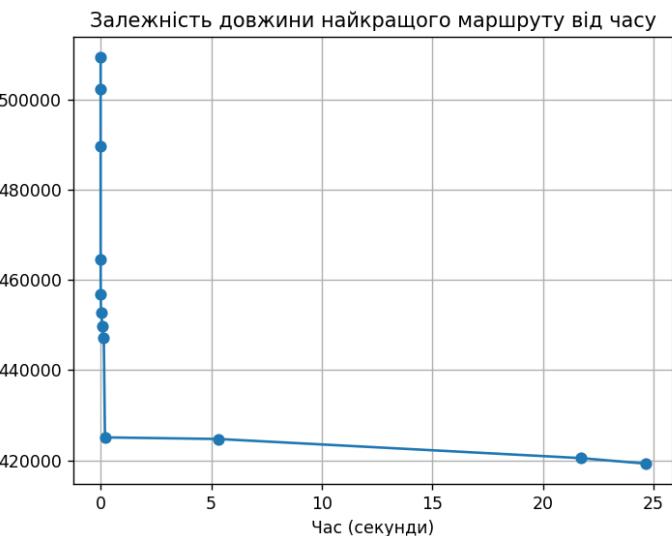
**Best length: 407913.9591615696**



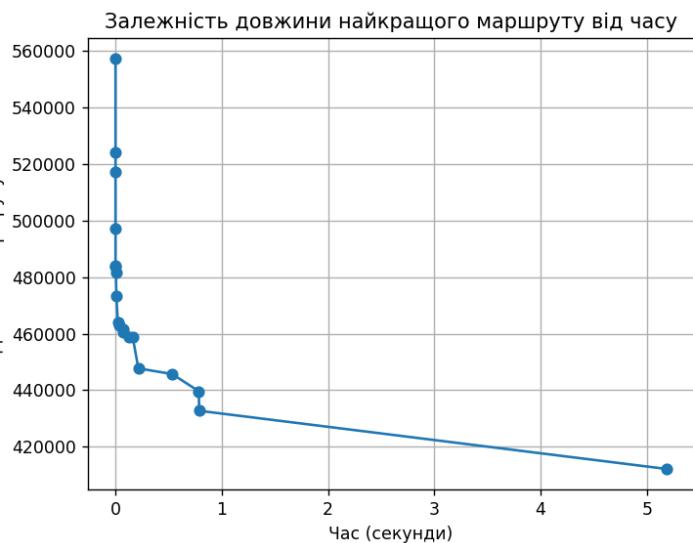
**Best length: 413159.5109617793**



**Best length: 416151.3780968007**

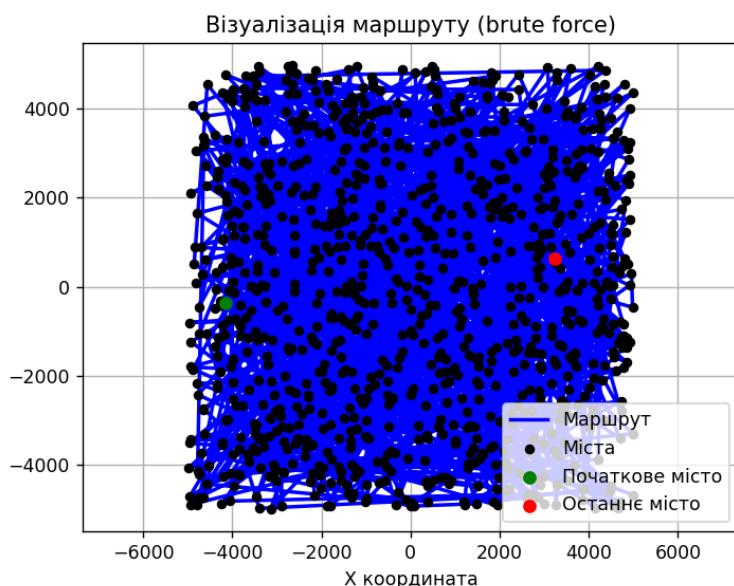


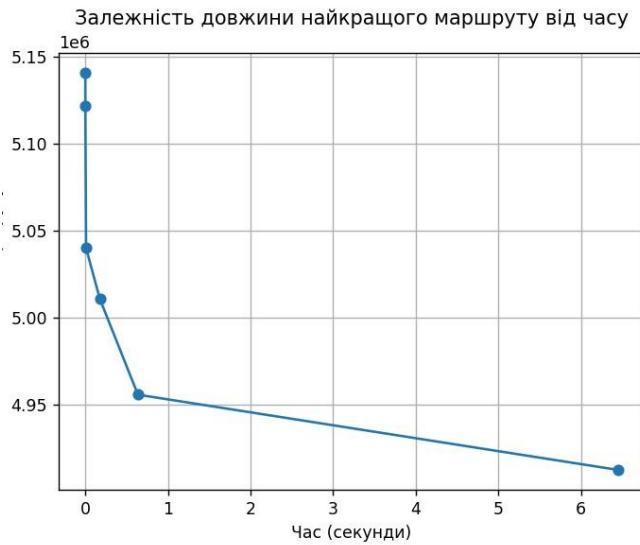
**Best length: 419249.3051931809**



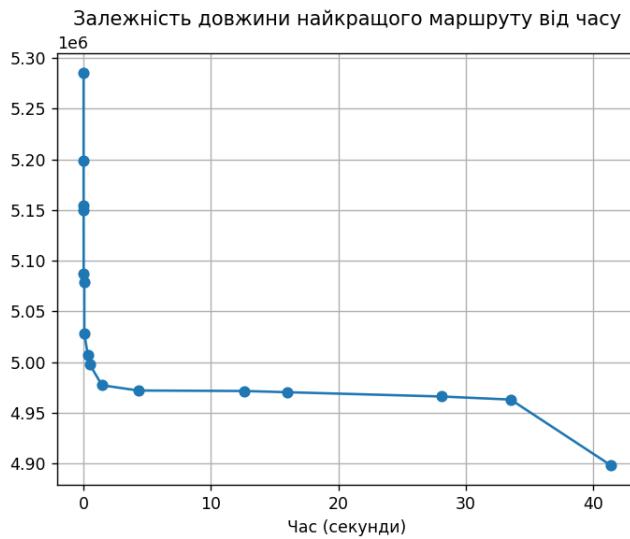
**Best length: 412157.773327797**

Тепер збільшимо до 1000 міст:

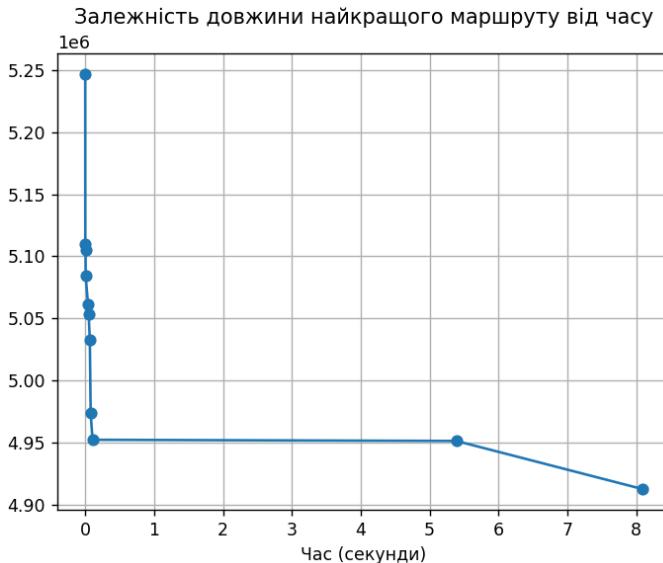




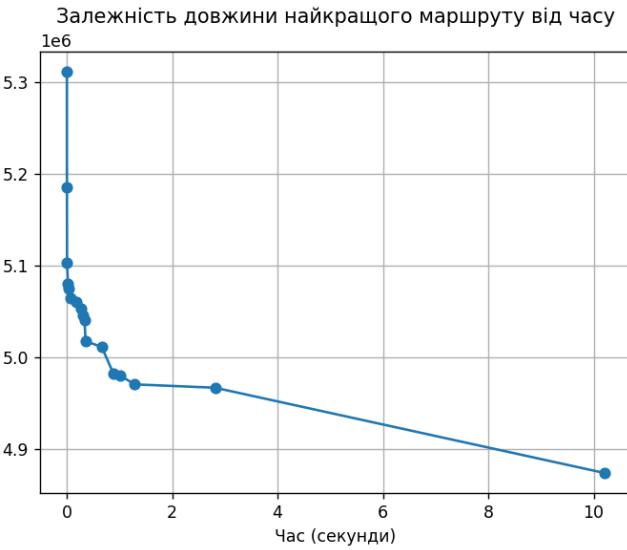
**Best length: 4912663.512915702**



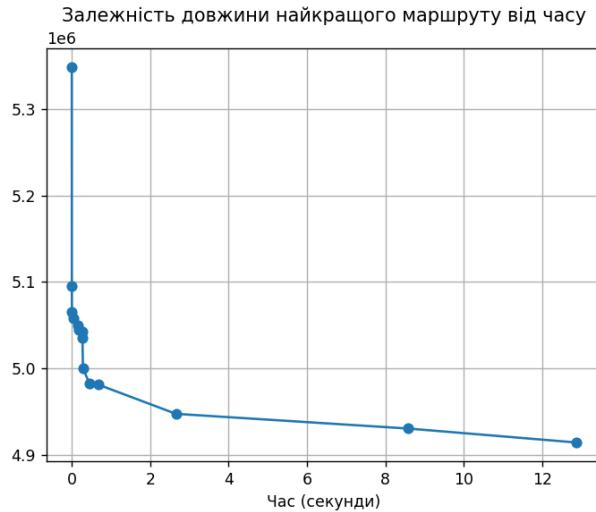
**Best length: 4898294.5557528315**



**Best length: 4912425.604517576**

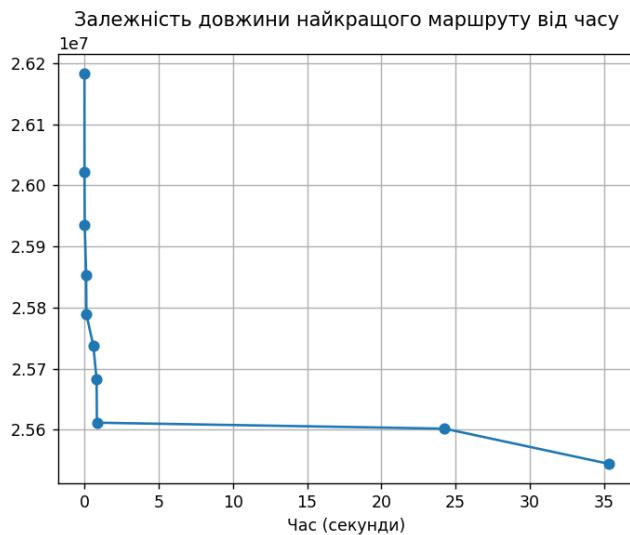


**Best length: 4874375.321042917**

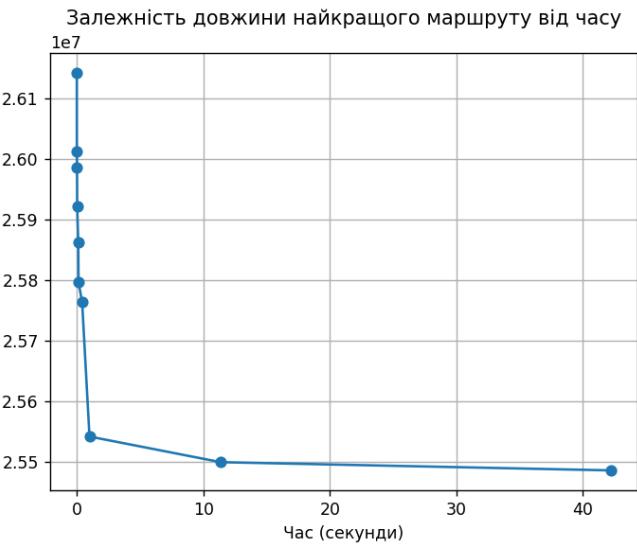


**Best length: 4914741.4560230505**

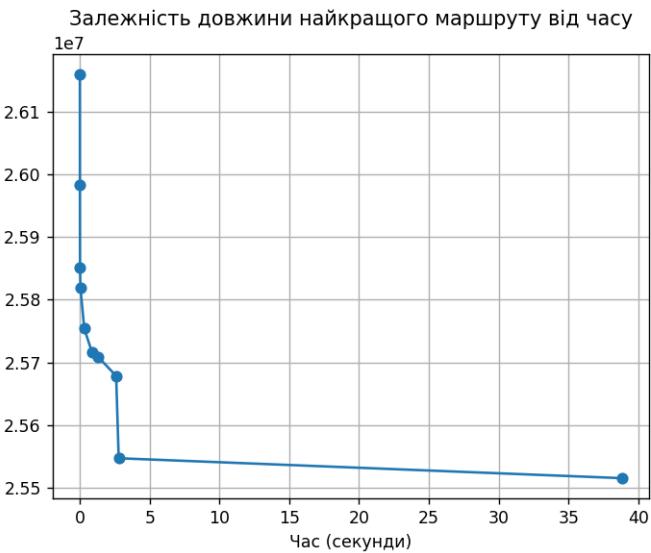
Тепер протестуємо з повним набором (5000 міст):



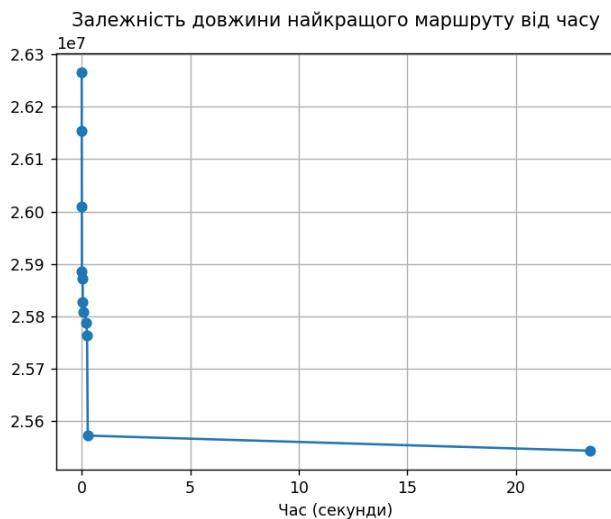
**Best length: 25544130.835744422**



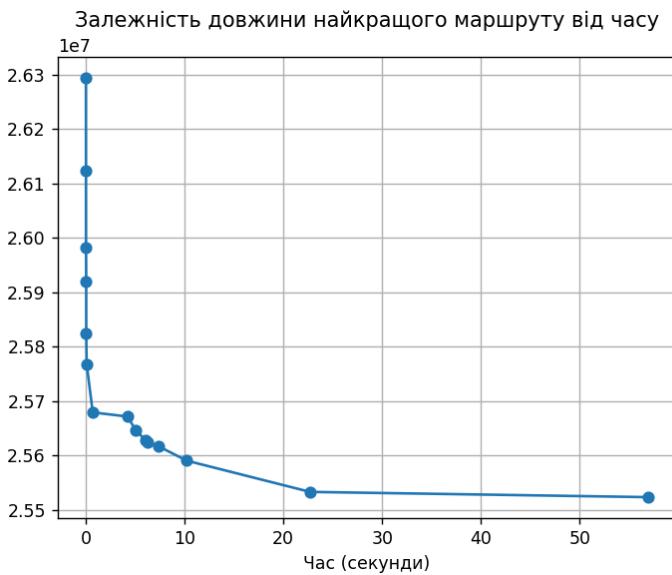
**Best length: 25485638.493139286**



**Best length: 25515557.96863001**



**Best length: 25543596.543327283**



**Best length: 25523287.329611227**

**Результати роботи брутфорсу:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	59954.17	185149.45	407913.95	4912663.51	25544130.83
2	60844.24	191062.71	413159.51	4898294.55	25485638.49
3	59751.63	172254.54	416151.37	4912425.60	25515557.96
4	61956.42	193131.47	419249.30	4874375.32	25543596.54
5	60778.69	183563.08	412157.77	4914741.45	25523287.32

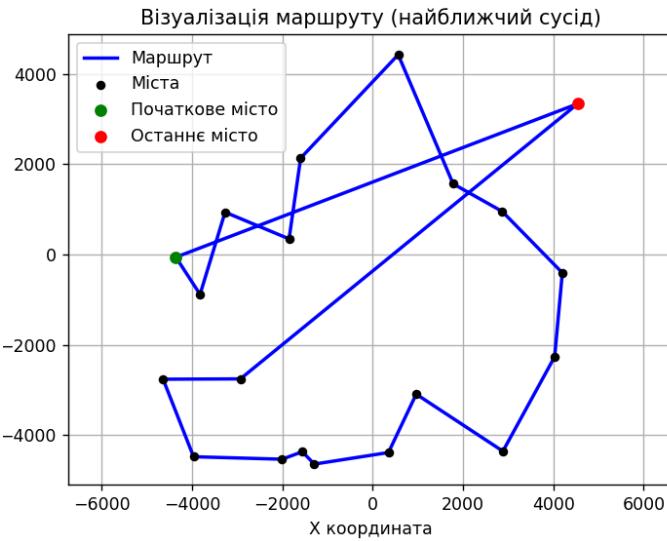
Якщо аналізувати графіки покращень за часом, то вони всі мають приблизно однакову форму. На початку покращення відбуваються досить часто, але з плином часу все важче і важче знаходити кращий варіант, тому графіки мають форму гілки гіперболи.

Аналізуючи результати у таблиці, можна помітити досить цікавий ефект -усі отримані значення досить близькі. Я думав, що оскільки алгоритм працює повністю рандомно, то і результати будуть відрізнятися надзвичайно сильно, але в результаті всі значення досить близькі. Такий результат був неочікуваним.

Загалом можна сказати, що алгоритм знаходить ненайоптимальніші маршрути. Особливо, якщо подивитися на мапу з великою кількістю міст.

**Тестування методу найближчого сусіда:**

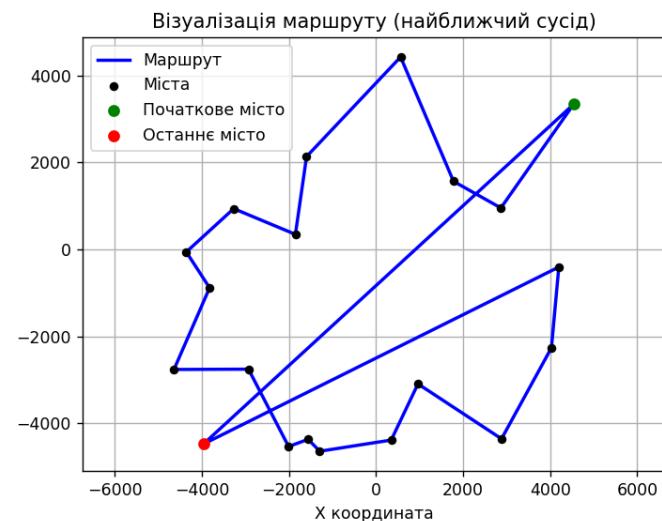
**Спробуємо запустити алгоритм з різних стартових точок:**



**Best length of route: 50800.65022450286**



**Best length of route: 50290.809307938835**



**Best length of route: 53144.77102844519**

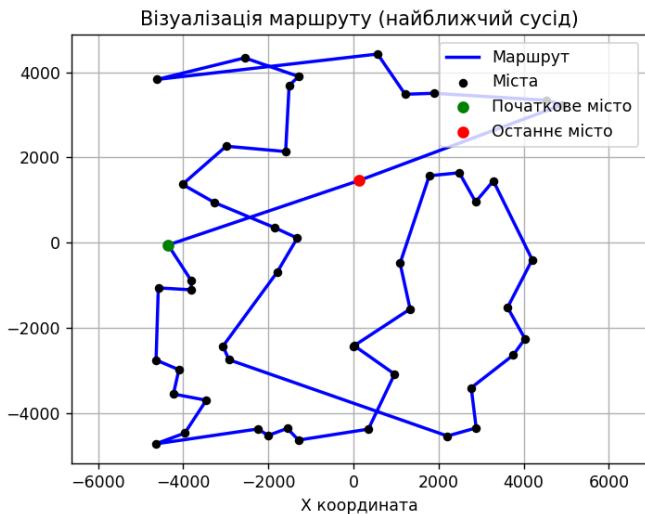


**Best length of route: 51006.21317678368**



**Best length of route: 38054.96923200974 (Оптимальний)**

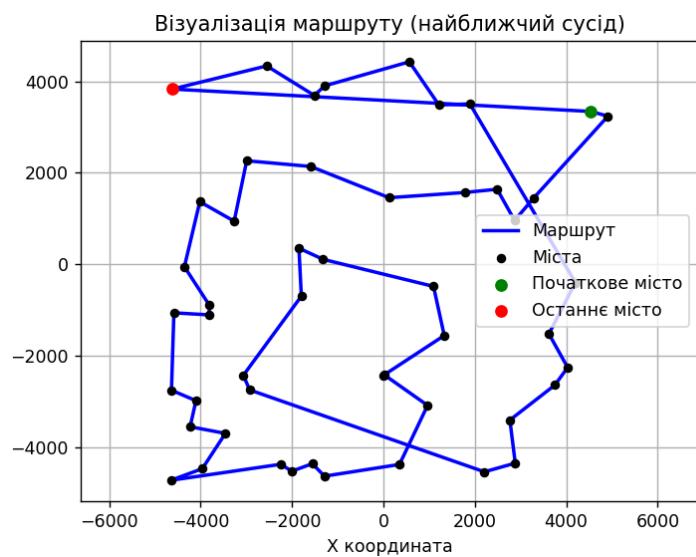
Тепер збільшимо кількість міст до 50 та запустимо алгоритм з різних точок:



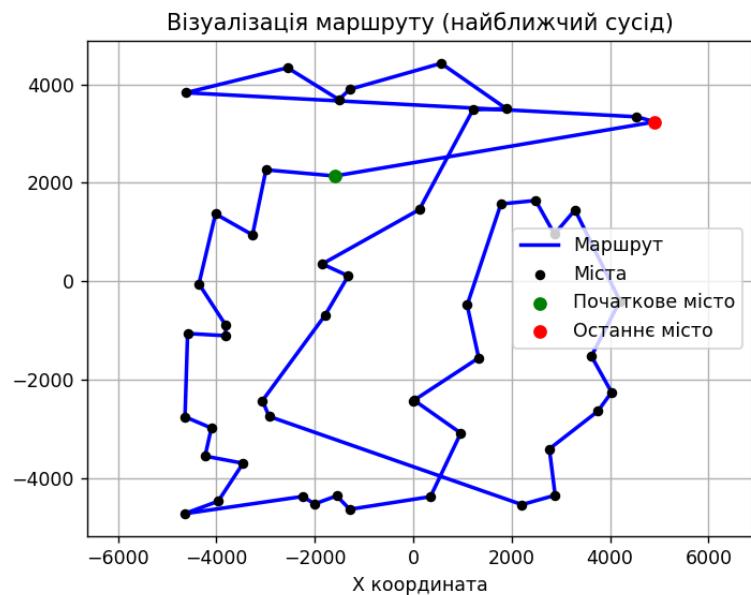
**Best length of route: 69532.21246425313**



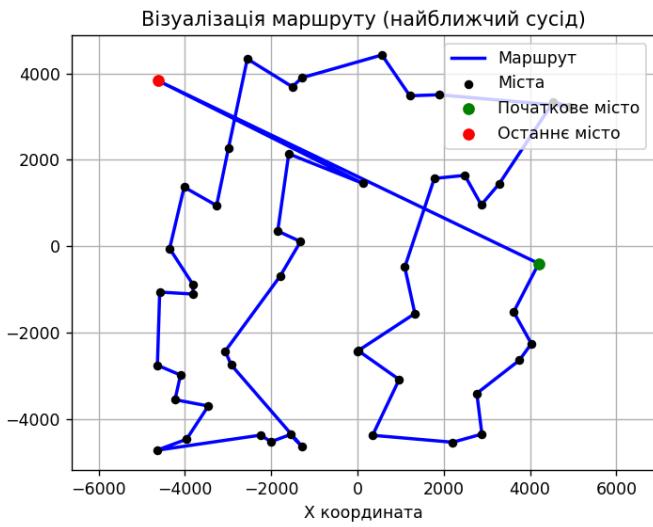
**Best length of route: 75872.28624131491**



**Best length of route: 70008.85511369162**

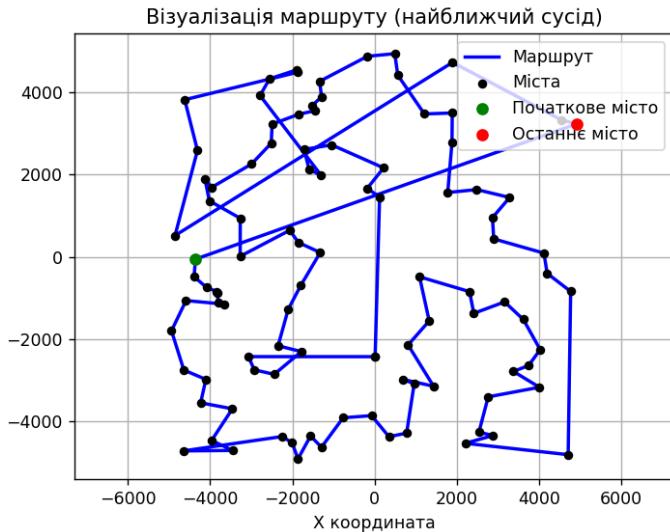


**Best length of route: 72800.0641431605**

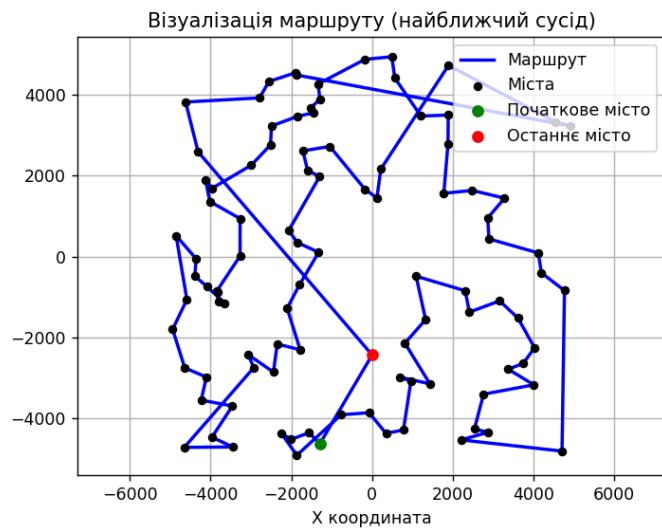


**Best length of route: 69828.57642113873**

**Збільшимо розмір мапи до 100 міст:**

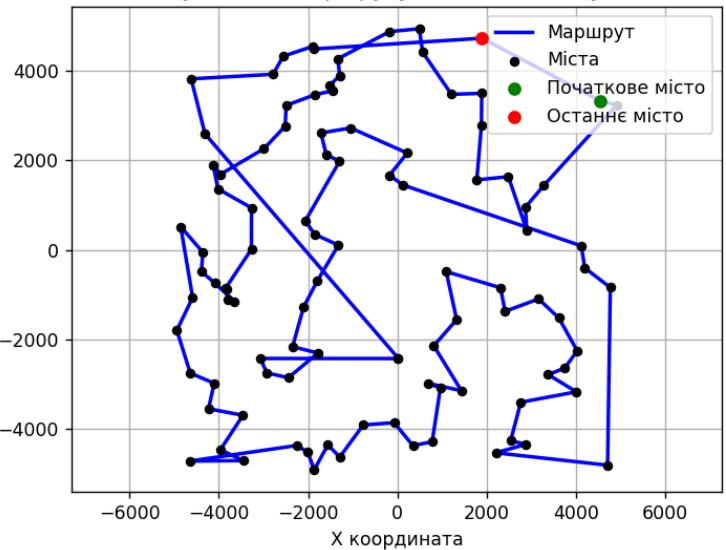


**Best length of route: 103891.36876358309**



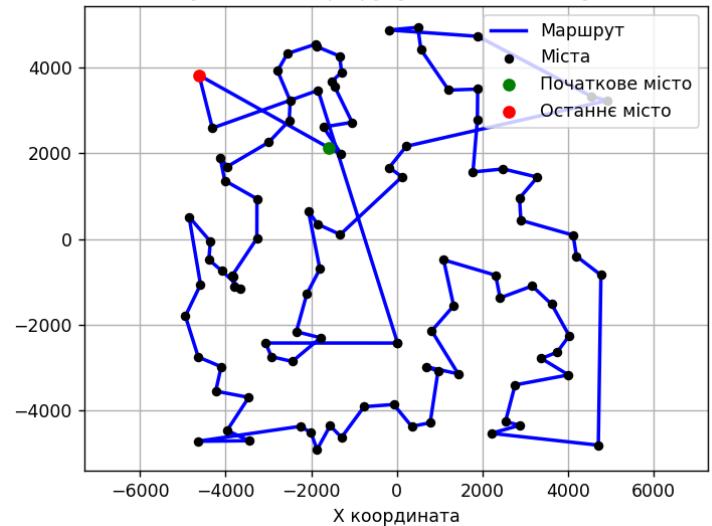
**Best length of route: 96889.94556415753**

Візуалізація маршруту (найближчий сусід)



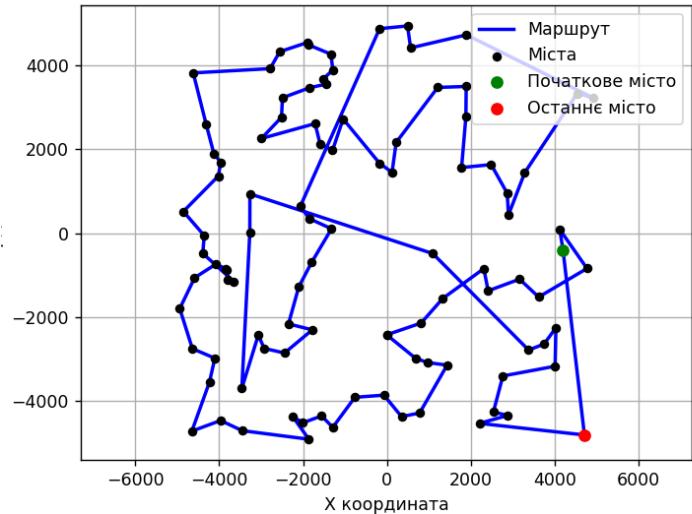
**Best length of route: 95804.07941064434**

Візуалізація маршруту (найближчий сусід)



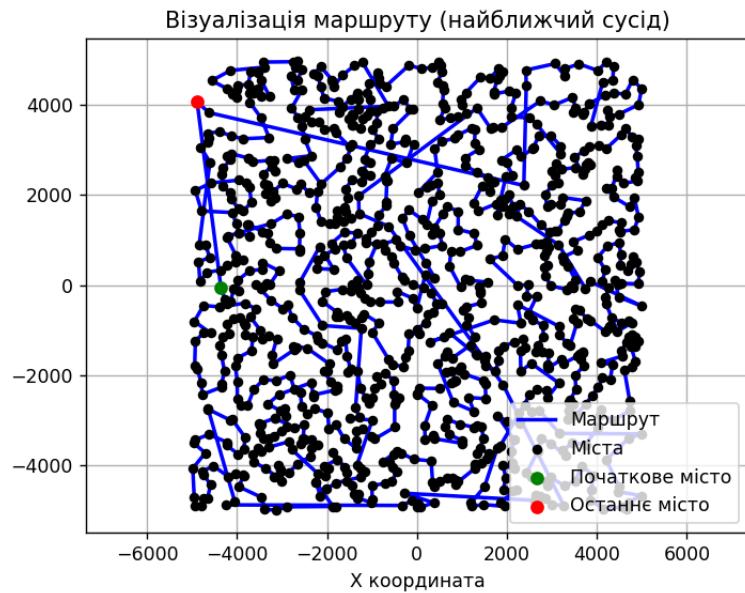
**Best length of route: 96521.28814285385**

Візуалізація маршруту (найближчий сусід)



**Best length of route: 93920.09622158257**

**Збільшимо розмір мапи до 1000 та 5000 міст і результати запишемо у таблицю:**



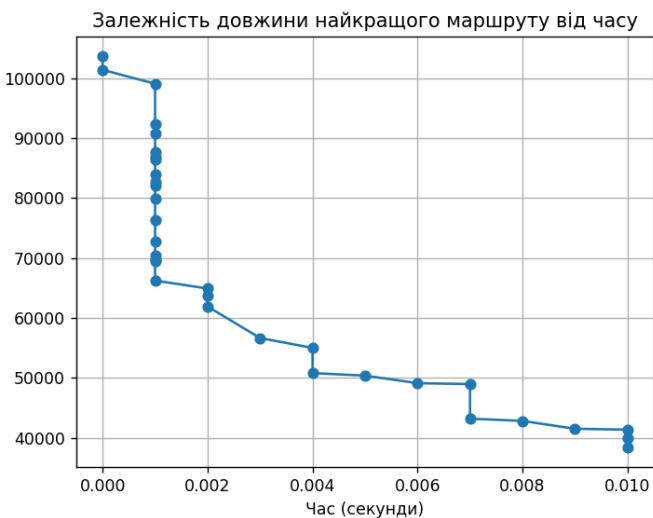
**Best length of route: 295677.0335457827**

**Результати роботи методу найближчого сусіда:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	50800.65	69532.21	103891.36	295677.03	636811.65
2	50290.80	75872.28	96889.94	291282.21	629902.49
3	53144.77	70008.85	95804.07	292995.68	629127.83
4	51006.21	72800.06	96521.28	289590.19	629126.00
5	38054.96	69828.57	93920.09	298727.56	623537.76

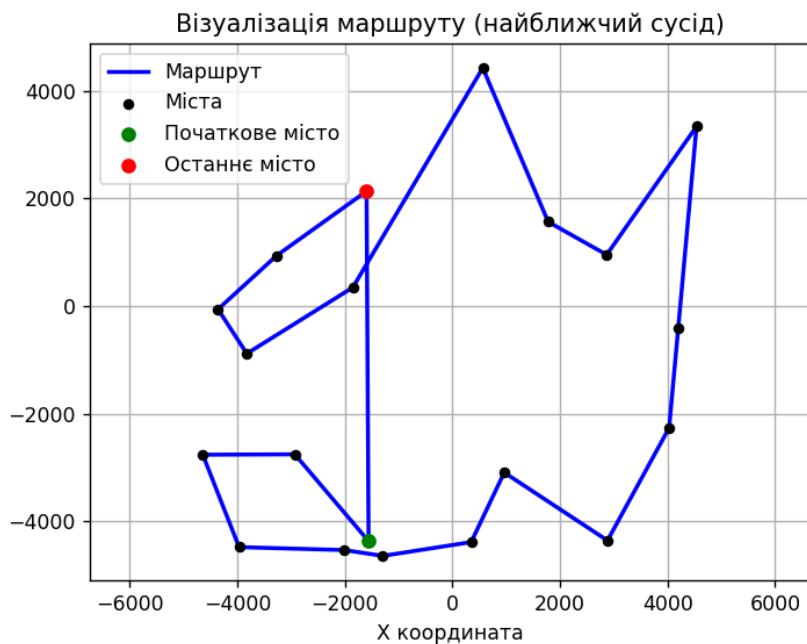
Алгоритм знаходить значно кращі значення за брутфорс. Запуски алгоритму з різних точок дає різні результати. Якщо правильно обрати точку, то можна значно скоротити маршрут. У кінці роботи, коли щалишилося мало невідвіданих міст відбуваються великі стрибки, які погіршують маршрут. Добре працює з мапами, на яких міста розташовані по контуру. У випадку з 20-ма містами знайшов ідеальний маршрут.

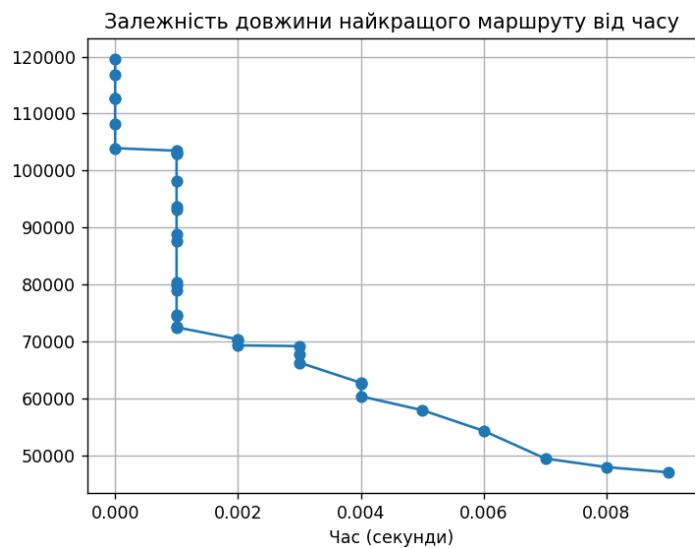
## Тестування 2-opt алгоритму (повний перебір):



Start length of route: 104306.42520051233

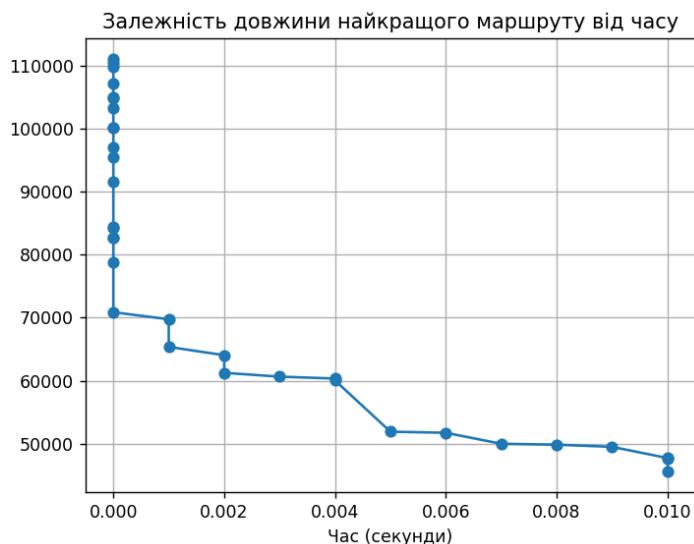
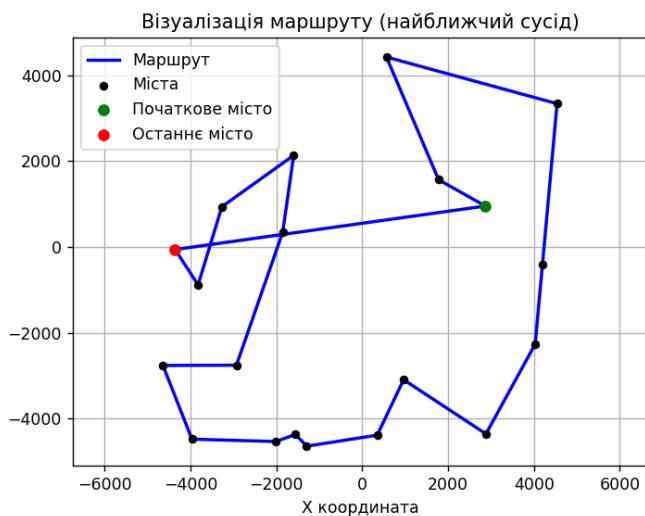
Best length of route: 38373.23534859422





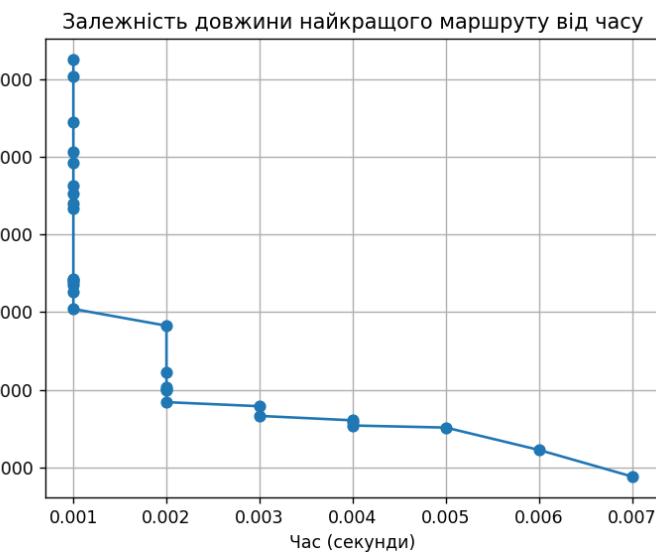
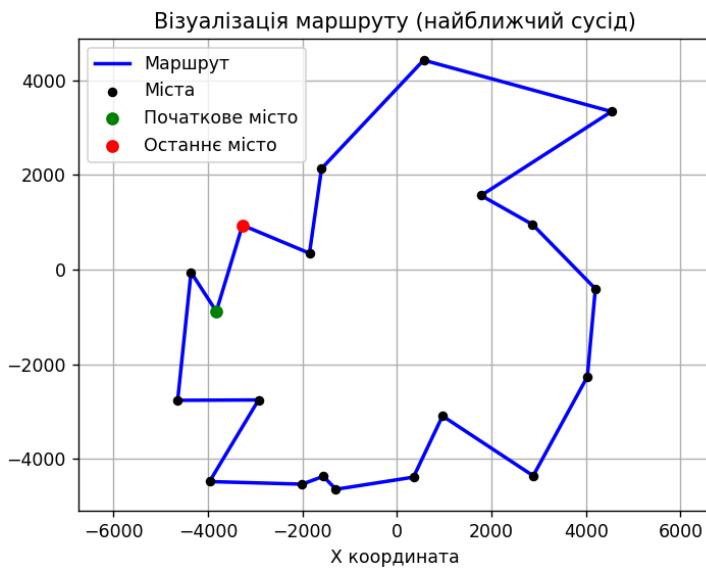
Start length of route: 121481.19352099339

Best length of route: 47093.52086598701



Start length of route: 117353.8917114183

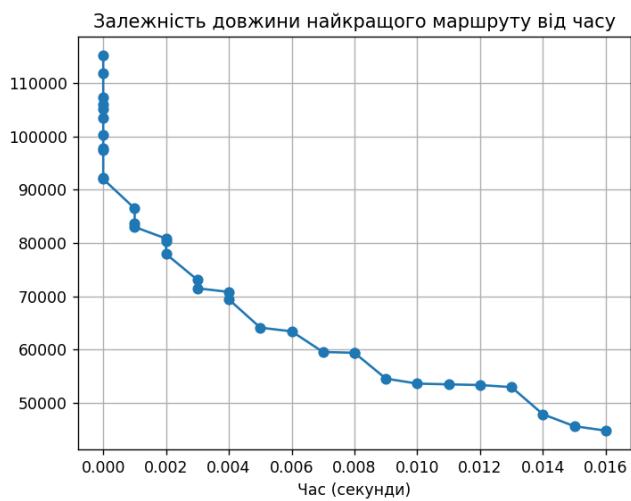
Best length of route: 45569.00944811554



**Start length of route: 101447.3575786743**

**Best length of route: 38818.51132747473**

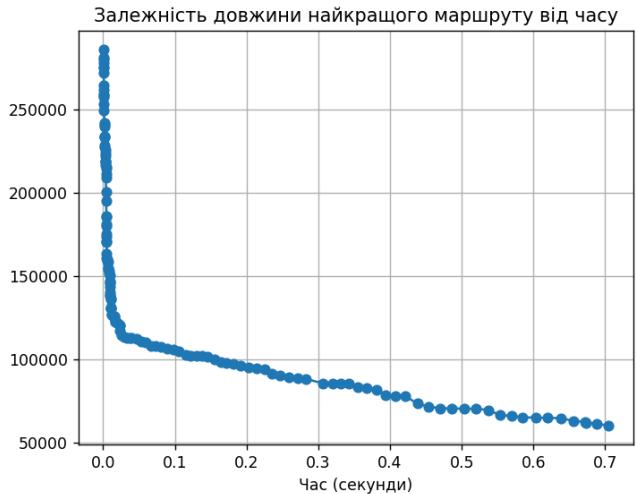




**Start length of route: 117400.63398165905**

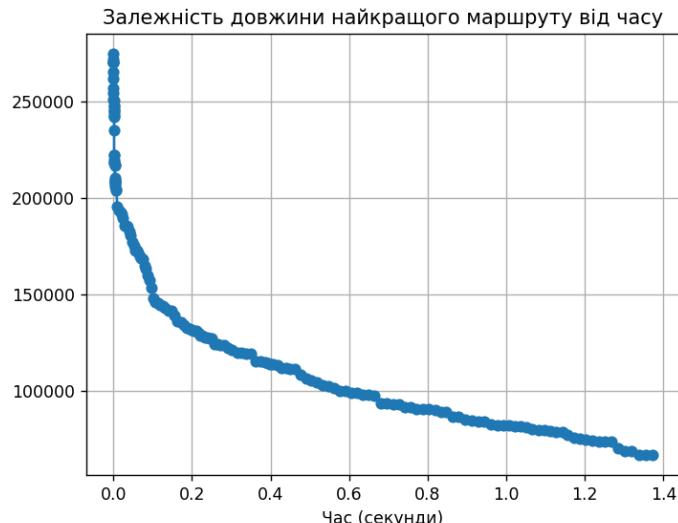
**Best length of route: 44782.41631585729**

**Збільшимо розмір мапи до 50 міст:**



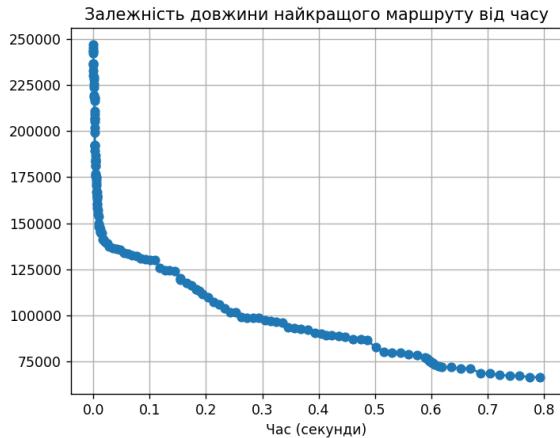
**Start length of route: 286238.7202653529**

**Best length of route: 59969.50874929439**



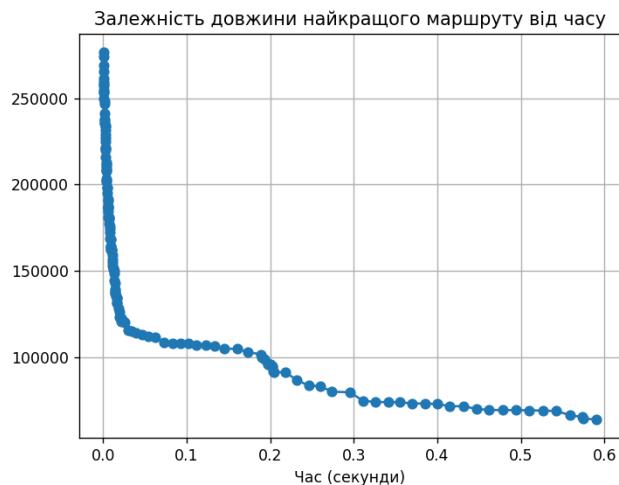
**Start length of route: 280856.30008735077**

**Best length of route: 66592.81599830094**



**Start length of route: 252360.85077249308**

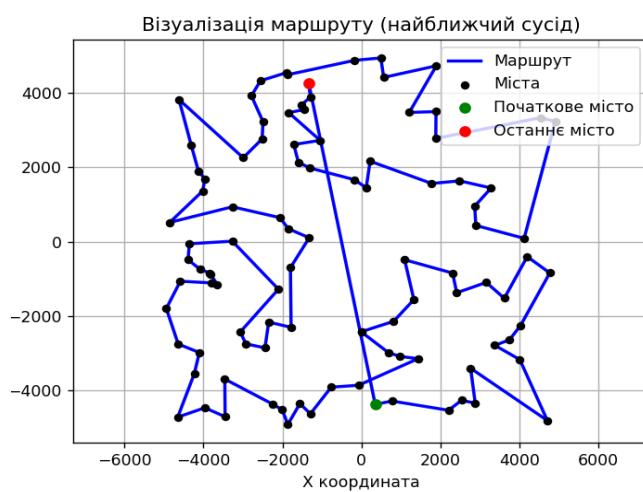
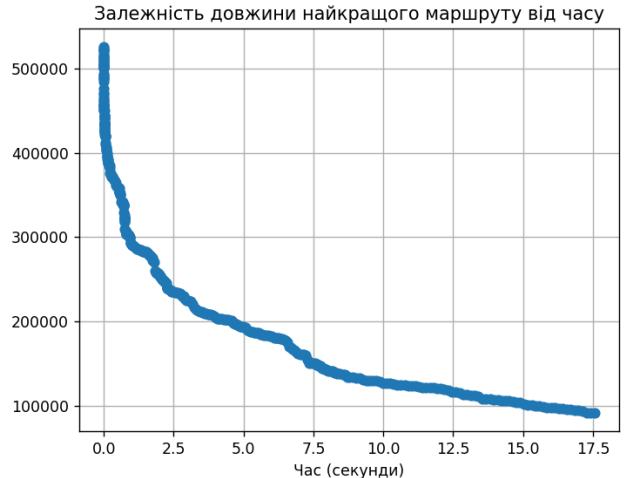
**Best length of route: 66427.97530705963**



**Start length of route: 280479.608324131**

**Best length of route: 63868.93691184380**

## Збільшимо розмір мапи до 100 міст:



**Start length of route: 527089.2083583098**

**Best length of route: 91988.54915286518**

**2 спроба:**

**Start length of route: 486173.7203103654**

**Best length of route: 85177.43460673094**

**3 спроба:**

**Start length of route: 535172.2566196343**

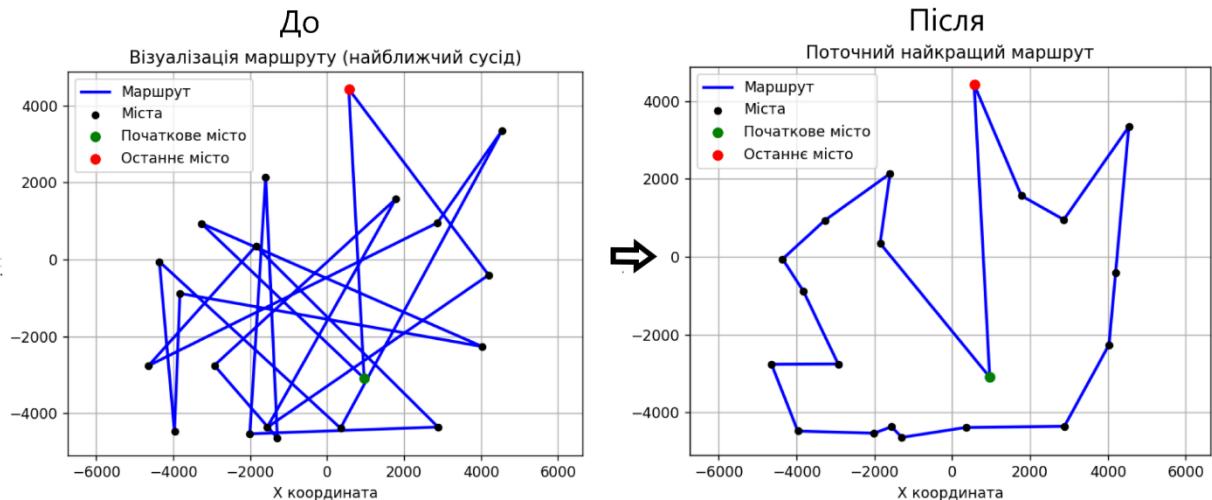
**Best length of route: 88089.57468143689**

Проведемо такі тести для мап розміром 1000 та 5000 і занесемо результати до таблиці

## Результати з випадковим маршрутом (повний перебір):

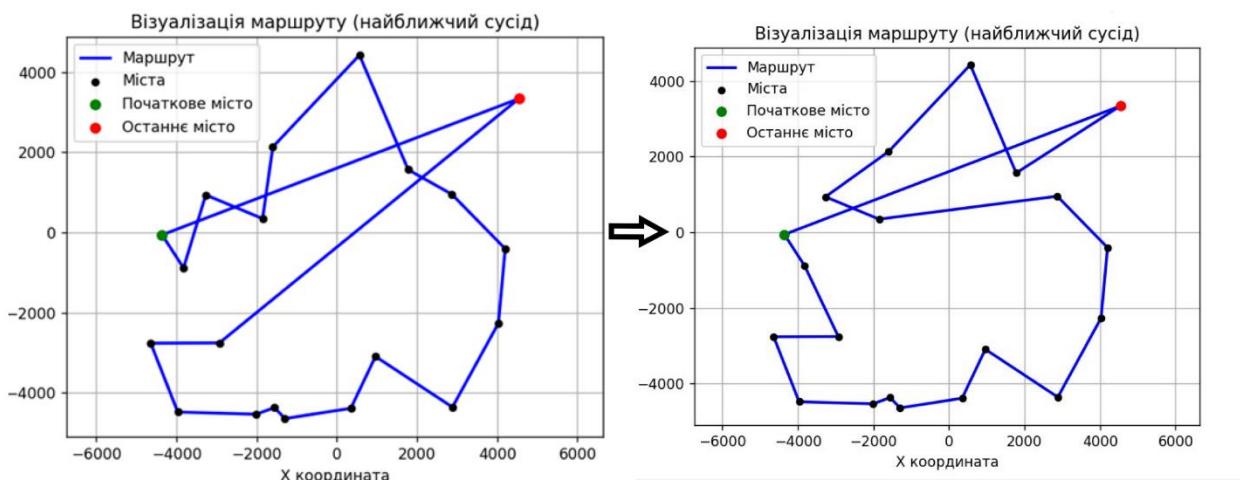
Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	38373.23	59969.50	91988.54	4022285.75	24795356.86
2	47093.52	66592.81	85177.43	3362800.35	25155016.78
3	45569.00	66427.97	88089.57	2993228.83	24548485.76
4	38818.51	63868.93	88002.49	3102152.78	24255816.86
5	44782.41	67450.79	87100.24	4072082.15	25104270.69

Алгоритм дуже добре розплутує початковий маршрут і перетворює маршрут так, як показано на рисунку:



Якщо мапа невеликого розміру, то алгоритм перебирає всі варіанти за мілісекунди, але при збільшенні розмірності алгоритм починає працювати довше. Якщо для 50 міст алгоритм працює 0.6 секунди, то для 100 міст вже 20 секунд. При великих розмірах мапи та рандомно згенерованому маршруті алгоритм видає дуже неоптимальні результати.

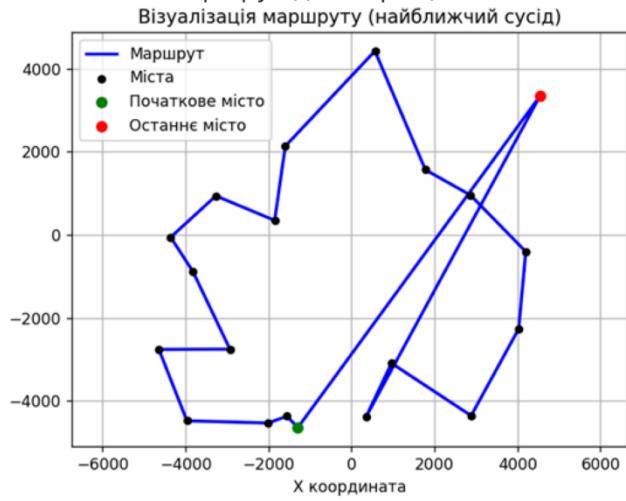
**Тепер протестуємо 2-opt алгоритм із початковим маршрутом, утвореним методом найближчого сусіда:**



**Start length of route: 50800.65022450286**

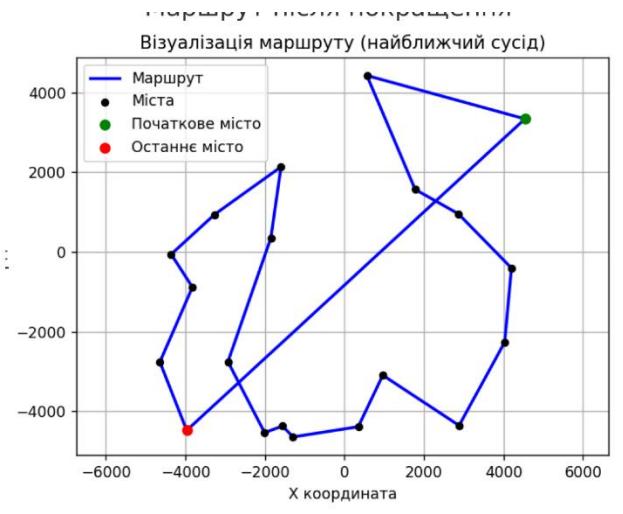
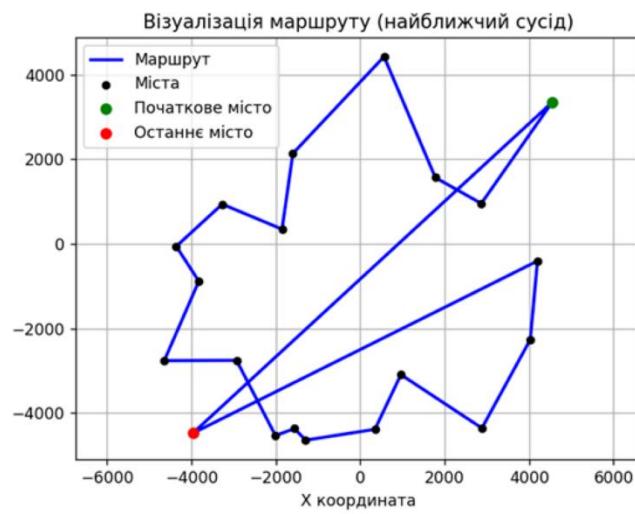
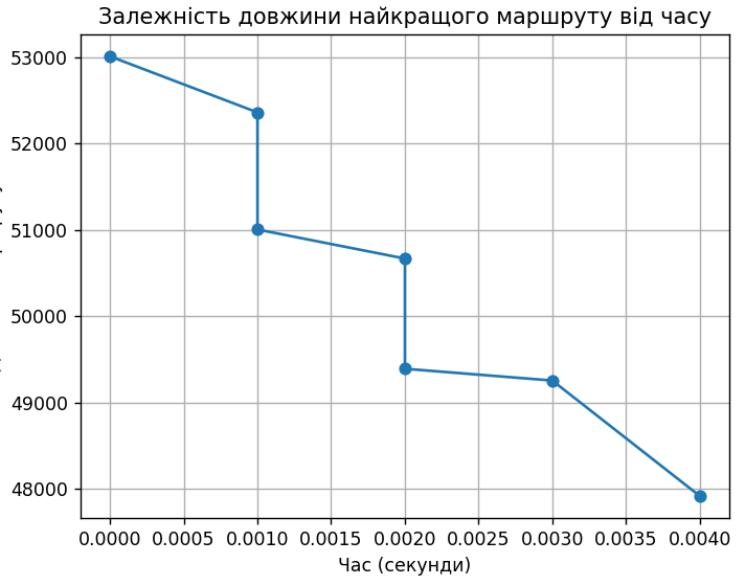
**Best length of route: 48373.62145401917**





**Start length of route: 50290.809307938835**

**Best length of route: 47680.82526132063**

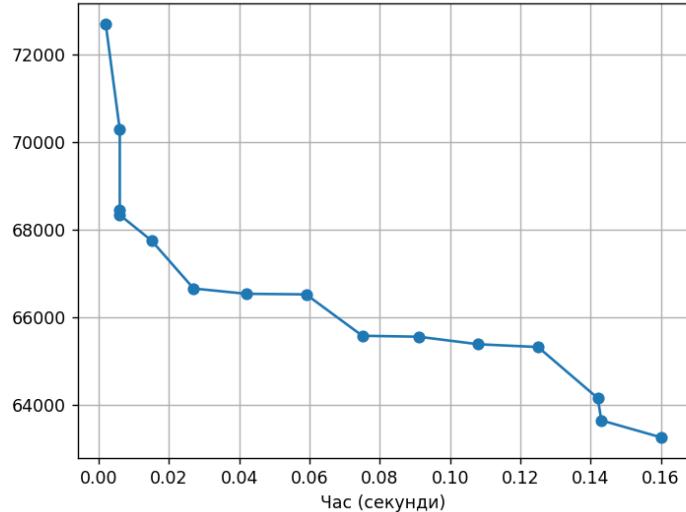


**Start length of route: 53144.77102844519**

**Best length of route: 47919.39569066746**

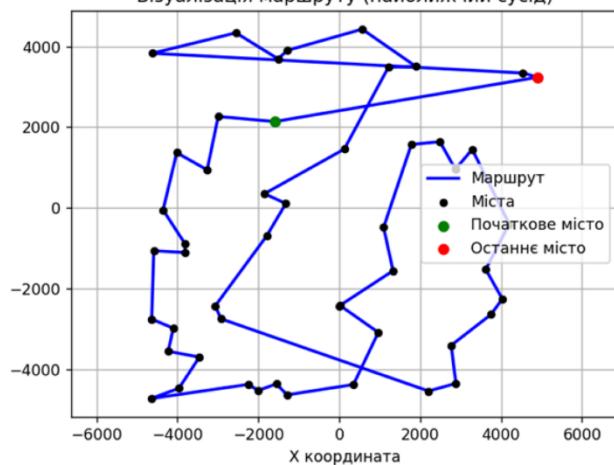
## Збільшимо кількість міст до 50:

Залежність довжини найкращого маршруту від часу



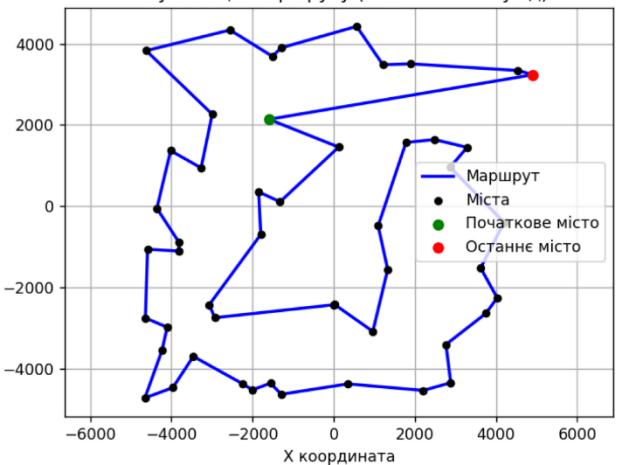
Маршрут до покращення

Візуалізація маршруту (найближчий сусід)



Маршрут після покращення

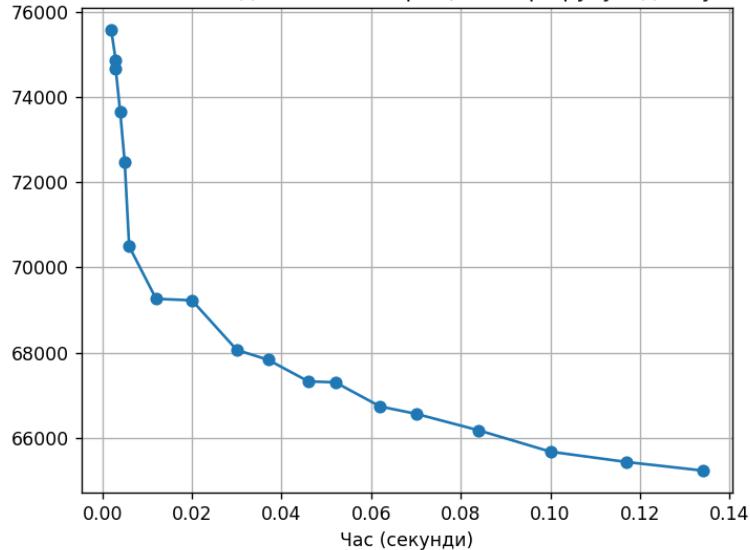
Візуалізація маршруту (найближчий сусід)

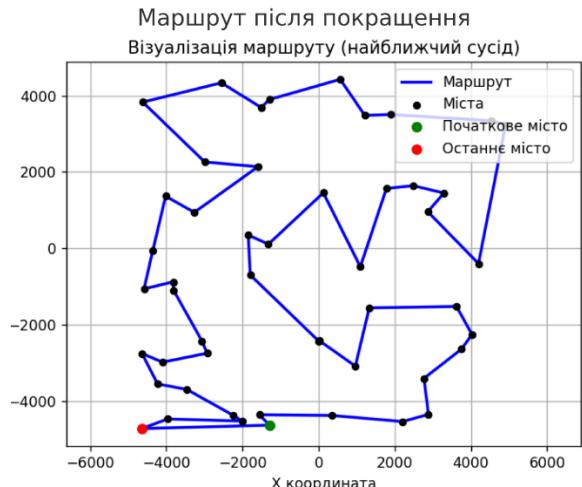
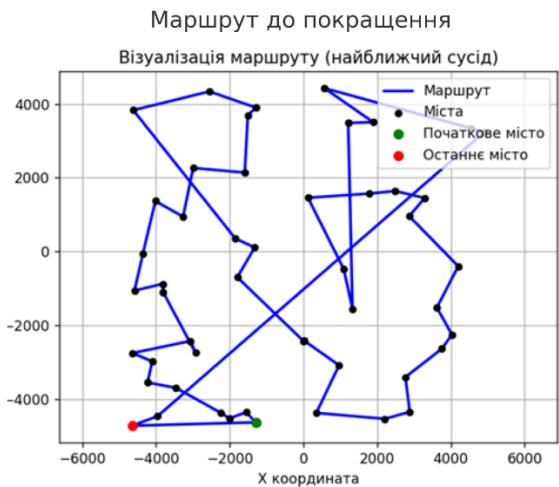


Start length of route: 72800.0641431605

Best length of route: 63257.42548832459

Залежність довжини найкращого маршруту від часу

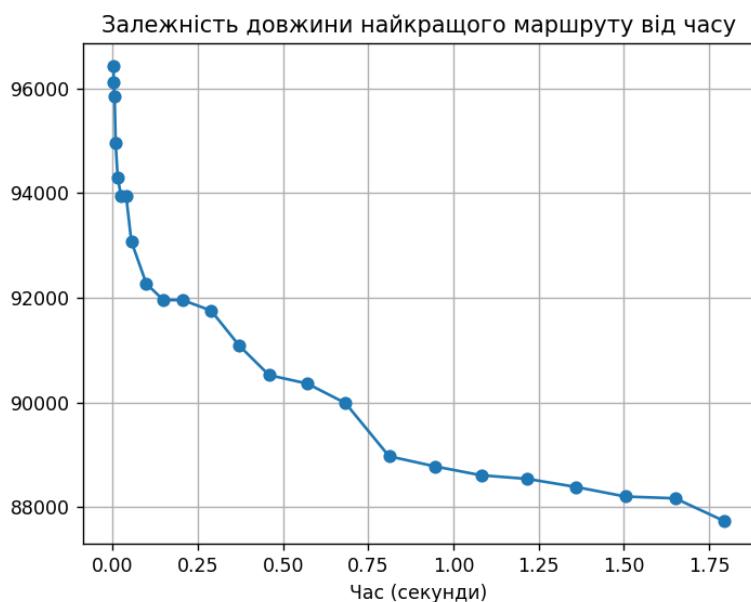
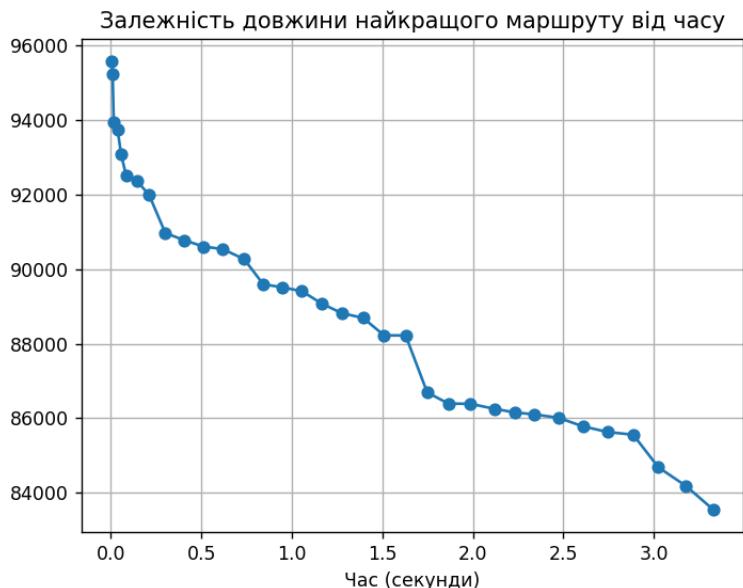


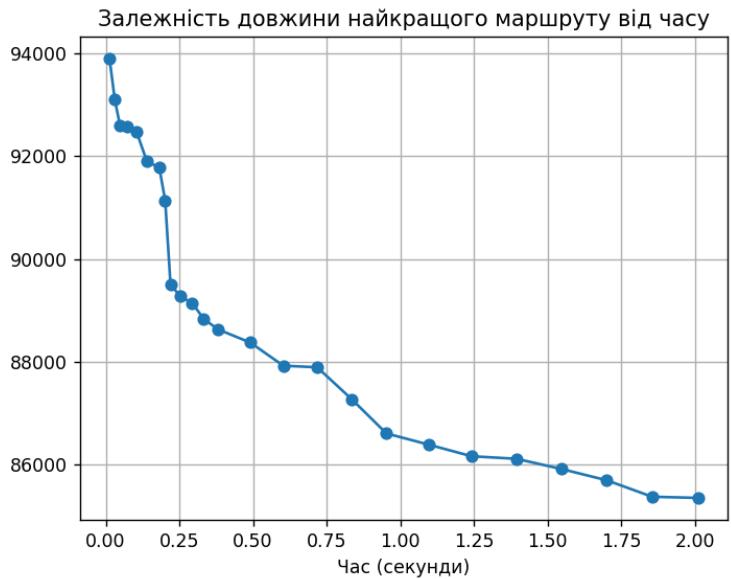


**Start length of route: 75872.28624131491**

**Best length of route: 65230.145356742636**

**Збільшимо кількість міст до 100:**

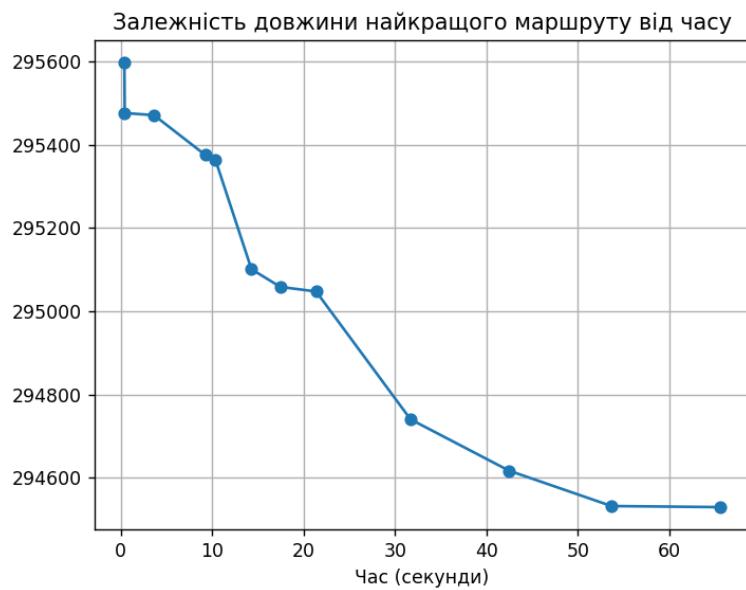




**Збільшимо кількість міст та заповнимо таблицю.**

**Результати з методом найближчого сусіда (повний перебір):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	48373.62	63685.85	91935.47	294529.85	634316.51
2	47680.82	65230.14	86207.16	289805.94	629887.07
3	47919.39	67688.35	83550.81	292440.66	628456.40
4	44385.51	63257.42	87735.04	285667.81	629108.73
5	38054.96	66812.94	85357.17	297157.64	620321.02



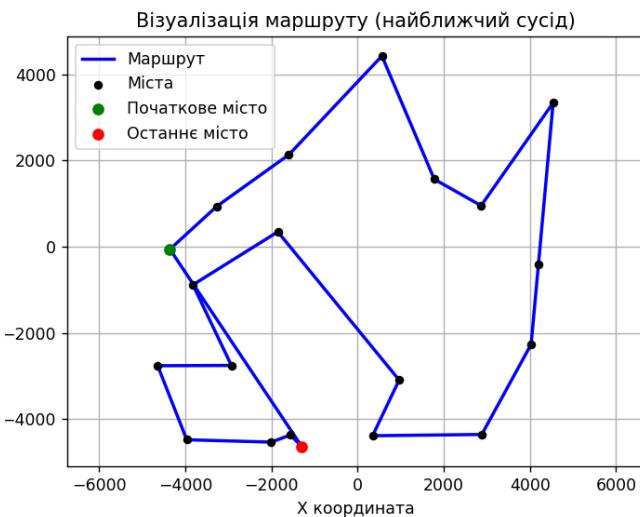
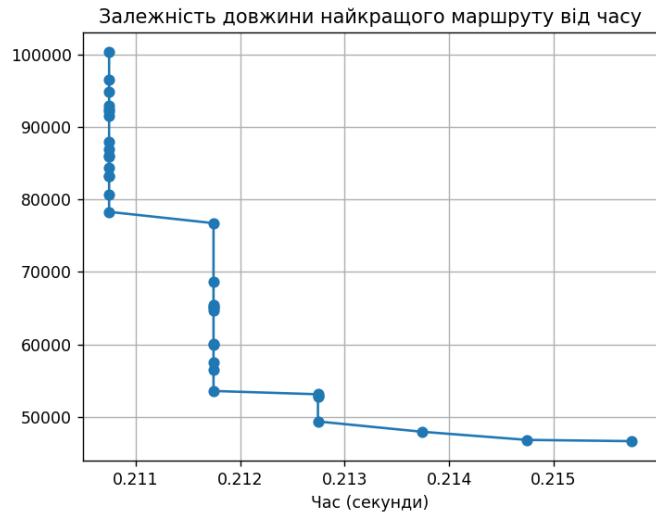
**Start length of route: 295677.0335457827**

**Best length of route: 294529.85898575356**

Для великої кількості міст покращення майже не відбуваються.

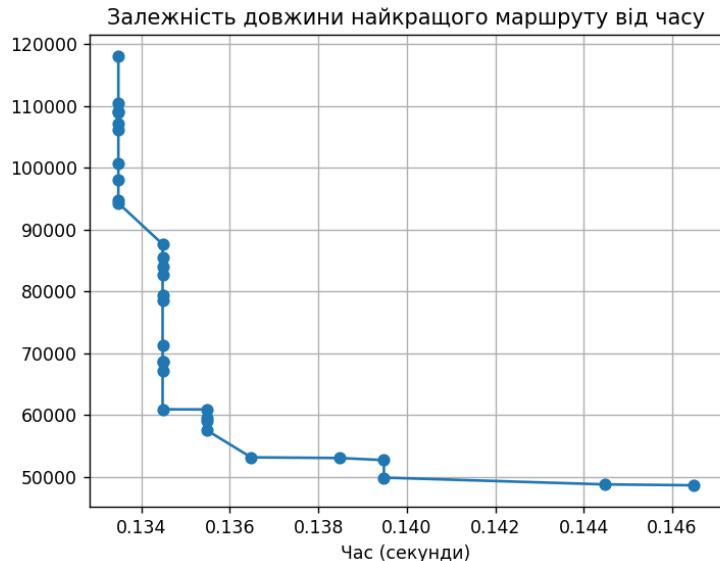
**Протестуємо 2-opt алгоритм, у якому індекси обираються рандомно:**

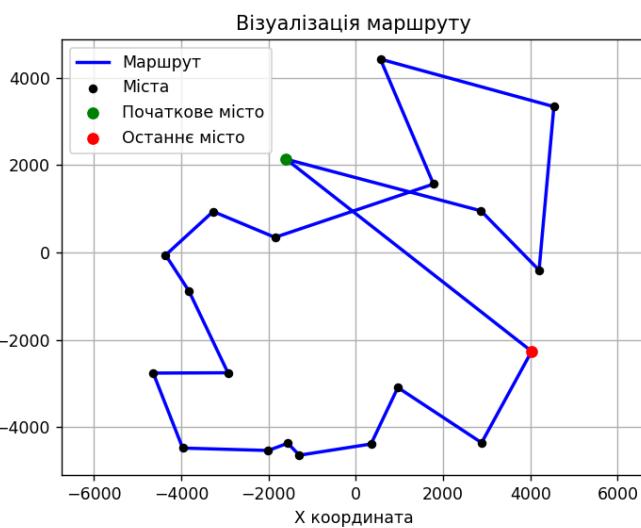
Спочатку будемо генерувати початковий маршрут рандомно:



Start length of route: 100364.70796874155

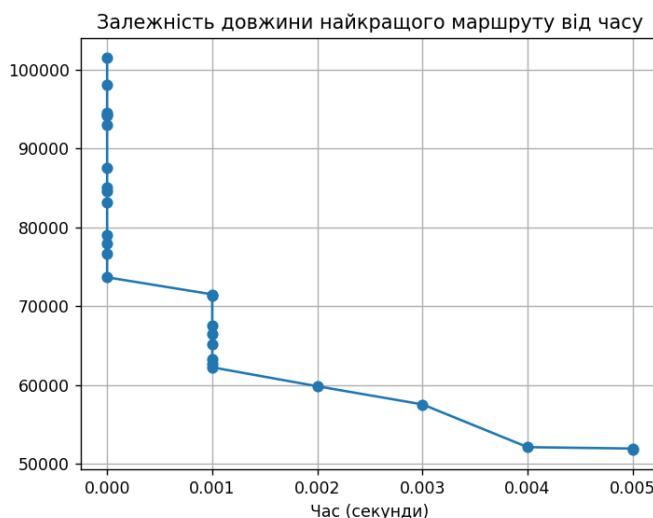
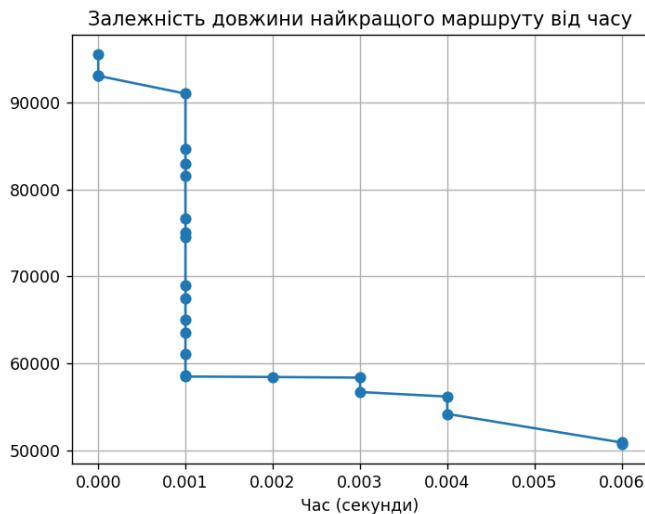
Best length of route: 46640.47263825395



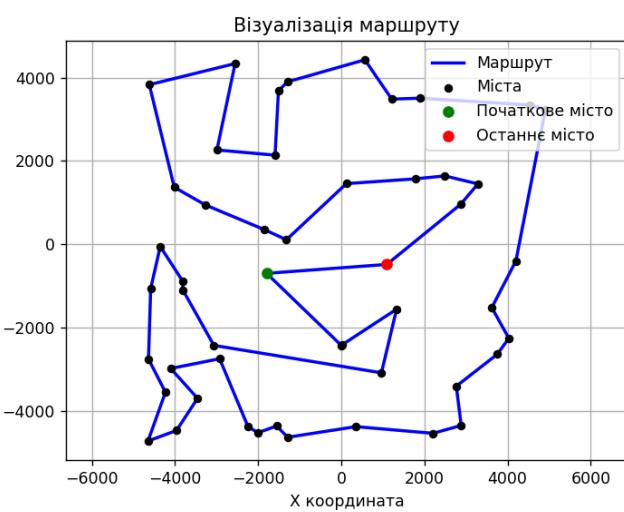
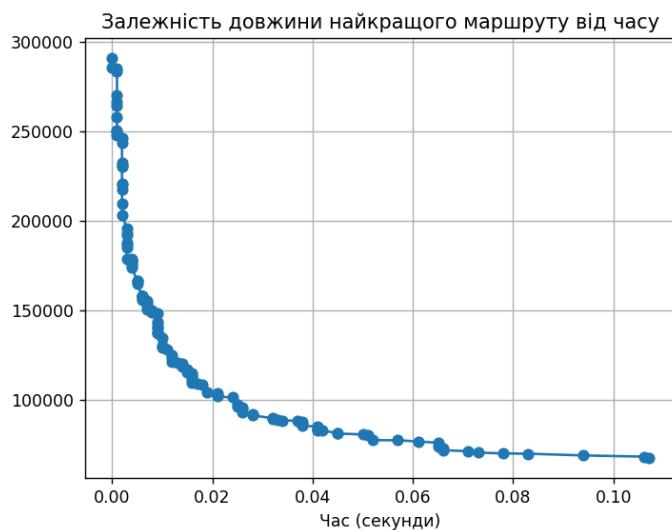


**Start length of route: 119428.22259039973**

**Best length of route: 48683.329174274906**

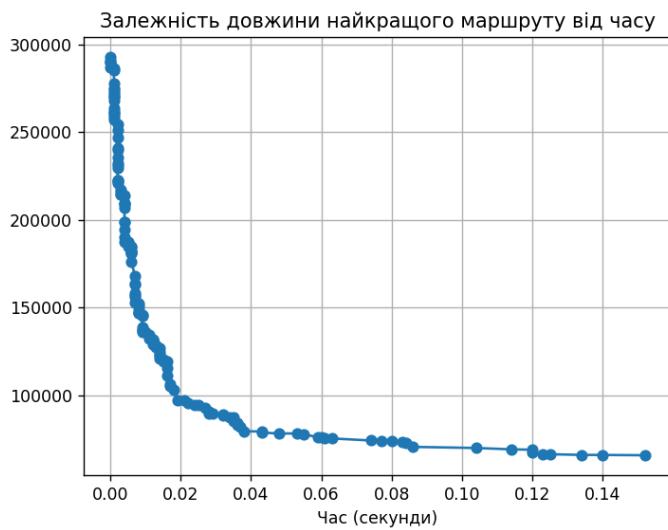


**Збільшимо кількість міст до 50:**



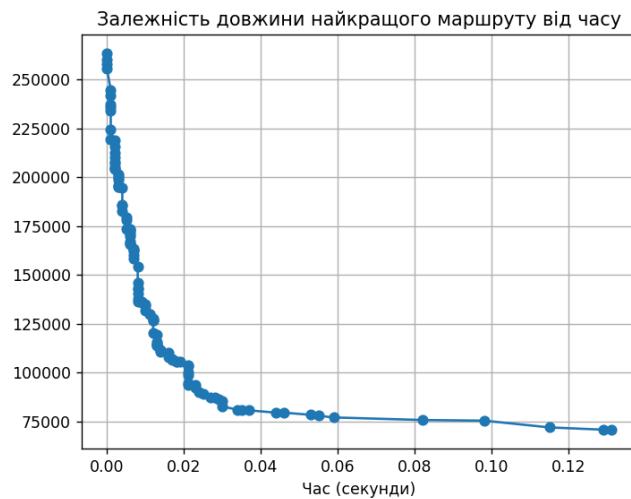
**Start length of route: 298381.6859762449**

**Best length of route: 67588.08493549086**



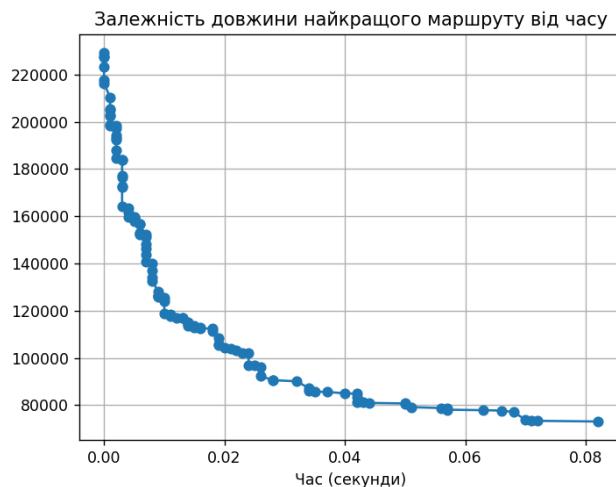
**Start length of route: 293509.4524107351**

**Best length of route: 65677.29634848729**



**Start length of route: 263977.28278096474**

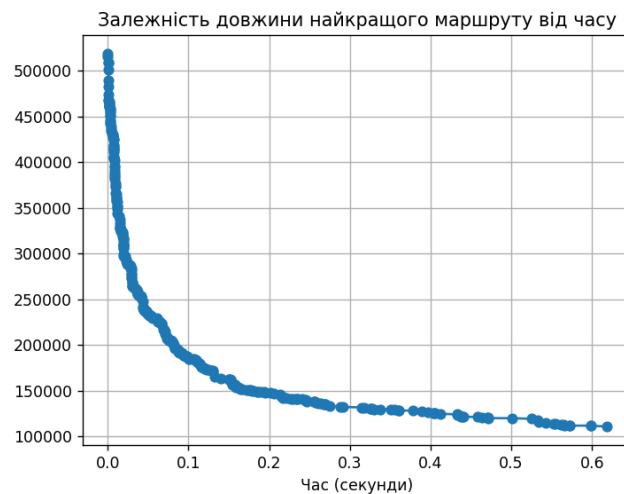
**Best length of route: 70826.91957795722**



**Start length of route: 234307.26203145305**

**Best length of route: 73029.07321056584**

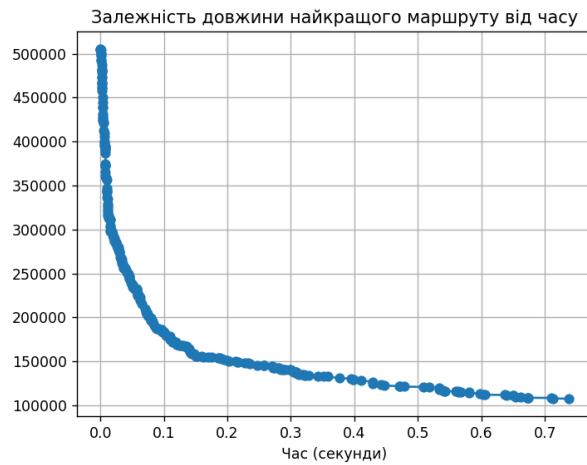
**Збільшимо кількість міст до 100:**





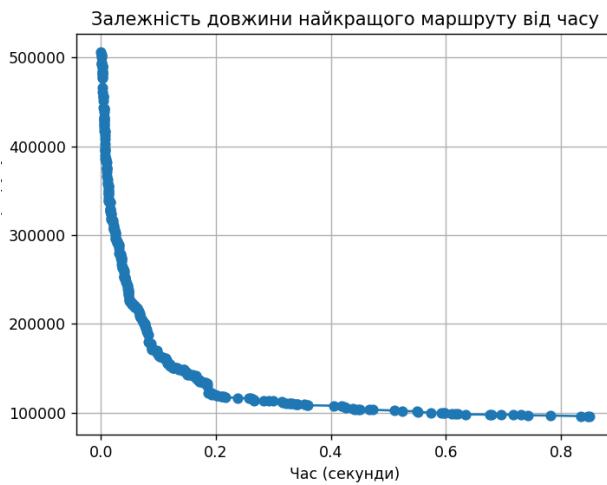
**Start length of route: 523413.9052975345**

**Best length of route: 110965.19975446266**



**Start length of route: 505926.6793777225**

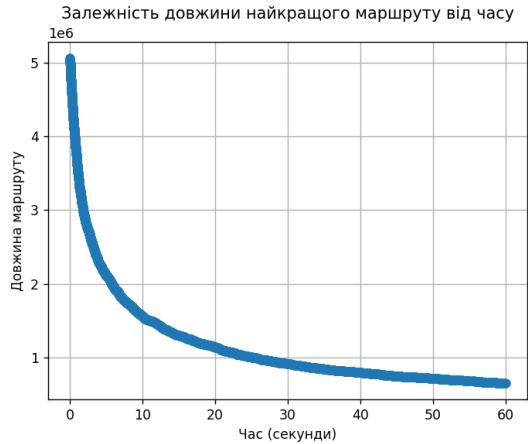
**Best length of route: 107499.72919687776**



**Start length of route: 507637.8760101912**

**Best length of route: 95786.27108913002**

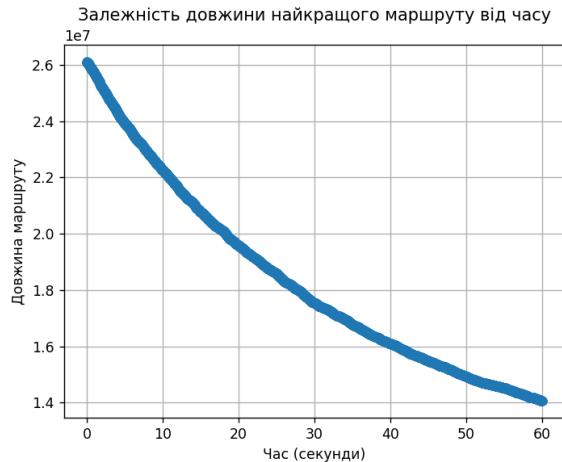
### Збільшимо кількість міст до 1000:



**Start length of route: 5064352.664379846**

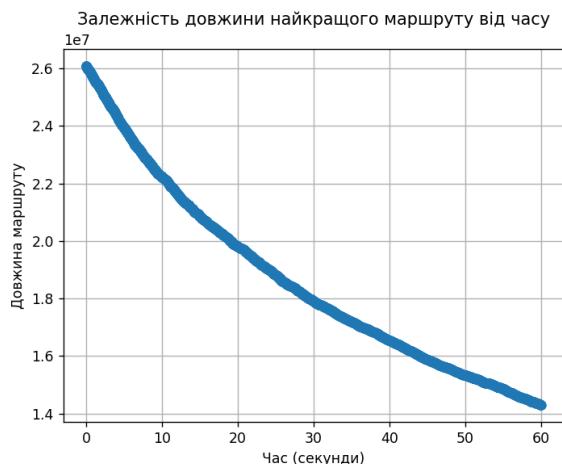
**Best length of route: 653765.7887434827**

### Збільшимо кількість міст до 5000:



**Start length of route: 26110548.215297047**

**Best length of route: 14070330.291096926**



**Start length of route: 26090863.0760411**

**Best length of route: 14296273.450832289**

**Результати роботи 2-opt алгоритму з рандомним вибором індексів  
(випадковий початковий маршрут):**

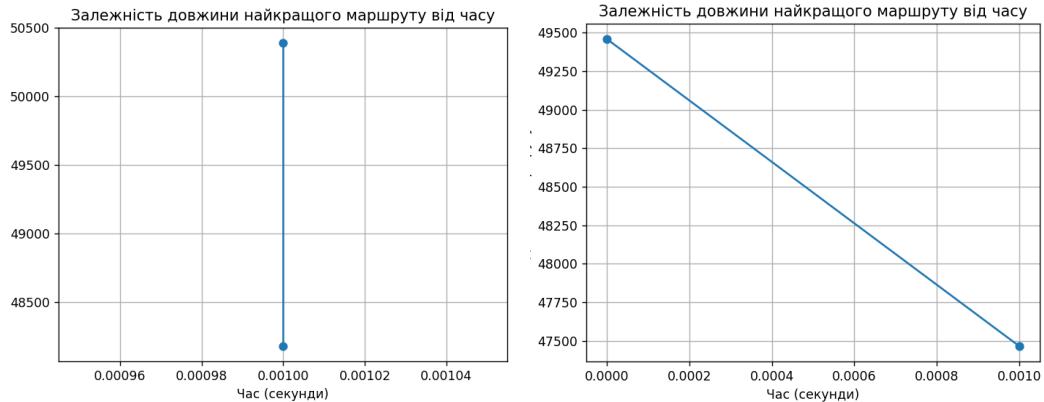
Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	46640.47	67588.08	110965.19	653765.78	14070330.29
2	48683.32	65677.29	107499.72	637540.61	14296273.45
3	51422.59	70826.91	95786.27	655974.51	14470623.74
4	50719.25	73029.07	104225.40	665402.29	14346914.44
5	51822.67	71369.93	92893.72	664503.03	14094085.66

**Результати з випадковим маршрутом (повний перебір):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	38373.23	59969.50	91988.54	4022285.75	24795356.86
2	47093.52	66592.81	85177.43	3362800.35	25155016.78
3	45569.00	66427.97	88089.57	2993228.83	24548485.76
4	38818.51	63868.93	88002.49	3102152.78	24255816.86
5	44782.41	67450.79	87100.24	4072082.15	25104270.69

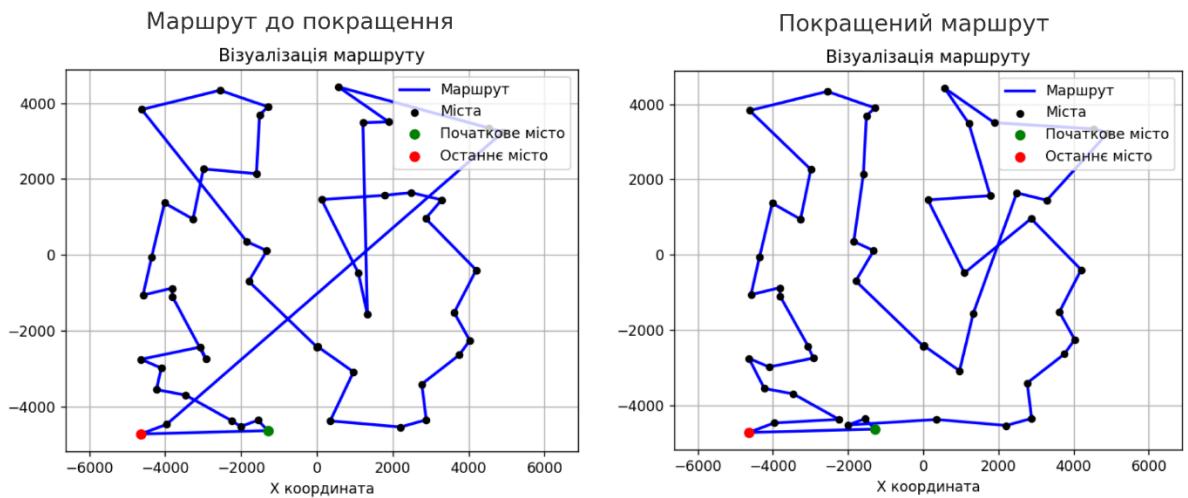
Якщо порівняти результати роботи 2-opt алгоритму з рандомним вибором індексів та 2-opt алгоритму з повним перебором, то видно, що для малих розмірів мапи 2-opt з повним перебором впорується краще, але для дуже великих розмірів (у нашому випадку 5000 міст) алгоритм з випадковим вибором індексів показує кращі результати, але при цьому вони все ще далекі від оптимальних.

## Протестуємо 2-opt алгоритм з маршрутом утвореним за допомогою методу найближчого сусіда:

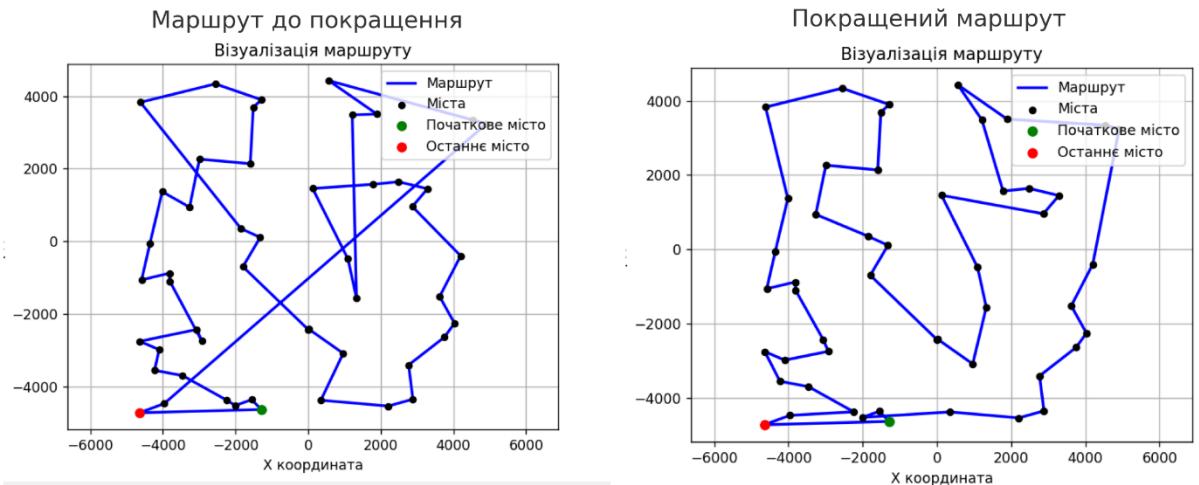


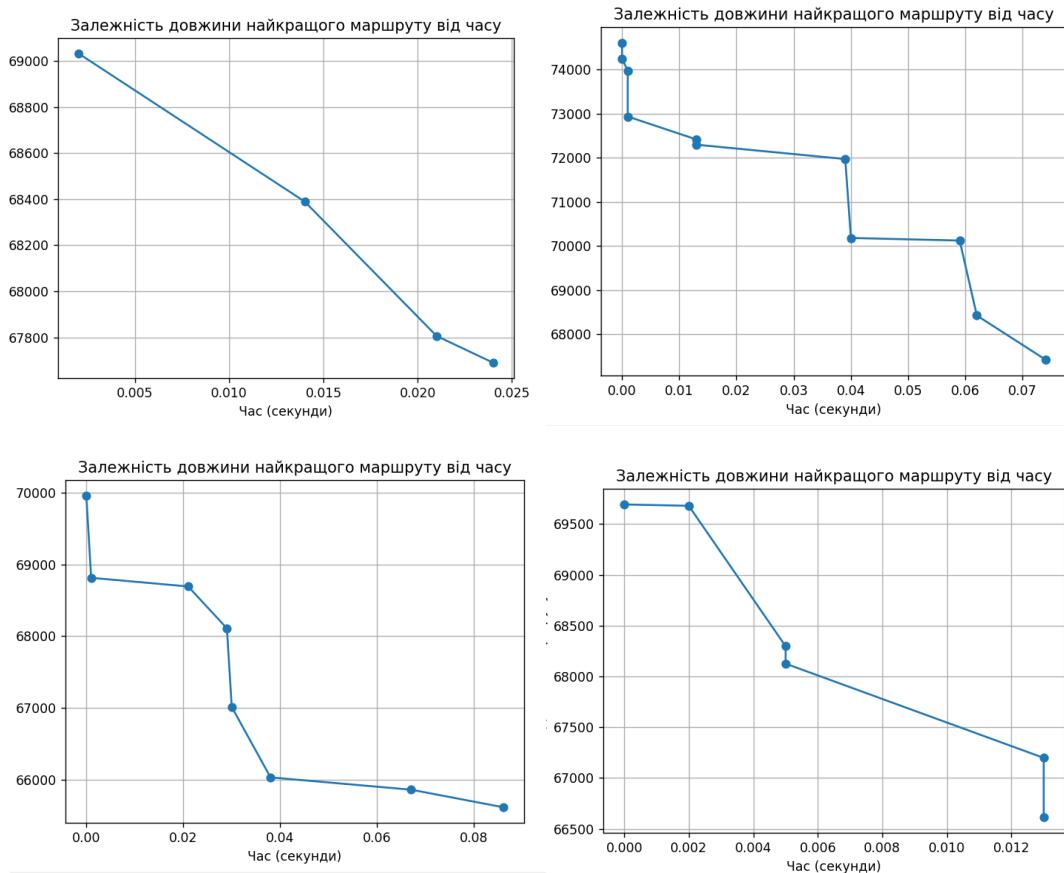
(інші графіки з розміром мапи 20 міст виглядають майже так само (тобто покращень майже немає)

### Збільшимо кількість міст до 50:



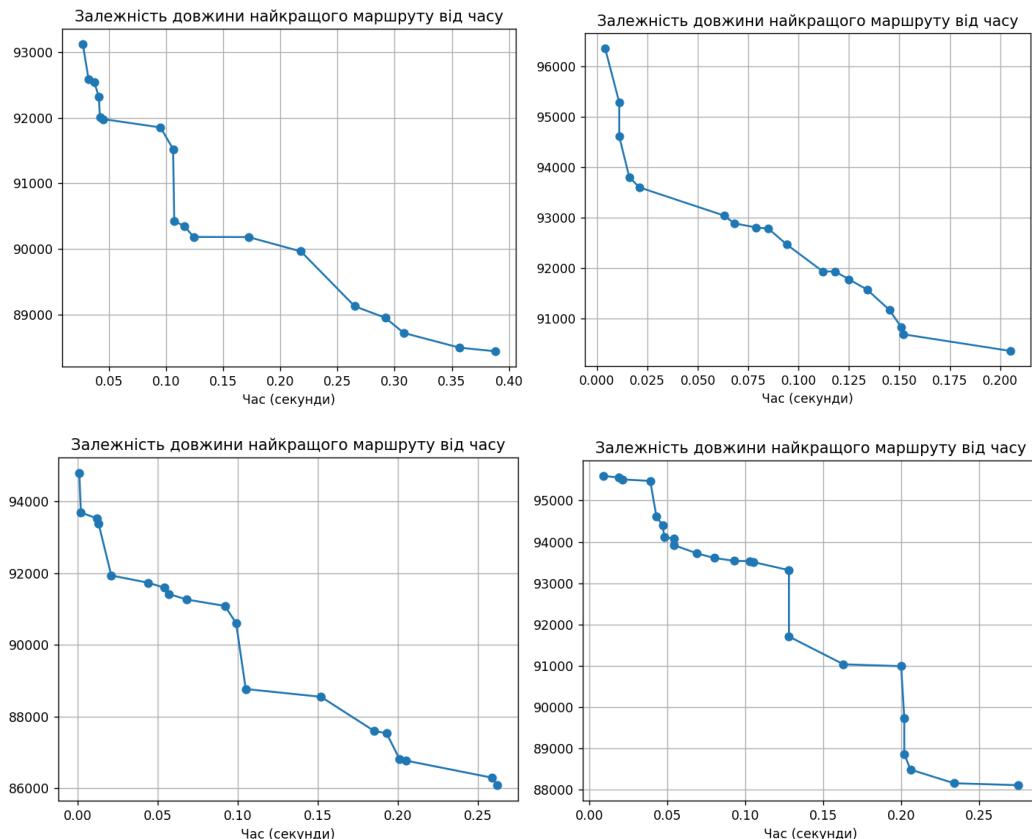
Різні запуски дають різні результати



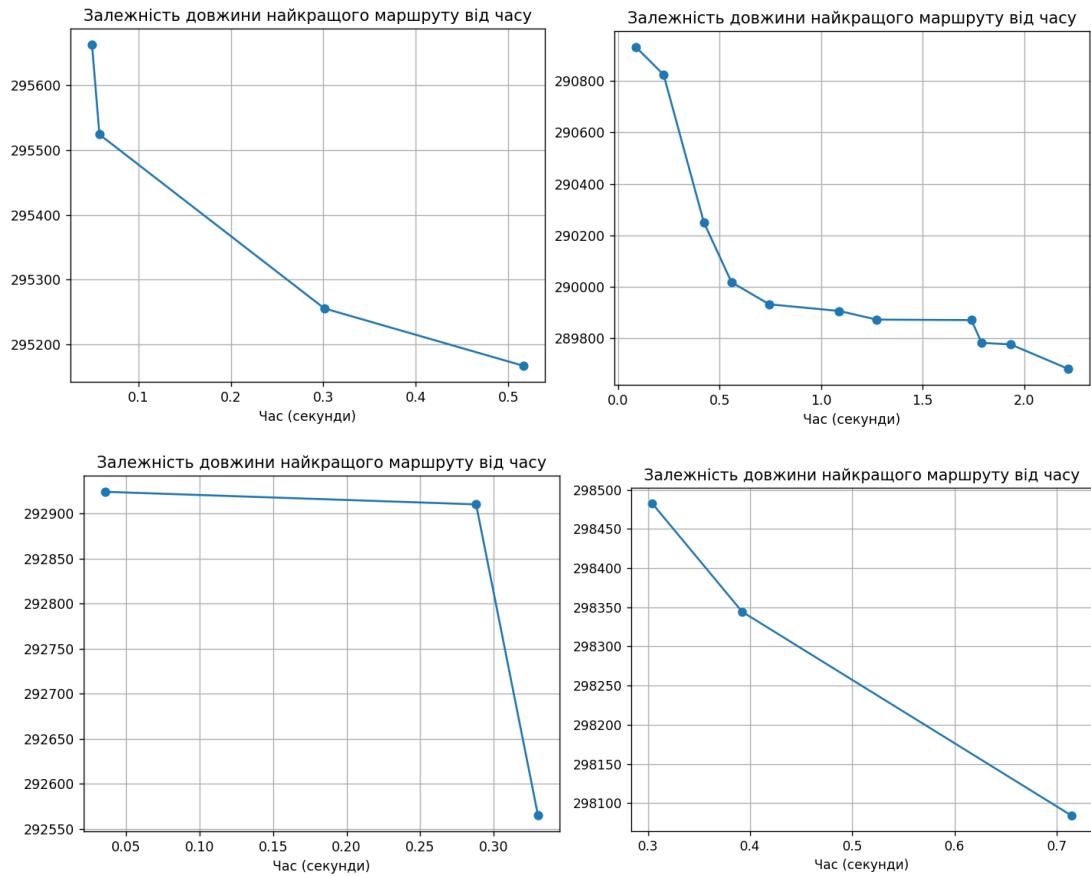


З невеликою кількістю міст покращень небагато.

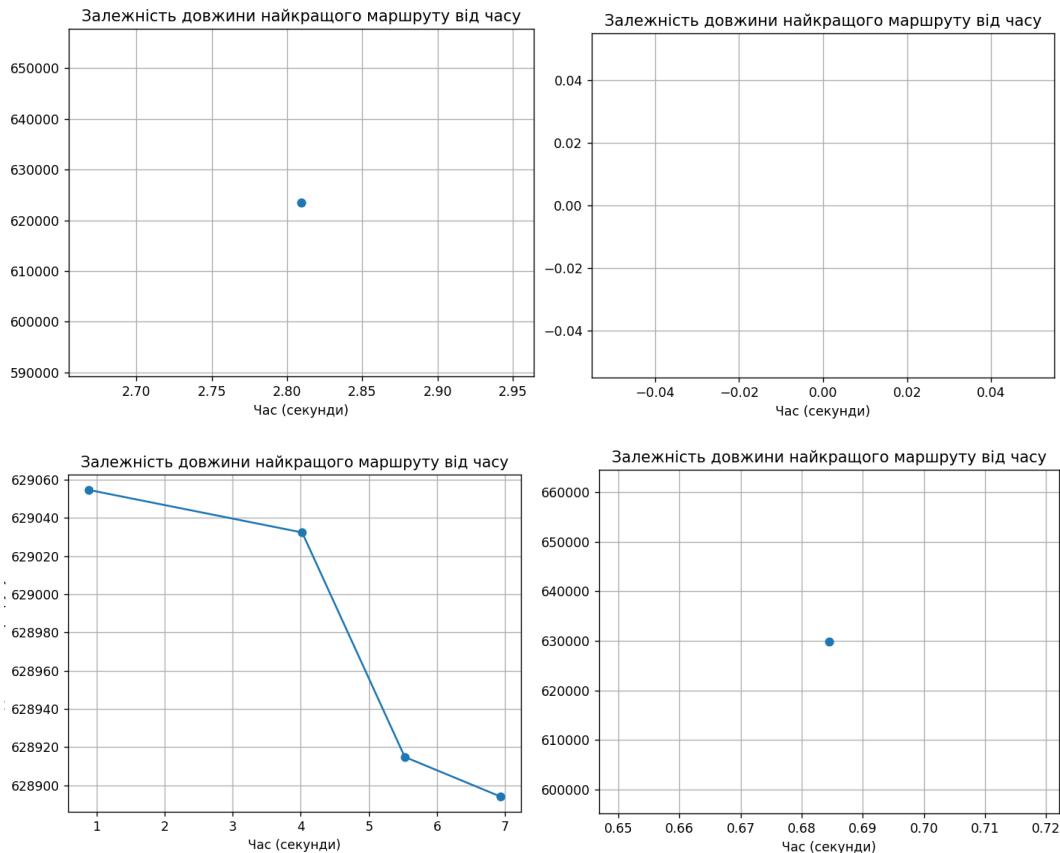
### Збільшимо кількість міст до 100:



## Збільшимо кількість міст до 1000:



## Збільшимо кількість міст до 5000:



**Результати роботи 2-opt алгоритму з маршрутом утвореним за допомогою методу найближчого сусіда:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	48178.87	67425.83	91173.51	295166.76	636811.65
2	48133.83	65256.17	88114.98	289682.62	629859.62
3	51852.51	67689.76	86092.52	292565.16	628894.10
4	47314.41	66947.06	90349.69	289565.14	629126.00
5	38054.96	66613.78	88439.41	298084.06	623492.08

**Результати з методом найближчого сусіда (повний перебір):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	48373.62	63685.85	91935.47	294529.85	634316.51
2	47680.82	65230.14	86207.16	289805.94	629887.07
3	47919.39	67688.35	83550.81	292440.66	628456.40
4	44385.51	63257.42	87735.04	285667.81	629108.73
5	38054.96	66812.94	85357.17	297157.64	620321.02

**Результати роботи методу найближчого сусіда:**

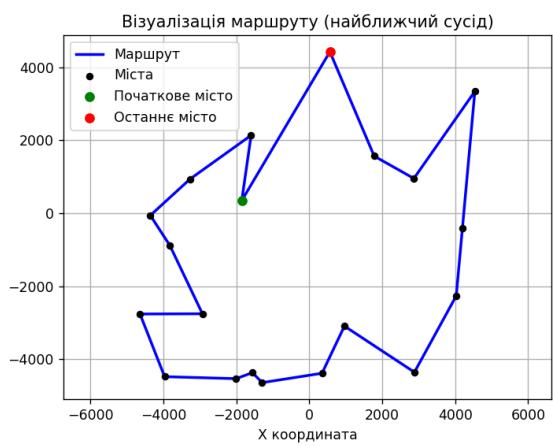
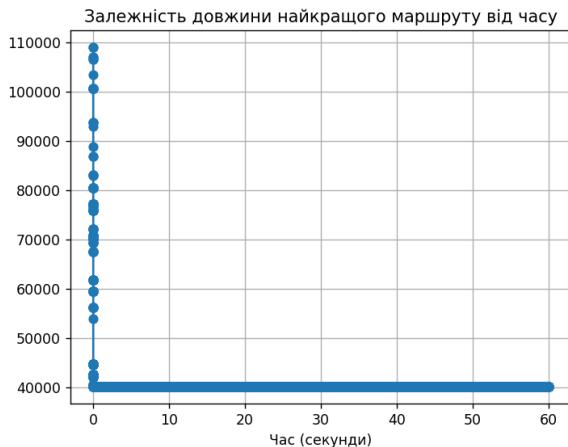
Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	50800.65	69532.21	103891.36	295677.03	636811.65
2	50290.80	75872.28	96889.94	291282.21	629902.49
3	53144.77	70008.85	95804.07	292995.68	629127.83
4	51006.21	72800.06	96521.28	289590.19	629126.00
5	38054.96	69828.57	93920.09	298727.56	623537.76

2-opt алгоритм з випадковим вибором індексів поводить себе так само, як і 2-opt з повним перебором: якщо відразу приходить досить оптимальний маршрут, то поїдається майже не відбувається. Знайти кардинально кращий маршрут алгоритму не вдається.

## Протестуємо алгоритм імітації відпалу:

Для того, щоб знаходити найкращі маршрути будемо змінювати параметри алгоритму під час нової спроби.

**Спочатку задамо такі параметри:  $T = 10$ ,  $T_{end} = 0$ ,  $a = 0.99$**

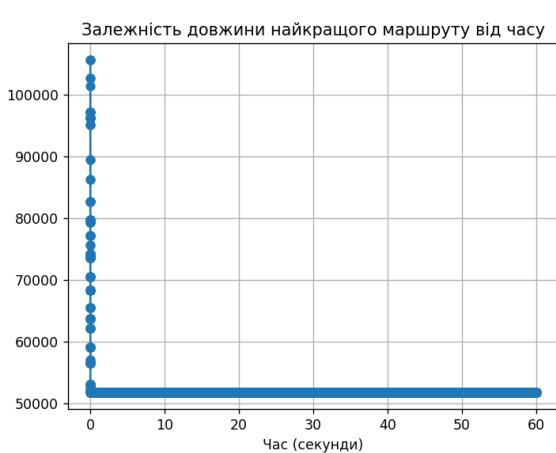


Бачимо, що алгоритм дуже швидко визначив хороший маршрут і далі намагався його покращити.

**Start length of route: 109011.06493338855**

**Best length of route: 40168.631242239244**

Тепер задамо такі параметри:  $T = 1$ ,  $T_{end} = 0$ ,  $a = 0.99$

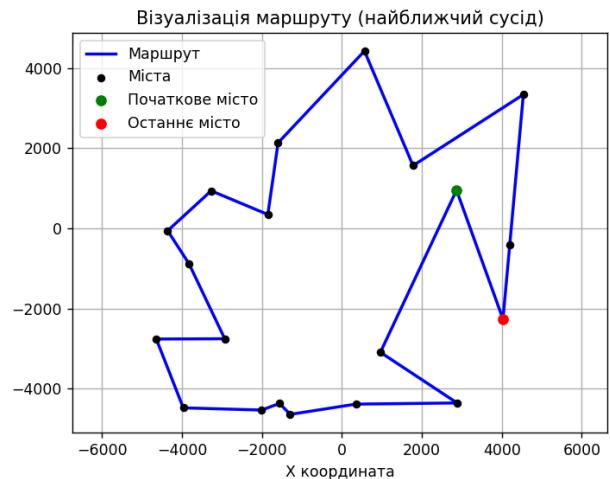
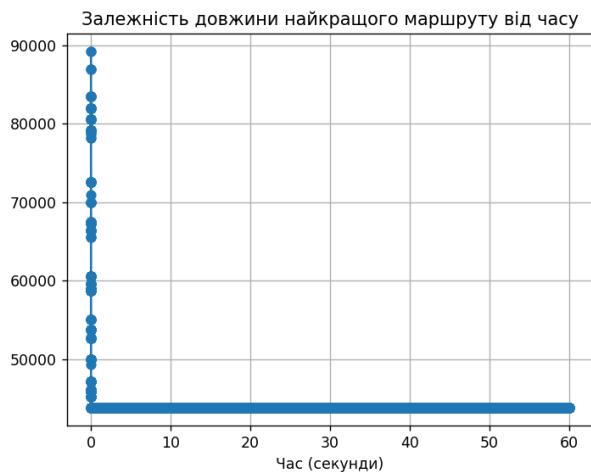


**Start length of route: 110143.79820535744**

**Best length of route: 51842.21917993589**

Бачимо, що алгоритм знову прийшов до одного варіанту та намагався його покращити. Але цього разу варіант не є оптимальним.

**Збільшимо початкову температуру до 100 градусів:**

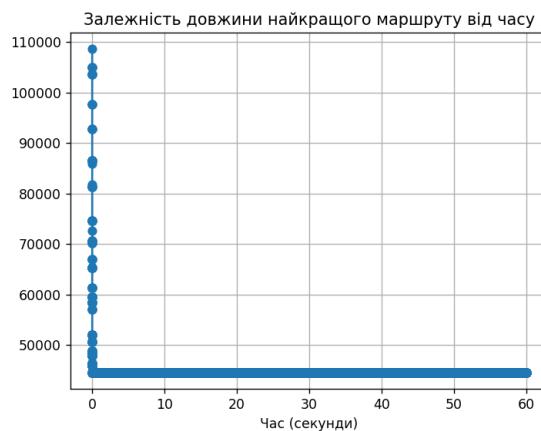


**Start length of route: 89247.90910343516**

**Best length of route: 43780.331777561616**

З високими початковими температурами алгоритм поки знаходить кращі рішення.

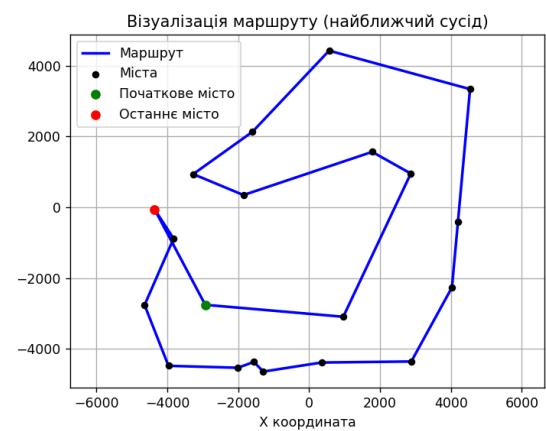
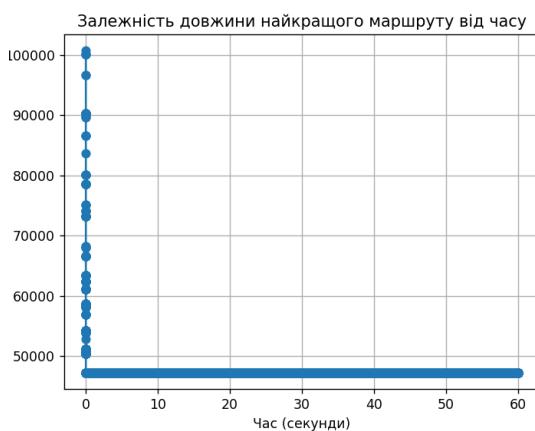
**Тепер задамо такі параметри:  $T = 100$ ,  $T_{end} = 0$ ,  $a = 0.995$**



**Start length of route: 109675.78392611461**

**Best length of route: 44578.504671963485**

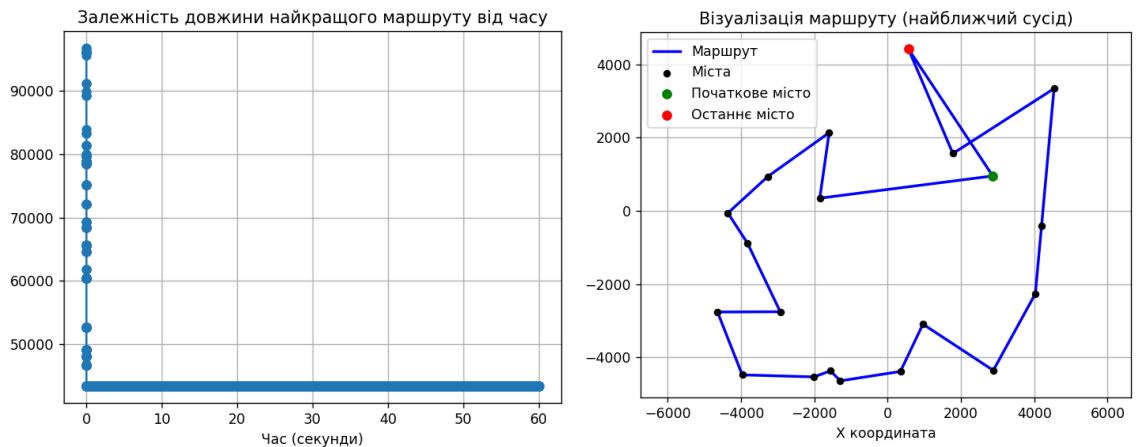
**Тепер задамо такі параметри:  $T = 5$ ,  $T_{end} = 0$ ,  $a = 0.995$**



**Start length of route: 100711.21121795524**

**Best length of route: 47259.7535502874**

Тепер задамо такі параметри:  $T = 10$ ,  $T_{end} = 0$ ,  $a = 0.995$



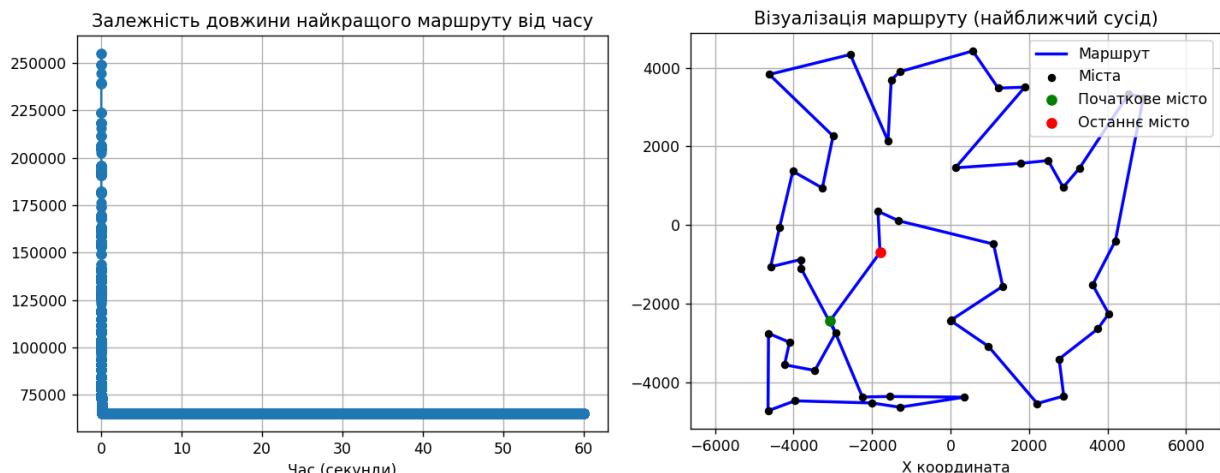
**Start length of route: 99269.05888512036**

**Best length of route: 43471.531821589626**

Можемо зробити висновок, що оптимальним значенням початкової температури є 10.

**Збільшимо розмір мапи до 50 міст:**

**Параметри:  $T = 10$ ,  $T_{end} = 0$ ,  $a = 0.99$**



**Start length of route: 254939.65141670057**

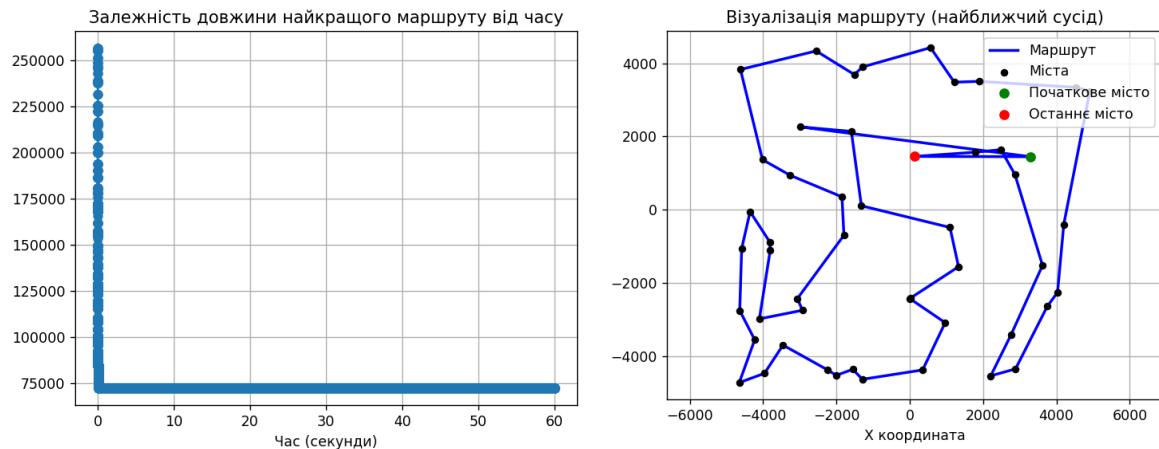
**Best length of route: 65159.59163517421**

**Спроба 2:**

**Start length of route: 277918.437354283**

**Best length of route: 70135.19079938224**

**Параметри:  $T = 20$ ,  $T_{end} = 0$ ,  $a = 0.99$**



**Start length of route: 259679.3813917753**

**Best length of route: 72418.74939437956**

**Спроба 2:**

**Start length of route: 289918.2417717835**

**Best length of route: 71335.49379268062**

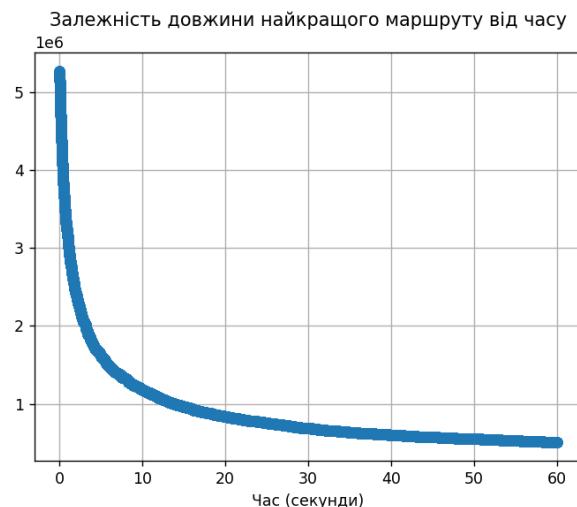
**Спроба 3:**

**Start length of route: 282773.015178341**

**Best length of route: 73553.80209766082**

Залишимо початкову температуру **10**, як оптимальну та будемо змінювати значення параметра **a**: перші 2 запуски виконаємо з  $a = 0.99$ , інші 3 з  $a = 0.995$ .  $0.99$  – дає більш оптимальні результати.

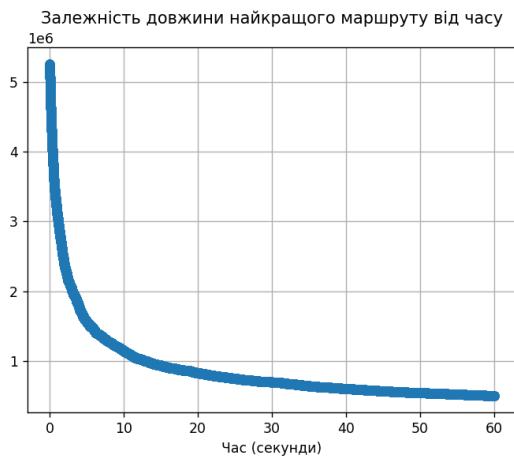
**Збільшимо кількість міст до 1000:**



**Параметри:  $T = 10$ ,  $T_{end} = 0$ ,  $a = 0.99$**

**Start length of route: 5273718.579436277**

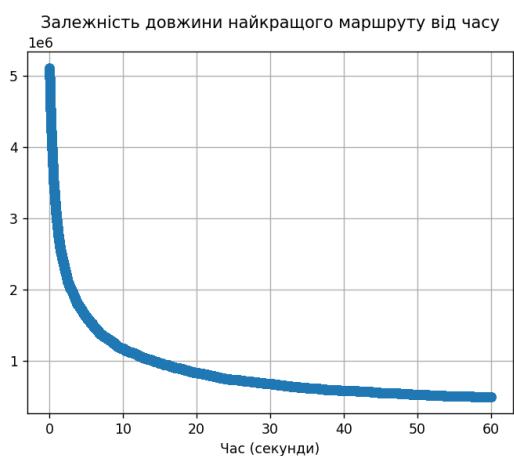
**Best length of route: 563111.016497225**



**Параметри:  $T = 100$ ,  $T_{end} = 0$ ,  $a = 0.99$**

**Start length of route: 5260803.709240571**

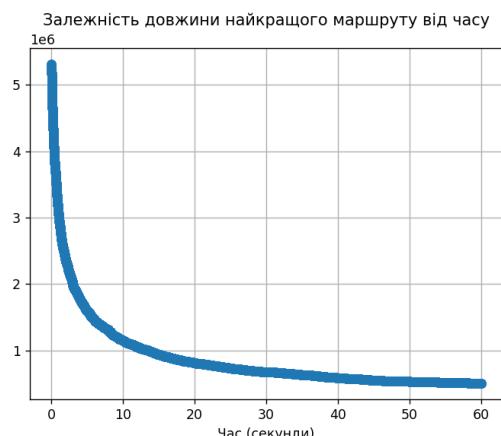
**Best length of route: 499353.19429756**



**Параметри:  $T = 1000$ ,  $T_{end} = 0$ ,  $a = 0.99$**

**Start length of route: 5114715.199651964**

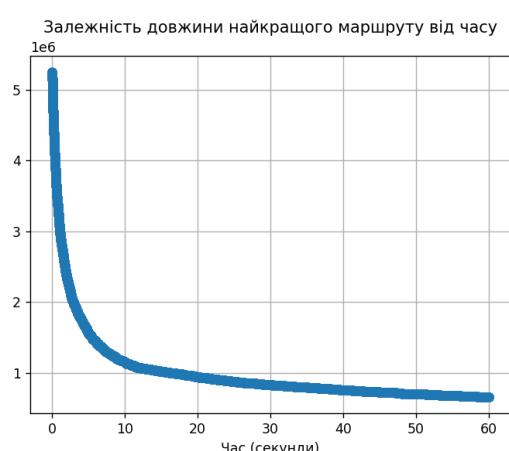
**Best length of route: 490705.52301521716**



**Параметри:  $T = 1000$ ,  $T_{end} = 0$ ,  $a = 0.99$**

**Start length of route: 5314168.052944642**

**Best length of route: 502910.9389816308**



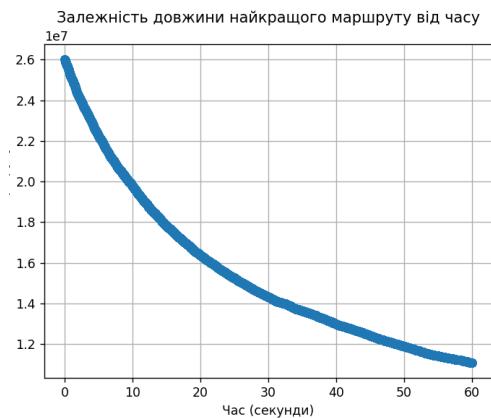
**Параметри:  $T = 10$ ,  $T_{end} = 0$ ,  $a = 0.99$**

**Start length of route: 5250129.327080997**

**Best length of route: 660287.0683509032**

Для великих наборів даних важко визначити оптимальне значення температури. Оскільки, параметр  $T = 10$  може давати результат 496989.84 так і 660287.06. Збільшення температури може давати більш стабільні результати.

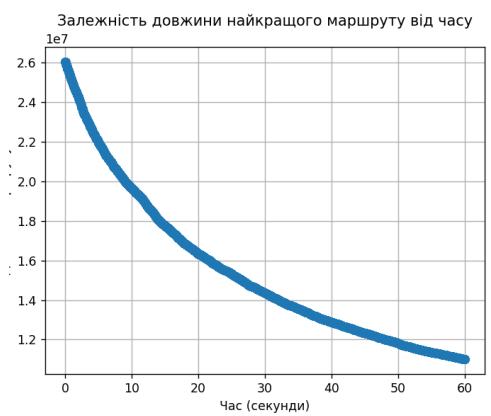
### Збільшимо кількість міст до 5000:



**Параметри:  $T = 10, T_{end} = 0, a = 0.99$**

**Start length of route: 26020896.569288358**

**Best length of route: 11077391.683877524**



**Параметри:  $T = 1000, T_{end} = 0, a = 0.99$**

**Start length of route: 26049469.94704759**

**Best length of route: 11013680.497146066**

Якщо пробувати підібрати інші параметри, то кардинально кращих результатів отримати не вдається.

### Результати роботи алгоритму імітації відпалу (рандомний початковий маршрут)

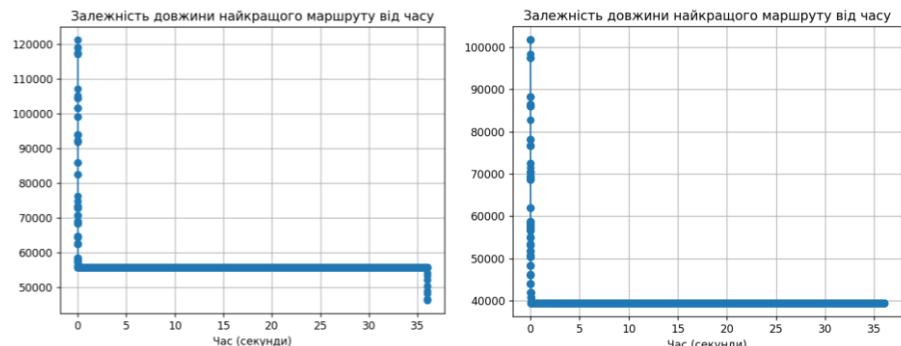
Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	40168.63	65159.59	93374.34	496989.84	11077391.68
2	45035.98	70135.19	96923.36	563111.01	10967118.54
3	47259.75	72418.74	96705.83	499353.19	11013680.49
4	43471.53	71335.49	100163.39	490705.52	10761266.13
5	43780.33	73553.80	95422.48	479893.75	10954861.25

## Результати роботи алгоритму імітації відпалу з нульовою ймовірністю прийняття гіршого варіанту:

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	46495.25	65710.59	99162.69	729904.82	10944906.51
2	58645.26	67526.17	92442.94	735494.29	15184476.51
3	46199.70	72740.59	89478.82	736861.46	15323182.42
4	47909.46	74447.03	100413.99	742866.06	15434861.51
5	52813.91	69122.01	97263.37	723278.17	14034191.23

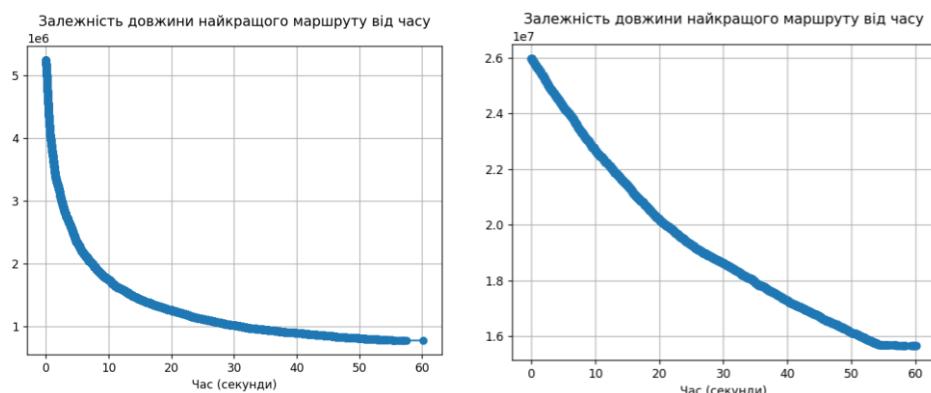
Якщо прибрати можливість вибору гіршого варіанту, то алгоритм повертає гірші маршрути. Хоча для невеликих мап результати досить близькі до звичайного методу імітації відпалу.

## Протестуємо комбінацію алгоритму імітації відпалу та 2-opt алгоритму:



Якщо алгоритм імітації відпалу знайшов хороший маршрут, то 2-opt алгоритм ніяк не покращує його.

Графіки для 1000 міст та 5000 міст:



**Результати роботи алгоритму імітації відпалу в комбінації з 2-opt алгоритмом (60% annealing + 40% 2-opt):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	45508.61	62925.49	86807.15	906443.43	16018139.86
2	46829.85	65735.19	92036.67	910177.67	15658748.65
3	46202.71	64104.96	96480.77	888948.83	15756408.72
4	39455.92	59878.92	89701.25	761583.09	15574693.63
5	37580.96	59677.02	87394.17	775510.00	12272213.29

**Результати роботи алгоритму імітації відпалу (рандомний початковий маршрут):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	40168.63	65159.59	93374.34	496989.84	11077391.68
2	45035.98	70135.19	96923.36	563111.01	10967118.54
3	47259.75	72418.74	96705.83	499353.19	11013680.49
4	43471.53	71335.49	100163.39	490705.52	10761266.13
5	43780.33	73553.80	95422.48	479893.75	10954861.25

Комбінація двох алгоритмів показує кращі результати для малих наборів міст, але для великих відстає. Тому спробуємо змінити відсоткове співвідношення часу виконання алгоритмів з 60 x 40 на 90 x 10. У підсумку, комбінація двох алгоритмів не дала сильно кращих результатів.

**Результати роботи алгоритму імітації відпалу з маршрутом, отриманим за допомогою методу найближчого сусіда:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	48373.62	63055.29	89555.84	275158.95	634876.13
2	48178.87	64068.41	84489.92	275729.44	627808.50
3	51482.85	68060.60	88699.74	277115.99	628756.83

**Результати роботи алгоритму імітації відпалу в комбінації з 2-opt алгоритмом (90% x 10%) з маршрутом, отриманим за допомогою методу найближчого сусіда:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	49216.33	63656.98	90919.90	277653.79	635496.86
2	48069.59	63508.24	84702.96	269575.67	625720.03
3	50896.40	65777.41	84307.10	283138.60	627904.01

## Генетичний алгоритм:

```
import random
import matplotlib.pyplot as plt
import csv
import time
import math

class Town:
    def __init__(self, x, y):
        self.location = [x, y]

    def print(self):
        print(f"location: [{self.location[0]}, {self.location[1]}]")

cities = []

with open("cities1.csv", newline='') as file:
    reader = csv.reader(file, delimiter=';')
    next(reader)
    for row in reader:
        x = int(row[0])
        y = int(row[1])
        cities.append((x, y))

cities20 = cities[:20]
cities50 = cities[:50]
cities100 = cities[:100]
cities1000 = cities[:1000]

def find_distance(city1, city2):
    dx = city1[0] - city2[0]
    dy = city1[1] - city2[1]
    dist = (dx ** 2 + dy ** 2) ** (1 / 2)
    return dist

def find_E(route):
    E = 0

    for i in range(len(route) - 1):
        E += find_distance(route[i], route[i + 1])

    E += find_distance(route[len(route) - 1], route[0])

    return E

def random_permutation(towns):
    route = towns[:]
    random.shuffle(route)
    return route

def nearest_neighbour(cities):
    unvisited = cities[:]
    route = []

    current_city = unvisited.pop(0)
    route.append(current_city)

    while len(unvisited) > 0:
        min_dist = float('inf')
        next_city = None
        for city in unvisited:
            dist = find_distance(route[-1], city)
            if dist < min_dist:
                min_dist = dist
                next_city = city
        route.append(next_city)
        unvisited.remove(next_city)
```

```

while unvisited:
    nearest = None
    best_dist = float('inf')

    for city in unvisited:
        dist = find_distance(current_city, city)
        if dist < best_dist:
            best_dist = dist
            nearest = city

    route.append(nearest)
    unvisited.remove(nearest)
    current_city = nearest

return route

# Генетичний алгоритм

def fitness(route):
    return find_E(route)

def build_population(cities, pop_size):
    population = []
    for _ in range(pop_size):
        population.append(random_permutation(cities))
    return population

def crossover(parent1, parent2):
    size = len(parent1)
    start, end = sorted(random.sample(range(size), 2))

    child = [None] * size
    child[start:end] = parent1[start:end]

    p2_filtered = [city for city in parent2 if city not in child]
    i = 0
    for j in range(size):
        if child[j] is None:
            child[j] = p2_filtered[i]
            i += 1

    return child

def mutation(route):
    a, b = random.sample(range(len(route)), 2)
    route[a], route[b] = route[b], route[a]
    return route

def tournament_selection(population, towns, k=3):
    selected = []
    for _ in range(len(population)):
        contenders = random.sample(population, k)
        best = min(contenders, key=lambda route: fitness(route))
        selected.append(best)
    return selected

def roulette_selection(population):
    epsilon = 1e-6 # захист від ділення на 0
    scores = [1 / (find_E(route) + epsilon) for route in population]
    total_score = sum(scores)

```

```

probabilities = [score / total_score for score in scores]

selected = []
for _ in range(len(population)):
    r = random.random()
    cumulative = 0
    for i, prob in enumerate(probabilities):
        cumulative += prob
        if r <= cumulative:
            selected.append(population[i])
            break

return selected


def genetic_tsp(towns, pop_size, cross_prob, mut_prob, generations, patience,
time_limit):
    population = build_population(towns, pop_size)
    best = min(population, key=fitness)
    best_score = fitness(best)
    start_time = time.time()
    no_improv = 0

    time_points = []
    E_points = []

    # plt.ion()
    # fig = plt.figure()

    for generation in range(generations):
        if time.time() - start_time > time_limit or no_improv >= patience:
            break

        new_population = []

        for _ in range(pop_size):
            p1, p2 = random.sample(population, 2)
            child = crossover(p1, p2) if random.random() < cross_prob else
p1[:]
            if random.random() < mut_prob:
                child = mutation(child)
            new_population.append(child)

        population = tournament_selection(new_population, towns)

        current_best = min(population, key=fitness)
        current_score = fitness(current_best)

        if current_score < best_score:
            best = current_best
            best_score = current_score
            no_improv = 0
        else:
            no_improv += 1

        time_points.append(time.time() - start_time)
        E_points.append(best_score)
        # show_route_live(best)

        print(f"Gen {generation} | Best length: {best_score:.2f}")

    # plt.ioff()
    # plt.show()

```

```

build_time_improvements_graphic(time_points, E_points)

return best

def build_time_improvements_graphic(time_points, E_points):
    plt.figure()
    plt.plot(time_points, E_points, marker='o')
    plt.title("Залежність довжини найкращого маршруту від часу")
    plt.xlabel("Час (секунди)")
    plt.ylabel("Довжина маршруту")
    plt.grid(True)
    plt.show()

def show_route_live(route, color='blue', size=20):
    plt.clf()

    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')
    plt.scatter(x_vals[0], y_vals[0], s=size * 2, c='green', zorder=6,
label='Початкове місто')
    plt.scatter(x_vals[-2], y_vals[-2], s=size * 2, c='red', zorder=6,
label='Останнє місто')

    plt.title("Поточний найкращий маршрут")
    plt.xlabel("X координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.pause(0.05)

def show_route(route, color='blue', size=20):
    x_vals = [t[0] for t in route] + [route[0][0]]
    y_vals = [t[1] for t in route] + [route[0][1]]

    plt.plot(x_vals, y_vals, color=color, linewidth=2, label='Маршрут')
    plt.scatter(x_vals, y_vals, s=size, c='black', zorder=5, label='Міста')
    plt.scatter(x_vals[0], y_vals[0], s=size * 2, c='green', zorder=6,
label='Початкове місто')
    plt.scatter(x_vals[-2], y_vals[-2], s=size * 2, c='red', zorder=6,
label='Останнє місто')

    plt.title("Візуалізація маршруту (найближчий сусід)")
    plt.xlabel("X координата")
    plt.ylabel("Y координата")
    plt.grid(True)
    plt.axis("equal")
    plt.legend()
    plt.show()

result = genetic_tsp(cities50, pop_size=50, cross_prob=0.9, mut_prob=0.2,
generations=500, patience=50, time_limit=60)
show_route(result)
print(f"Best length: {find_E(result)}")

```

## **Результати роботи генетичного алгоритму:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	37723.55	87734.92	192023.13	4032057.79	25582133.20
2	47009.45	94277.33	177803.72	4086546.63	25670815.28
3	51410.82	79280.50	198210.83	4068542.52	25699237.91
4	46204.55	104348.35	170777.83	4071017.95	25741238.27
5	38373.23	88984.62	193627.76	4019888.66	25444711.56

З обраними параметрами алгоритм досить добре знаходить маршрути для мап невеликих розмірів, але для великих результати неоптимальні. Треба пробувати різні стратегії та параметри для того, щоб отримати кращі результати.

## **Результати роботи генетичного алгоритму з початковим маршрутом, отриманим за допомогою методу найближчого сусіда:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	39345.85	69472.65	102094.95	295677.03	636811.65
2	43690.17	75872.28	96586.79	291282.21	629902.49
3	38054.96	69949.29	95804.07	292995.68	629127.83
4	37899.23	72216.36	96521.28	289590.19	629126.00
5	38054.96	66506.06	93918.36	298727.56	623537.76

Генетичний алгоритм добре покращує маршрут для невеликої кількості міст, але для великої кількості міст значних покращень не спричиняє.

**Висновок:** під час виконання даної лабораторної роботи я дізнався про такі евристичні методи вирішення задачі комівояжера, як: метод найближчого сусіда, 2-opt алгоритм, 3-opt алгоритм, алгоритм імітації відпалу. Реалізував та порівняв між собою: власний метод грубої сили, метод найближчого сусіда, 2-opt алгоритм з різними стратегіями вибору індексів елементів, алгоритм імітації відпалу та генетичний алгоритм.

Після порівняння я отримав наступні результати:

#### Результати роботи брутфорсу:

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	59954.17	185149.45	407913.95	4912663.51	25544130.83
2	60844.24	191062.71	413159.51	4898294.55	25485638.49
3	59751.63	172254.54	416151.37	4912425.60	25515557.96
4	61956.42	193131.47	419249.30	4874375.32	25543596.54
5	60778.69	183563.08	412157.77	4914741.45	25523287.32

#### Результати роботи методу найближчого сусіда:

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	50800.65	69532.21	103891.36	295677.03	636811.65
2	50290.80	75872.28	96889.94	291282.21	629902.49
3	53144.77	70008.85	95804.07	292995.68	629127.83
4	51006.21	72800.06	96521.28	289590.19	629126.00
5	38054.96	69828.57	93920.09	298727.56	623537.76

#### Результати з випадковим маршрутом (повний перебір):

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	38373.23	59969.50	91988.54	4022285.75	24795356.86
2	47093.52	66592.81	85177.43	3362800.35	25155016.78
3	45569.00	66427.97	88089.57	2993228.83	24548485.76

4	38818.51	63868.93	88002.49	3102152.78	24255816.86
5	44782.41	67450.79	87100.24	4072082.15	25104270.69

**Результати роботи 2-opt алгоритму з рандомним вибором індексів (випадковий початковий маршрут):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	46640.47	67588.08	110965.19	653765.78	14070330.29
2	48683.32	65677.29	107499.72	637540.61	14296273.45
3	51422.59	70826.91	95786.27	655974.51	14470623.74
4	50719.25	73029.07	104225.40	665402.29	14346914.44
5	51822.67	71369.93	92893.72	664503.03	14094085.66

**Результати роботи 2-opt алгоритму з маршрутом утвореним за допомогою методу найближчого сусіда:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	48178.87	67425.83	91173.51	295166.76	636811.65
2	48133.83	65256.17	88114.98	289682.62	629859.62
3	51852.51	67689.76	86092.52	292565.16	628894.10
4	47314.41	66947.06	90349.69	289565.14	629126.00
5	38054.96	66613.78	88439.41	298084.06	623492.08

**Результати з методом найближчого сусіда (повний перебір):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	48373.62	63685.85	91935.47	294529.85	634316.51
2	47680.82	65230.14	86207.16	289805.94	629887.07
3	47919.39	67688.35	83550.81	292440.66	628456.40
4	44385.51	63257.42	87735.04	285667.81	629108.73
5	38054.96	66812.94	85357.17	297157.64	620321.02

**Результати роботи алгоритму імітації відпалу в комбінації з 2-opt алгоритмом (60% annealing + 40% 2-opt):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	45508.61	62925.49	86807.15	906443.43	16018139.86
2	46829.85	65735.19	92036.67	910177.67	15658748.65
3	46202.71	64104.96	96480.77	888948.83	15756408.72
4	39455.92	59878.92	89701.25	761583.09	15574693.63
5	37580.96	59677.02	87394.17	775510.00	12272213.29

**Результати роботи алгоритму імітації відпалу (рандомний початковий маршрут):**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	40168.63	65159.59	93374.34	496989.84	11077391.68
2	45035.98	70135.19	96923.36	563111.01	10967118.54
3	47259.75	72418.74	96705.83	499353.19	11013680.49
4	43471.53	71335.49	100163.39	490705.52	10761266.13
5	43780.33	73553.80	95422.48	479893.75	10954861.25

**Результати роботи алгоритму імітації відпалу з нульовою ймовірністю прийняття гіршого варіанту:**

Спроба №	Розмір мапи та довжини маршрутів				
	20	50	100	1000	5000
1	46495.25	65710.59	99162.69	729904.82	10944906.51
2	58645.26	67526.17	92442.94	735494.29	15184476.51
3	46199.70	72740.59	89478.82	736861.46	15323182.42
4	47909.46	74447.03	100413.99	742866.06	15434861.51
5	52813.91	69122.01	97263.37	723278.17	14034191.23

У процесі того, як я отримував ці результати я проводив їх порівняння. У підсумку хочу поділитися враженям від роботи 2-opt алгоритму. Я

реалізував анімований вивід графіку та побачив, як 2-opt алгоритм приходить від дуже запутаного рандомно-згенерованого маршруту до досить оптимального. Це вишлядало дуже ефектно, і я не очікував, що він буде настільки добре справлятися.

Метод найближчого сусіда особливо добре працює у випадках, коли міста розташовані по якомусь контуру. А ще він знаходить найкращі маршрути у випадку дуже великих мап. Жоден з алгоритмів не зміг прийти від рандомного початкового маршруту до такого, щоб його довжина приблизно дорівнювала довжині того, який ми отримуємо за допомогою методу найближчого сусіда. Це теж досить неочікувано. Можливо це пов'язано з жорсткими часовими обмеженнями (60 секунд), через які алгоритм не встигає розплутати початковий маршрут.

Алгоритм імітації відпалу дуже ефективно працює на початку, доки температури високі. Його графік покращень маршруту за часом має вигляд гілки параболи. Коли температури знижуються значних покращень майже не відбувається. Комбінація його з 2-opt алгоритмом значних переваг не дали. Також працює краще, якщо ймовірність прийняття гіршого маршруту не дорівнює нулю.

Генетичний алгоритм також може знайти хороші маршрути, але потрібно ретельно підбирати параметри.