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

Звіт
про виконання лабораторної роботи №3
з дисципліни «Методи оптимізції»
“Дослідження еволюційних алгоритмів для задач оптимізації”

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

Київ – 2025

Варіант 20

Мета роботи: Ознайомитись і дослідити роботу еволюційних алгоритмів для вирішення задач оптимізації та NP-повних задач, порівняти застосування точних та неточних алгоритмів пошуку. Набути навичок реалізації одного з еволюційних алгоритмів (генетичного алгоритму). Провести експерименти, дослідити та порівняти різні варіації та параметри роботи еволюційного алгоритму.

Завдання:

Задача 3

Задача розміщення систем ППО. Маємо набір міст кількістю N, що характеризуються параметрами: 2D-координати $x, y \in [-10^3; 10^3]$, населення $n \in [0, 10^3]$; та набір систем ППО кількістю $D, D \in [8; 20]$, що характеризуються радіусом охоплення $r \in [10, 100]$. Необхідно знайти оптимальну розстановку систем ППО – 2D-координати $(d_x; d_y)$ для кожної наявної системи ППО.

Рівень 1. Оптимальність розстановки: покриття максимальної кількості міст хоча б однією системою ППО без врахування населення:

$$F = \sum_{i=1}^{N} (\exists (d_x; d_y) \in D | (x_i, x_j) is\_in\ d_{radius}).$$

Рівень 2. Оптимальність розстановки: покриття максимальної кількості населення хоча б однією системою ППО, за умови, що на одну систему ППО повинно припадати мінімум 3 міста:

$$F = \sum_{i=1}^{N} (n_i, \exists (d_x; d_y) \in D | \{(x_i, x_j) is\_in\ d_{radius}\} \geq 3).$$

Хід роботи:

1. **Підготувати вхідні дані.**

1.1. Обрати задачу оптимізації з заданих у варіантах. Задачі подані в порядку зростання складності. Кожна містить 2 рівня складності – одноцільова та багатоцільова оптимізація.

Задача 3

1.2. Згенерувати рандомний набір предметів розміром $N = 10^5$ для своєї задачі зі значеннями параметрів у заданому діапазоні (для деяких задач потрібно декілька окремих наборів предметів).

1.3. Зберегти згенерований набір у файл та додати можливість завантаження з файлу.

1.4. Всі наступні експерименти проводити на одному константному наборі предметів (на всьому наборі чи на його частині).

Генерація початкового набору даних:

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

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

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


# Генерація набору даних
# Діапазон координат
X_RANGE = (-1000, 1000)
Y_RANGE = (-1000, 1000)


# Кількість кластерів кожного типу
clusters = {
    "big_dense": 2,
    "small_dense": 3,
    "big_sparse": 3,
    "small_sparse": 2,
}
```

```python
towns = []


for cluster_type, count in clusters.items():

    for _ in range(count):

        # визначаємо точку на мапі, яка стане центром кластера
        center_x = random.uniform(*X_RANGE)
        center_y = random.uniform(*Y_RANGE)

        if "big" in cluster_type:
            num_towns = random.randint(200, 300)
        else:
            num_towns = random.randint(30, 70)

        for _ in range(num_towns):

            if "dense" in cluster_type:
                spread = random.randint(20, 50)
            else:
                spread = random.randint(250, 300)


            x = random.gauss(center_x, spread)
            y = random.gauss(center_y, spread)


            if "dense" in cluster_type:
                population = random.randint(500, 1000)
            else:
                population = random.randint(10, 300)

            towns.append((x, y, population))


print(f"amount of towns: {len(towns)}")



# візуалізація
x_vals = [t[0] for t in towns]
y_vals = [t[1] for t in towns]
population_sizes = [t[2] for t in towns]


plt.figure(figsize=(10, 10))
plt.scatter(x_vals, y_vals, s=[p / 10 for p in population_sizes], alpha=0.6)
plt.title("Мапа")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.show()


#Запис до файлу
with open("towns1.csv", mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["x", "y", "population"])
    for town in towns:
        writer.writerow(town)
```

Для того, щоб уникнути прямолінійних залежностей між параметрами та зробити розподіл нерівномірним, а більш реалістичним, генеруємо дані наступним чином:

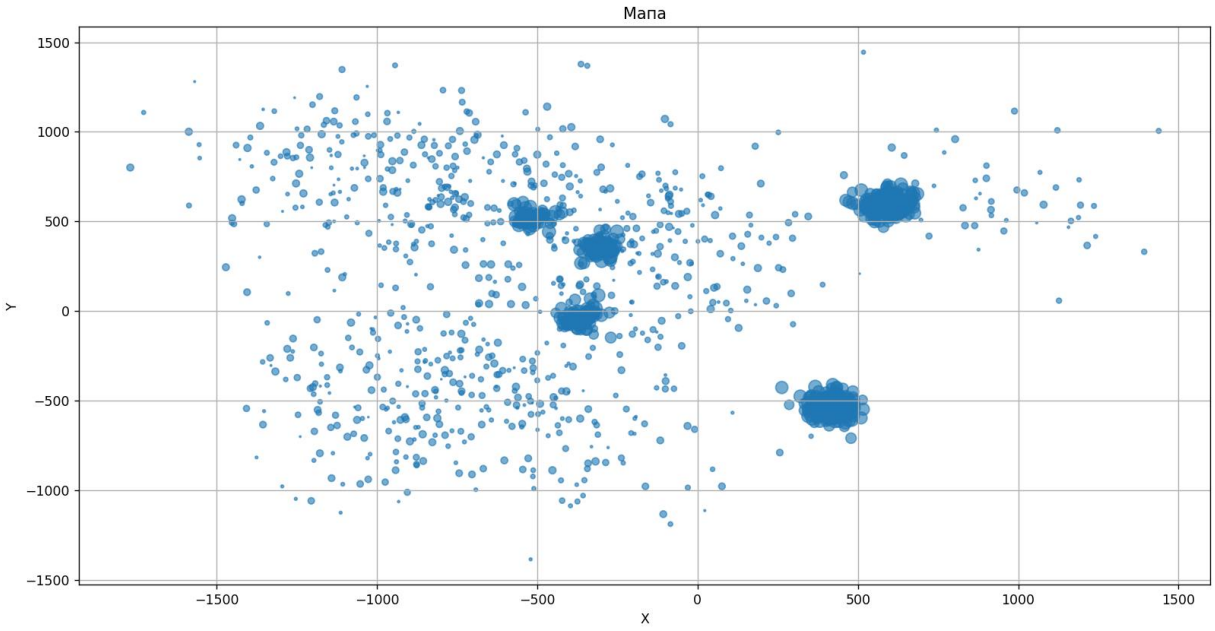
Введемо категорії (кластери) міст:

- Великі та щільнозаселені
- Малі та щільнозаселені
- Великі та малонаселені
- Малі та малонаселені

Та визначимо кількість кожного з кластерів.

Генерація буде відбуватися шляхом вибору центру кластера та розміщенням навколо центру міст. Наприклад, якщо кластер великий та щільнозаселений, то в такому кластері буде багато міст, а населення самих міст буде великим і відстані між містами будуть невеликі.

Після того, як виконали генерацію та зберегли набір до файлу, отримуємо таку мапу:



Бачимо яскравовиражені райони з великою щільністю населення (2 великі та 3 малі) і також райони з невеликою щільністю населення.

Тепер **згенеруємо набір систем ппо:**

```python
import random
import csv

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

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


amount = random.randint(8,20)

ADSs = []

for i in range(amount):
    x = 0
    y = 0
    radius = random.uniform(10, 100)
    print(f"system {i} radius: {radius}")
    ADSs.append((x, y, radius))

# Зберігаємо набір систем у файл
with open("ADSs.csv", mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["x", "y", "radius"])
    for ads in ADSs:
        writer.writerow(ads)
```

У результаті отримуємо такий набір:

1	x,y,radius
2	0,0,76.43773302549874
3	0,0,49.5848134839935
4	0,0,55.4800532978875
5	0,0,90.38704041969221
6	0,0,38.96040928227099
7	0,0,61.59135532366424
8	0,0,26.618044041123333
9	0,0,26.426765565041325
10	0,0,19.950285986269662
11	0,0,78.37145649786882
12	0,0,91.15817921302222
13	0,0,61.41911709531563
14	0,0,66.71782088188357
15	0,0,58.33503967268127
16	0,0,53.162642220679146
17	0,0,27.377211781415735
18	0,0,95.69557573995515
19	0,0,67.51516439303238

18 одиниць із різним радіусом дії.

Тепер необхідно знайти оптимальну розстановку систем ППО – 2D-координати $(d_x; d_y)$ для кожної наявної системи ППО.

Рівень 1. Оптимальність розстановки: покриття максимальної кількості міст хоча б однією системою ППО без врахування населення:

$$F = \sum_{i=1}^{N} (\exists (d_x; d_y) \in D | (x_i, x_j) is\_in\ d_{radius}).$$

2. **Підготувати цільову функцію $F(x)$ для обраної задачі, де x – набір вхідних параметрів.**

   Вхідними параметрами є локація та радіус дії системи ППО, а на виході має бути кількість покритих міст без врахування населення.

```
def optimality_lvl_1(adss, towns):
    covered = set()

    for i, ads in enumerate(adss):
        for j, town in enumerate(towns):
            dx = ads.location[0] - town.location[0]
            dy = ads.location[1] - town.location[1]
            distance = (dx**2 + dy**2)**(1/2)
            if distance <= ads.radius:
                covered.add(j)

    return len(covered)
```

   Оскільки місто може бути покрито одразу декількома системами ППО, використовуємо структуру Set(), тоді кожне місто буде враховано лише один раз.

3. **Реалізувати алгоритм точного пошуку (перебір).**

   Жадібний брутфорс:

   Ідея полягає в тому, що ми будемо шукати найкращу локацію для кожної системи ППО по черзі.

   Алгоритм:

   Для кожної системи ППО:

   - Рандомно пробуємо багато точок (x, y)
   - Для кожної точки обчислюємо, скільки нових міст покривається (які ще не накриті)

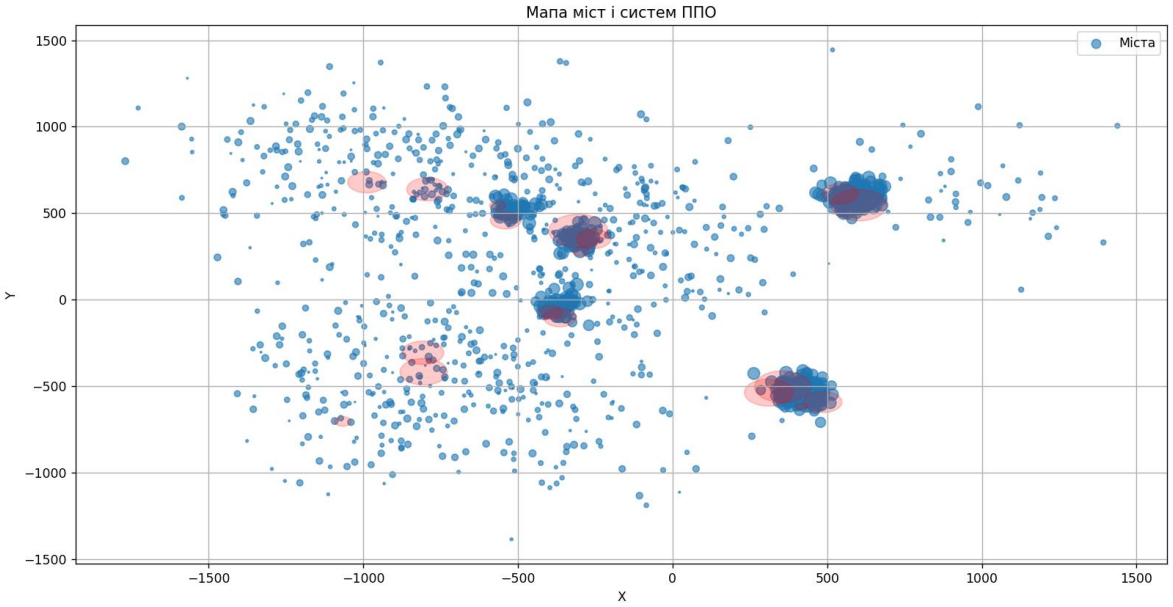
- Вибираємо точку у якій покривається найбільша кількість міст
- Викреслюємо міста, які вже накриті

Лістинг:

```
# Брутфорс

def evaluate_ads_coverage(ads, towns, already_covered):
    new_covered = set()

    for i, town in enumerate(towns):
        if i in already_covered:
            continue  # це місто вже покрите іншою системою

        dx = ads.location[0] - town.location[0]
        dy = ads.location[1] - town.location[1]
        distance = (dx**2 + dy**2)**0.5

        if distance <= ads.radius:
            new_covered.add(i)

    return new_covered  # повертаємо набір індексів міст, які покриває ця
система


def brutforce(adss, towns):
    already_covered = set()

    for ads in adss:
        best_location = None
        best_covered = set()

        for _ in range(100):

            x = random.uniform(-1750, 1500)
            y = random.uniform(-1500, 1500)

            ads.location[0] = x
            ads.location[1] = y

            new_covered = evaluate_ads_coverage(ads, towns, already_covered)

            if(len(new_covered) > len(best_covered)):
                best_covered = new_covered
                best_location = (x, y)

        if(best_location):
            ads.location[0] = best_location[0]
            ads.location[1] = best_location[1]
            already_covered.update(best_covered)

    return adss
```

Запустимо алгоритм та оцінимо результати.

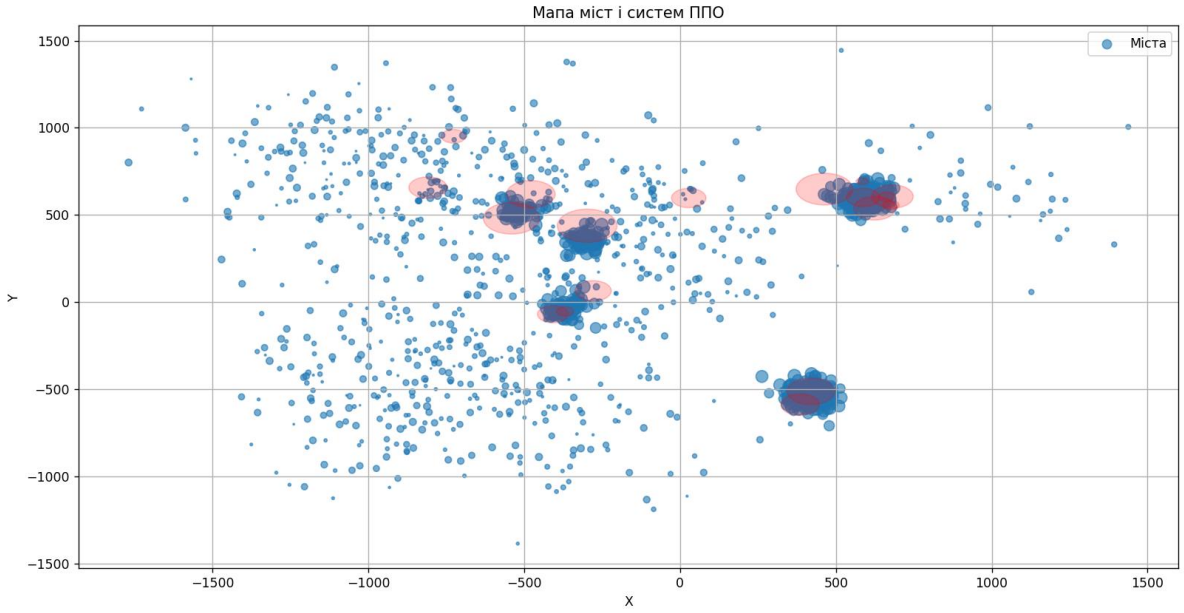
Якщо пробувати 100 рандомних точок:



Покрили:

```
D:\PythonProjects\MO_Lab_3\BrutForce\.venv\Scripts\pytho
Загальна кількість покритих міст: 647, КОЦФ: 1800
```

Спроба 2:




```
D:\PythonProjects\MO_Lab_3\BrutForce\.venv\Scripts\pytho
Загальна кількість покритих міст: 674, КОЦФ: 1800
```

Для зручності зберемо результати у таблицю:

Спроба	Кількість пробних точок	
	100 (КОЦФ 1800)	1000 (КОЦФ 18000)
1	647	761
2	674	735
3	630	739
4	578	763
5	708	759

Результати роботи цього алгоритму є дуже хорошими, оскільки він розставив системи у райони, де міста розташовані найбільш щільно.

Генетичний алгоритм

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

# вхідні дані
call_count = 0  # глобальний лічильник

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

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


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

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


adss = []
towns = []

with open("towns1.csv", mode="r") as file:
    reader = csv.reader(file)
    next(reader)  # Пропускаємо заголовок

    for row in reader:
        x = float(row[0])
        y = float(row[1])
        population = int(row[2])

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

with open("ADSs.csv", mode="r") as file:
    reader = csv.reader(file)
    next(reader)

    for row in reader:
        x = float(row[0])
        y = float(row[1])
        radius = float(row[2])

        ads = ADS(x, y, radius)
        adss.append(ads)


# Цільова функція

def optimality_lvl_1(adss, towns):
    global call_count
```

```
    call_count += 1

    covered = set()
    for i, ads in enumerate(adss):
        for j, town in enumerate(towns):
            dx = ads.location[0] - town.location[0]
            dy = ads.location[1] - town.location[1]
            distance = (dx ** 2 + dy ** 2) ** (1 / 2)
            if distance <= ads.radius:
                covered.add(j)

    return len(covered)


# цільова функція для одної системи

def optimality_single(ads, towns):
    global call_count
    call_count += 1

    covered = set()
    for i, town in enumerate(towns):
        dx = ads.location[0] - town.location[0]
        dy = ads.location[1] - town.location[1]
        distance = (dx ** 2 + dy ** 2) ** (1 / 2)
        if distance <= ads.radius:
            covered.add(i)

    return len(covered)


# Генетичний алгоритм

def build_population(template_adss, population_size):
    population = []

    for _ in range(population_size):
        individual = []

        for ads in template_adss:
            x = random.uniform(-1500, 1500)
            y = random.uniform(-1500, 1500)
            radius = ads.radius

            new_ads = ADS(x, y, radius)
            individual.append(new_ads)

        population.append(individual)

    return population


# Проходимось по спискам та для кожної ппо схрещуємо координати, і на виході
отримуємо хромосому нащадка

def crossover(ancestor1, ancestor2):
    descendant = []

    for i in range(len(ancestor1)):
        # # Перша стратегія схрещування
        # x = ancestor1[i].location[0]
        # y = ancestor2[i].location[1]

        # # Друга стратегія (усереднення)
```

```
        # x = (ancestor1[i].location[0] + ancestor2[i].location[0]) / 2
        # y = (ancestor1[i].location[1] + ancestor2[i].location[1]) / 2

        # Можна також комбінувати першу і другу стратегії
        if random.randint(0, 1):
            x = ancestor1[i].location[0]
            y = ancestor2[i].location[1]
        else:
            x = (ancestor1[i].location[0] + ancestor2[i].location[0]) / 2
            y = (ancestor1[i].location[1] + ancestor2[i].location[1]) / 2

        radius = ancestor2[i].radius

        new_ads = ADS(x, y, radius)
        descendant.append(new_ads)

    return descendant


# Функції для ін/аутбрідінгу

def count_chromosome_distance(set1, set2):
    total = 0

    for i in range(len(set1)):
        dx = set1[i].location[0] - set2[i].location[0]
        dy = set1[i].location[1] - set2[i].location[1]
        dist = (dx ** 2 + dy ** 2) ** (1 / 2)
        total += dist
    return total


def find_same(population, set1):
    i2 = 0
    minimum = float("inf")

    for i in range(len(population)):
        dif = count_chromosome_distance(population[i], set1)
        if dif < minimum:
            minimum = dif
            i2 = i

    return i2


def find_different(population, set1):
    i2 = 0
    maximum = float("-inf")

    for i in range(len(population)):
        dif = count_chromosome_distance(population[i], set1)
        if dif > maximum:
            maximum = dif
            i2 = i

    return i2


# Мутація

def mutation(descendant):
    i = random.randint(0, len(descendant) - 1)

    # стратегія 1 - рандомно обрати одну з координат
```

```
    if random.randint(0, 1):
        descendant[i].location[0] = random.uniform(-1500, 1500)
    else:
        descendant[i].location[1] = random.uniform(-1500, 1500)

    # # стратегія 2 - зсунути координату
    # delta = random.uniform(-100, 100)
    #
    # if random.randint(0, 1):
    #     descendant[i].location[0] += delta
    # else:
    #     descendant[i].location[1] += delta

    return descendant


# Схрещування

def crossbreeding(population, crossbreeding_probability,
mutation_probability, amount_of_descendants):
    population_size = len(population)
    new_generation = []

    for _ in range(amount_of_descendants):

        if random.randint(0, 100) / 100 < crossbreeding_probability:

            # # Панміксія (обидва батьки обираються випадково)
            # i1 = random.randint(0, population_size - 1)
            # i2 = random.randint(0, population_size - 1)
            #
            # ancestor1 = population[i1]
            # ancestor2 = population[i2]

            # # Інбрідінг (перший - випадково, другий - найбільш схожим
(схожість - схожість координат))
            # i1 = random.randint(0, population_size - 1)
            # ancestor1 = population[i1]
            #
            # i2 = find_same(population, ancestor1)
            # ancestor2 = population[i2]
            #
            # Аутбрідінг (перший - випадково, другий - найменш схожим)
            i1 = random.randint(0, population_size - 1)
            ancestor1 = population[i1]

            i2 = find_different(population, ancestor1)
            ancestor2 = population[i2]

            descendant = crossover(ancestor1, ancestor2)

            # Мутація

            if random.randint(0, 100) / 100 < mutation_probability:
                descendant = mutation(descendant)

            new_generation.append(descendant)

    return new_generation


# Селекція

def top_selection(population, pop_size, new_generation, towns):
```

```
    elite = []
    children = []
    parent_scores = []

    elite_size = int(pop_size * 0.2)
    children_size = pop_size - elite_size

    for individual in population:
        score = optimality_lvl_1(individual, towns)
        parent_scores.append((individual, score))

    parent_scores.sort(key=lambda x: x[1], reverse=True)

    elite = [pair[0] for pair in parent_scores[:elite_size]]

    new_children_count = min(children_size, len(new_generation))
    for _ in range(new_children_count):
        index = random.randint(0, len(new_generation) - 1)
        children.append(new_generation.pop(index))

    while len(children) < children_size:
        backup_index = elite_size + (len(children) % (pop_size - elite_size))
        children.append(parent_scores[backup_index][0])

    return elite + children



def tournament_selection(population, pop_size, new_generation, towns):

    new_population = []
    population += new_generation
    num_of_comparison = len(population) // 2

    for i in range(num_of_comparison):
        i1 = random.randint(0, len(population) - 1)
        i2 = random.randint(0, len(population) - 1)

        value1 = optimality_lvl_1(population[i1], towns)
        value2 = optimality_lvl_1(population[i2], towns)

        if value1 > value2:
            winner = population[i1]
        else:
            winner = population[i2]

        new_population.append(winner)

    return new_population

def roulette_selection(population, pop_size, new_generation, towns):

    population += new_generation

    scores = []
    for chromosome in population:
        score = optimality_lvl_1(chromosome, towns)
        scores.append(score)

    total_score = 0
    for score in scores:
        total_score += score

    probabilities = []
```

```
    for score in scores:
        prob = score / total_score
        probabilities.append(prob)


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

    return selected



def selection(population, pop_size, new_generation, towns):
    # Топ найкращих
    return top_selection(population, pop_size, new_generation, towns)

    # # Турнірна селекція
    # return tournament_selection(population, pop_size, new_generation,
towns)

    # # Метод рулетки
    # return roulette_selection(population, pop_size, new_generation, towns)

# Перевірка досягнення результату

def find_best_chromosome(population, towns):
    best_score = 0
    best_chromosome = population[0]

    for individual in population:
        score = optimality_lvl_1(individual, towns)
        if score > best_score:
            best_score = score
            best_chromosome = individual

    return best_chromosome, best_score



# Генетичний алгоритм


def genetic(adss, towns, pop_size, cross_prob, mut_prob,
amount_of_generations, target_score, patience, time_limit_sec):
    # 1 Генерція початкової популяції
    population = build_population(adss, pop_size)
    amount_of_descendants = pop_size

    best_score = 0
    best_chromosome = None
    generations_without_improvement = 0

    start_time = time.time()

    for generation in range(amount_of_generations):
        # 2 Схрещування / мутація
        descendants = crossbreeding(population, cross_prob, mut_prob,
```

```
amount_of_descendants)

        # 3 Селекція  і формування нового покоління
        population = selection(population, pop_size, descendants, towns)

        # 5 Перевірка досягнення результату
        # Пошук найкразого представника
        current_best, current_score = find_best_chromosome(population, towns)

        # Умова 1: Досягнуто заданого значення
        if current_score >= target_score:
            print(f"Знайдено ціль на {generation + 1}-му поколінні")
            best_chromosome = current_best
            best_score = current_score
            break

        # Умова 2 — немає покращень K поколінь
        if current_score > best_score:
            best_score = current_score
            best_chromosome = current_best
            generations_without_improvement = 0
        else:
            generations_without_improvement += 1

        if generations_without_improvement >= patience:
            print(f"Зупинка через відсутність покращень {patience} поколінь")
            break

        # Умова 3 — досягнення максимальної кількості поколінь
        if generation == amount_of_generations - 1:
            print("Досягнуто максимальної кількості поколінь")

        # Умова 4 — перевищено час
        if time.time() - start_time > time_limit_sec:
            print("Перевищено ліміт часу")
            break

        # log
        print(f"generation {generation} best_score: {best_score}")

    return best_chromosome, best_score

# Тестування



# population = build_population(adss, 10)
# new_generation = crossbreeding(population, 0.9, 0.1, 100)
# population = selection(population, 10, new_generation, towns)
#
#
# for i in range(len(descendants)):
#     print(f"n: {i}")
#     chromosome = descendants[i]
#     for i in range(18):
#        chromosome[i].print()

adss_final, result = genetic(adss, towns, 20, 0.9, 0.2, 10000, 1000, 500,
600)
print(f"Загальна кількість покритих міст: {result}, КОЦФ: {call_count}")

# візуалізація

x_towns = [t.location[0] for t in towns]
```

```
y_towns = [t.location[1] for t in towns]
population_sizes = [t.population for t in towns]

fig, ax = plt.subplots(figsize=(10, 10))
# вивід міст
ax.scatter(x_towns, y_towns, s=[p / 10 for p in population_sizes], alpha=0.6,
label="Міста")

# вивід ППО
for ads in adss_final:
    circle = patches.Circle((ads.location[0], ads.location[1]), ads.radius,
color="red", alpha=0.2, fill=True)
    ax.add_patch(circle)


ax.set_title("Мапа міст і систем ППО")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.grid(True)
ax.legend()
# plt.axis('equal')  # Щоб кола були кругами, а не еліпсами
plt.show()
```

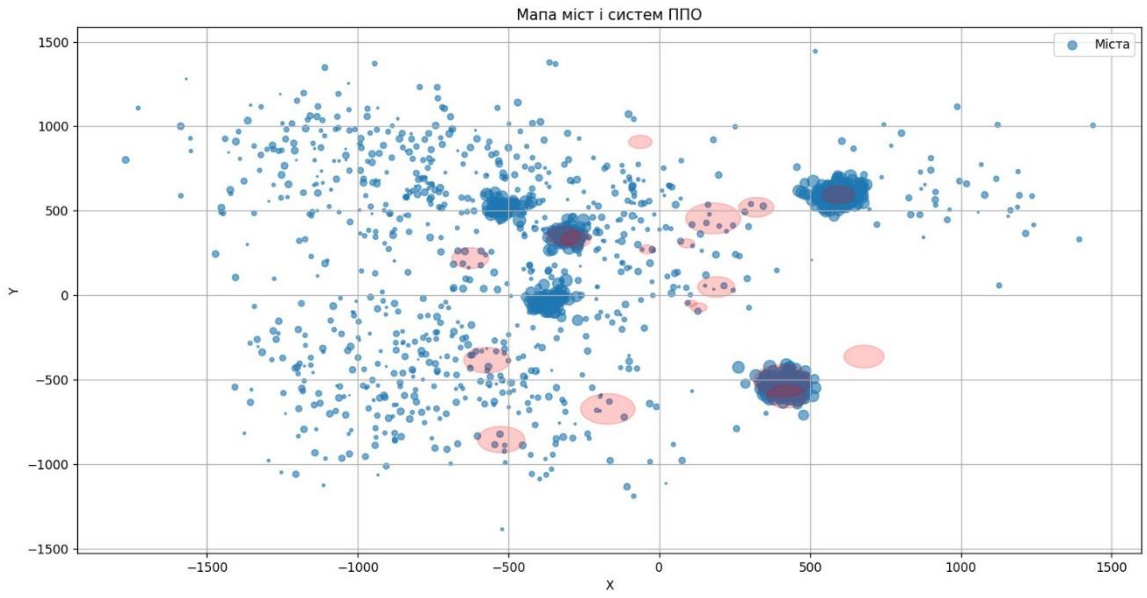
У коді реалізовано декілька стратегій схрещувань, мутацій та селекції.

Тестування:

Спочатку протестуємо алгоритм з такими параметрами:

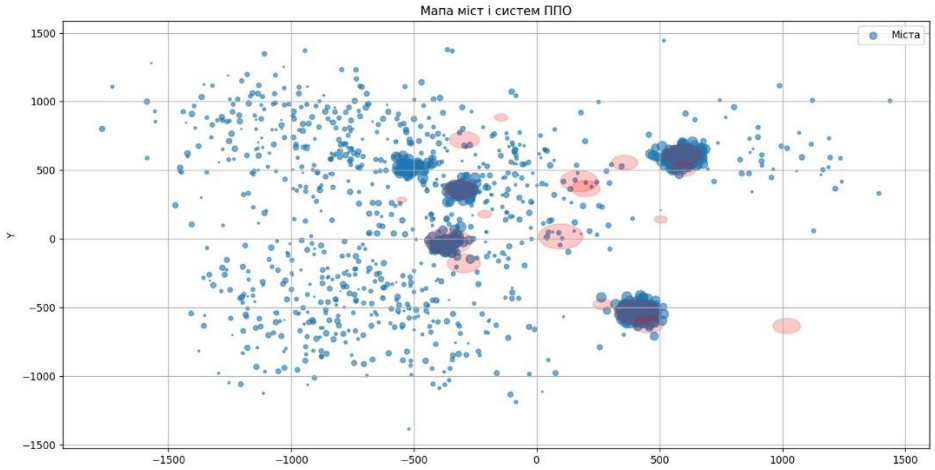
- Схрещування: аутбрідінг, комбінація двох стратегій (усереднення та комбінування координат), ймовірність: 0.9
- Мутація: 2 – зсув на невелику відстань, ймовірність: 0.1
- Селекція: Топ найкращих
- Розмір популяції: 100

Результат:



Накрито **588** міст. Цей результат є гіршим за Брутфорс, спробуємо ще раз з цими ж параметрами:

Результат:



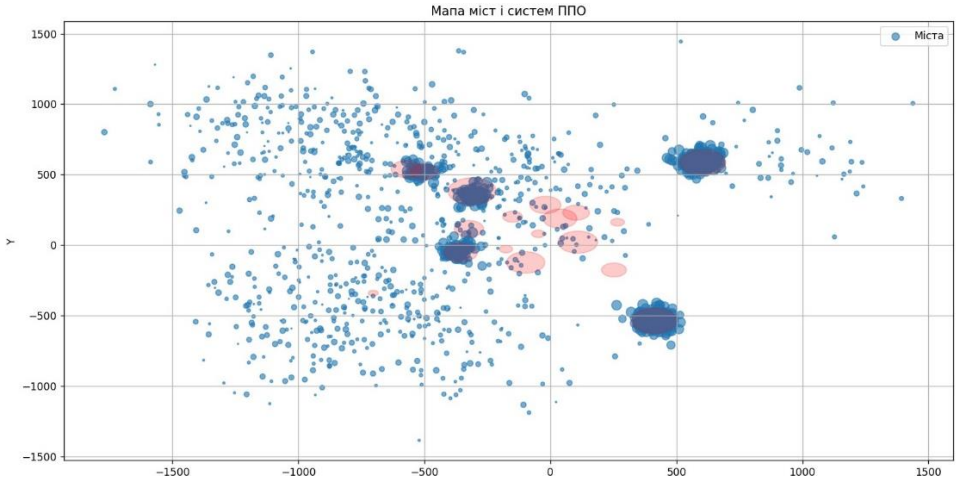
Накрито 681 місто. Вже краще, але гірше за брутфорс. Спробуємо інші параметри.

Спробуємо використовувати мутацію 1 (одну з координат генерує рандомно), а також зменшимо розмір популяції, тоді ми пройдемо більше поколінь.

Параметри:

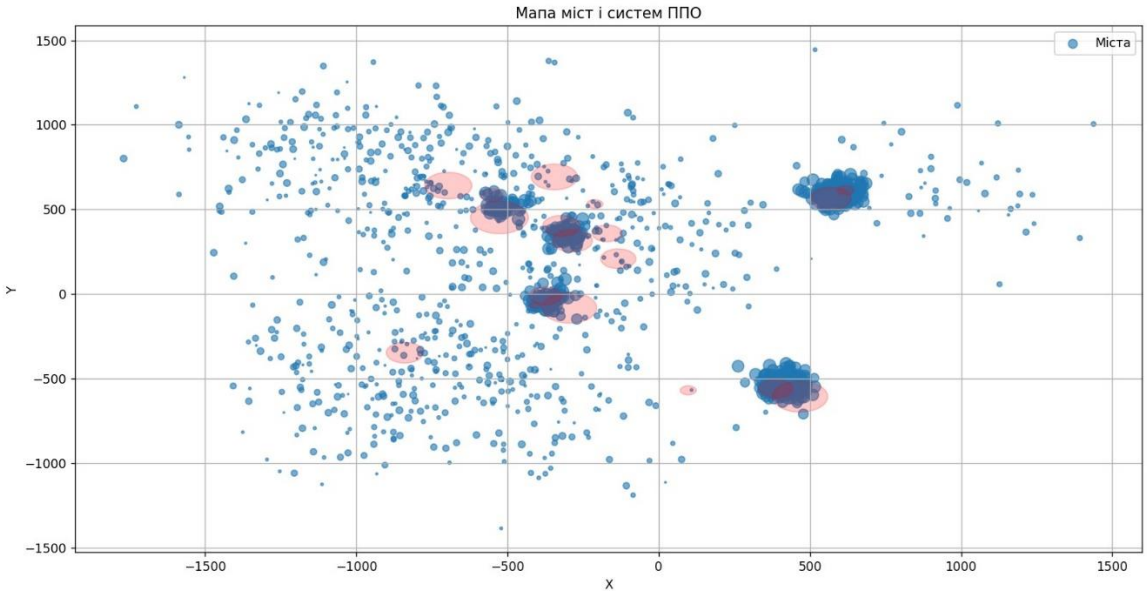
- Схрещування: аутбрідінг, комбінація двох стратегій (усереднення та комбінування координат), ймовірність: 0.9
- Мутація: 1 – одну з координат генеруємо рандомно, ймовірність: 0.1
- Селекція: Топ найкращих
- Розмір популяції: 20

Результат:

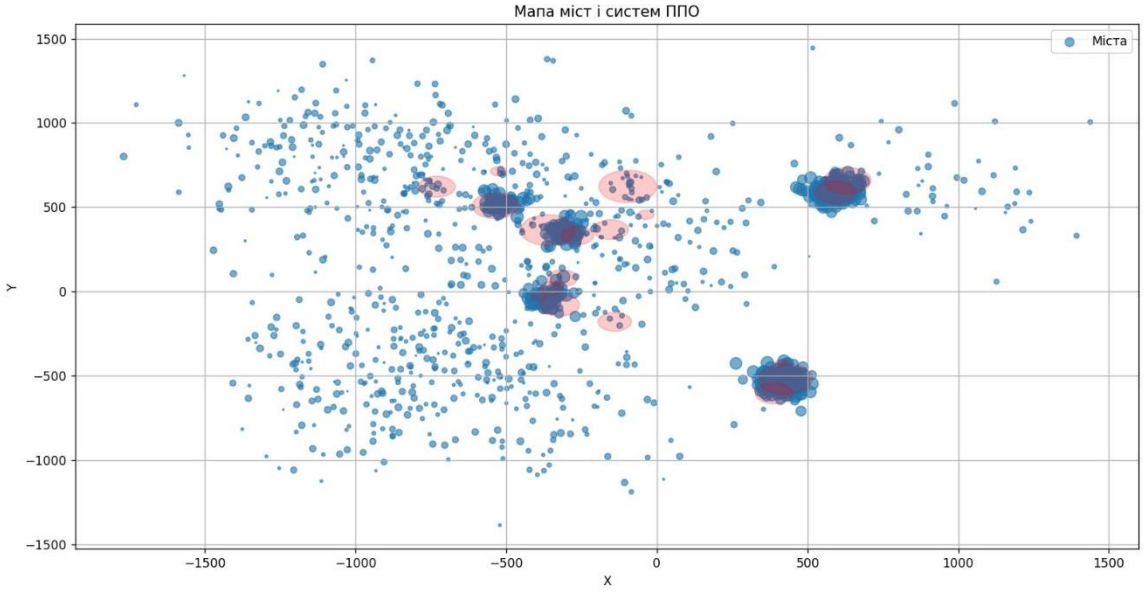


Накрито 739 міст. Це вже на рівні з брутфорсом, але видно, що багато систем нічого не накривають, але інші накривають дуже добре.

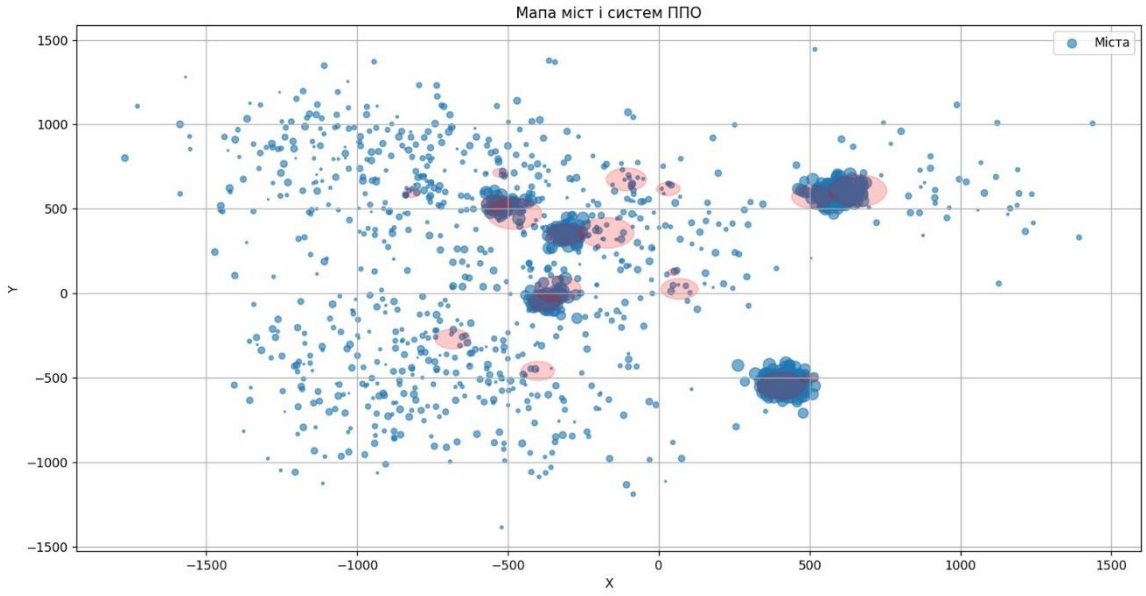
Змінимо стратегію селекції на турнірну:



Накрито 643 міста.



Накрито 752 міста.



Накрито 754 міста.

Така селекція видає результати на рівні з брутфорсом. І тут варто сказати, що результат, який дав брутфорс є досить оптимальним. Оскільки на мапі є чітко окреслені райони з великою щільністю міст, і найкращі результати цільової функції будуть саме в них. Жадібний брутфорс влучає в такі точки та залишає систему там, тому він дає хороші результати.

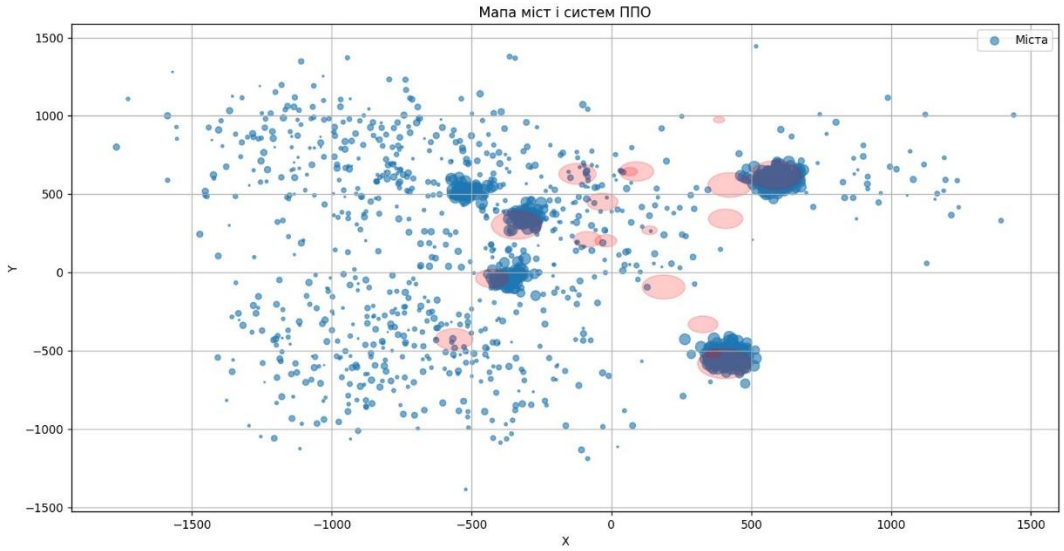
Спробуємо використати стратегію селекції “метод рулетки”. Але, на мою думку, тут доцільно застосовувати стратегію елітизму, оскільки найкращі точки варто зберігати.

Тестування методу рулетки:

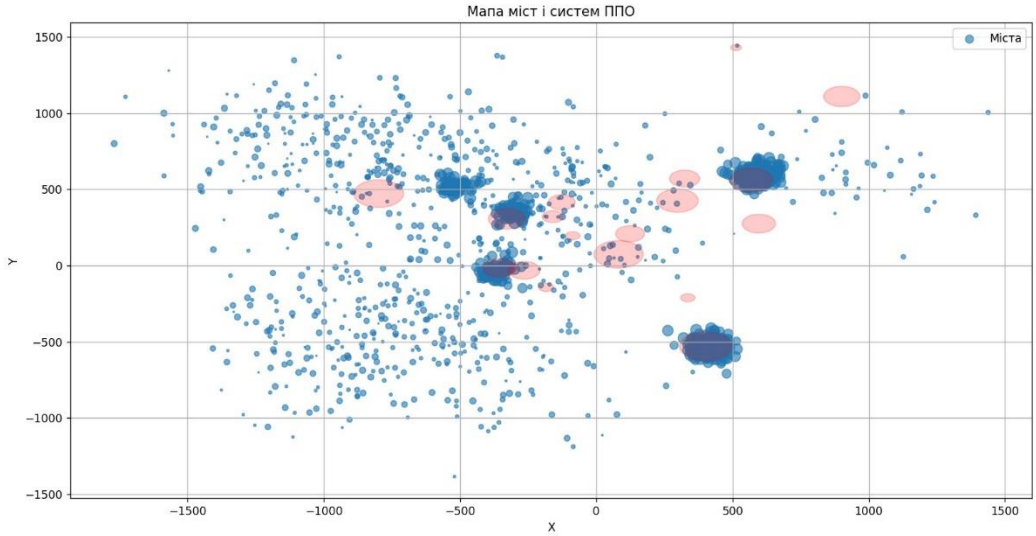
Параметри:

- Схрещування: аутбрідінг, комбінація двох стратегій (усереднення та комбінування координат), ймовірність: 0.9
- Мутація: 1 – одну з координат генеруємо рандомно, ймовірність: 0.2
- Селекція: Топ найкращих

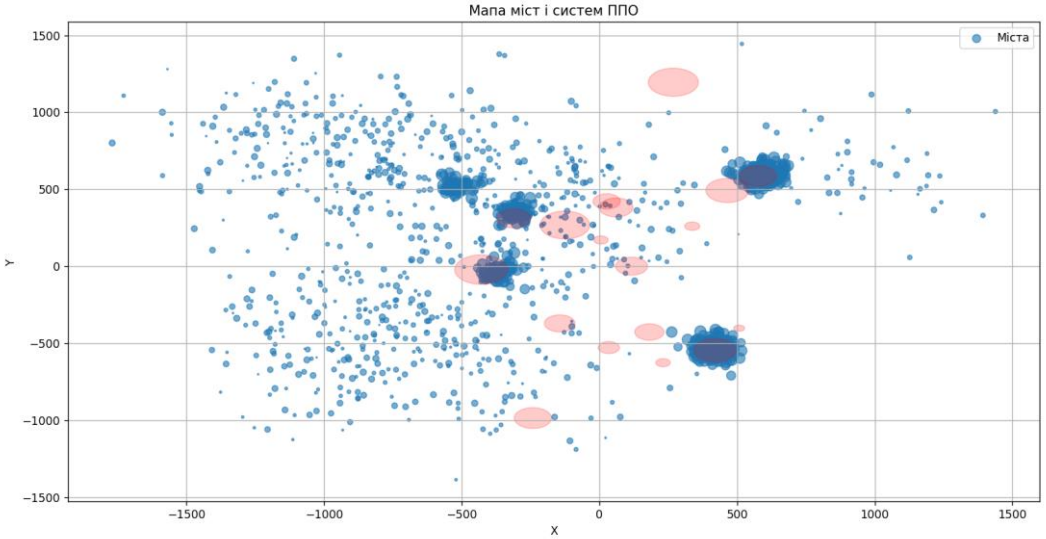
Розмір популяції: 20



Накрито 601 місто.



Накрито 607 міст.

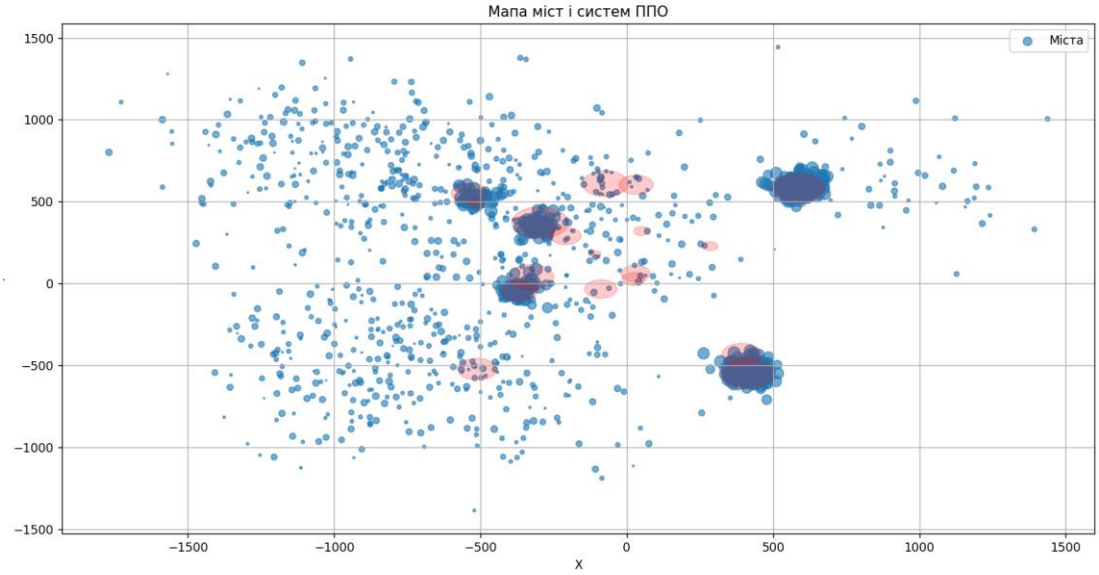


Накрито 581 місто.

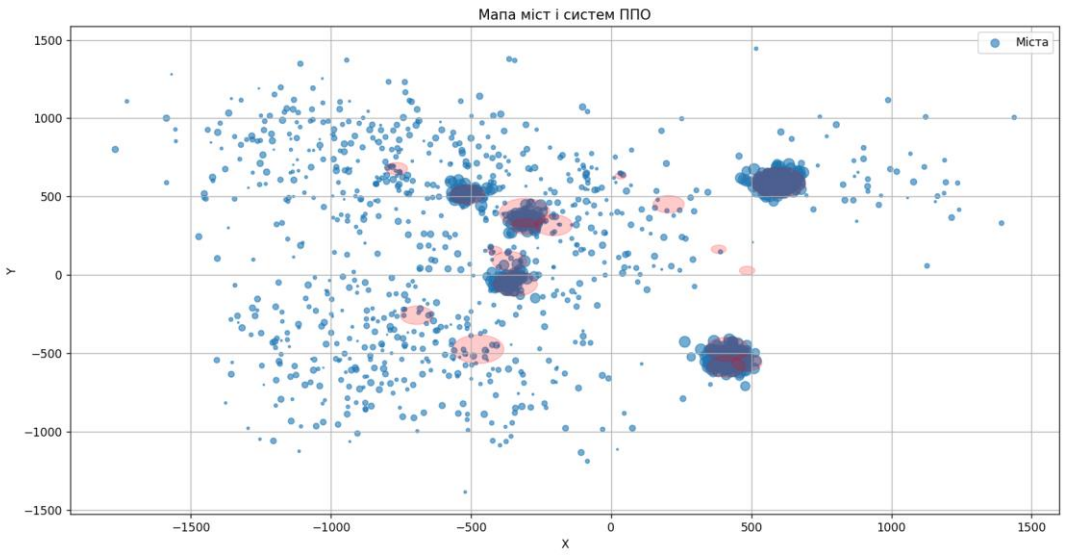
Метод рулетки у поєднанні з аутбрідінгом поки дає найгірші результати. Спробуємо змінити стратегію схрещування на панміксію

Параметри:

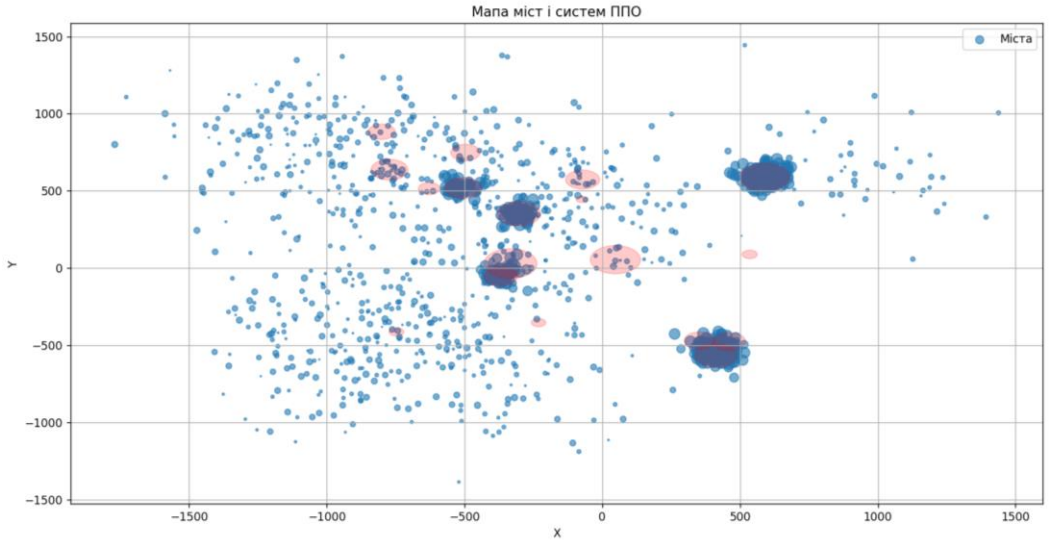
- Схрещування: панміксія, комбінація двох стратегій (усереднення та комбінування координат), ймовірність: 0.9
- Мутація: 1 – одну з координат генеруємо рандомно, ймовірність: 0.1
- Селекція: Топ найкращих
- Розмір популяції: 100



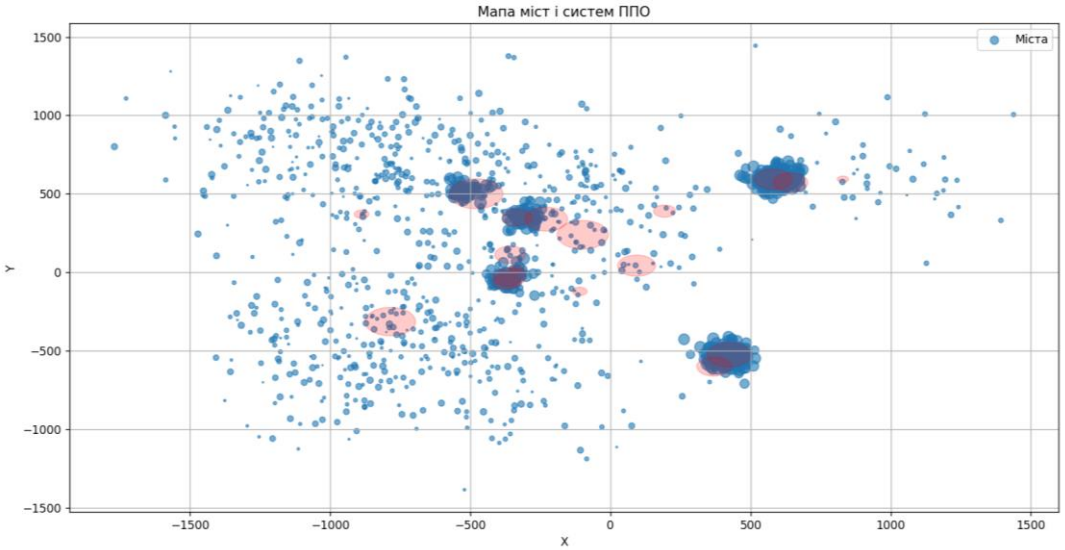
Накрито 780 міст. Це найкраще рішення, яке вдавалось знайти.



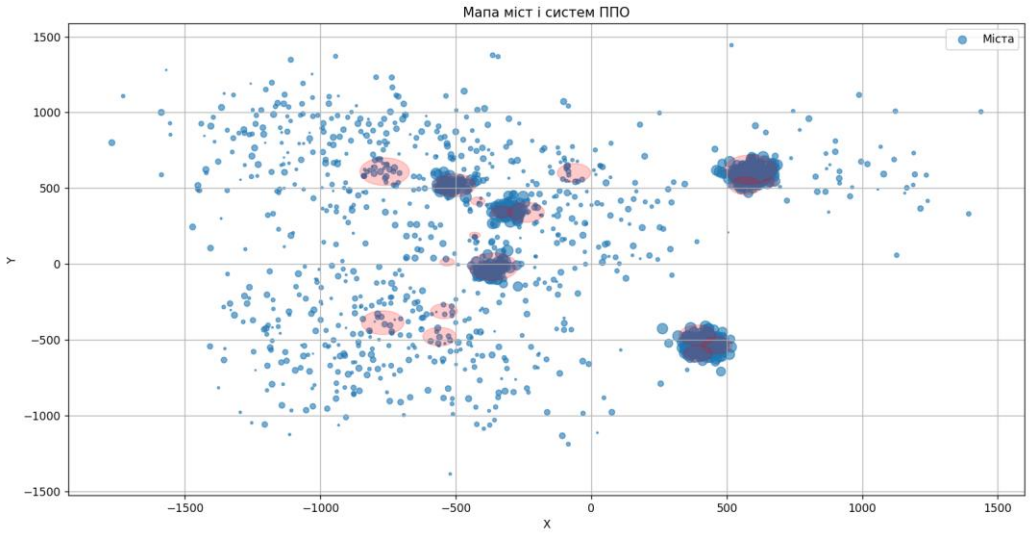
Накрито 782 міста. Кількість хромосом в популяції: 50 та ймовірність мутації 0.2.



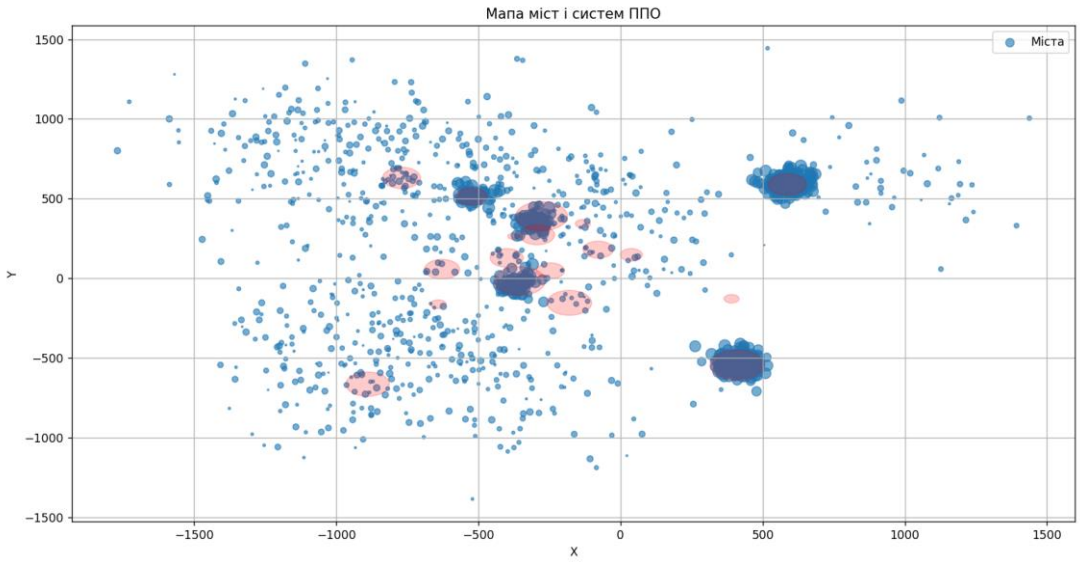
Накрито 788 міст. Розмір популяції: 20 та ймовірність мутації: 0.2.



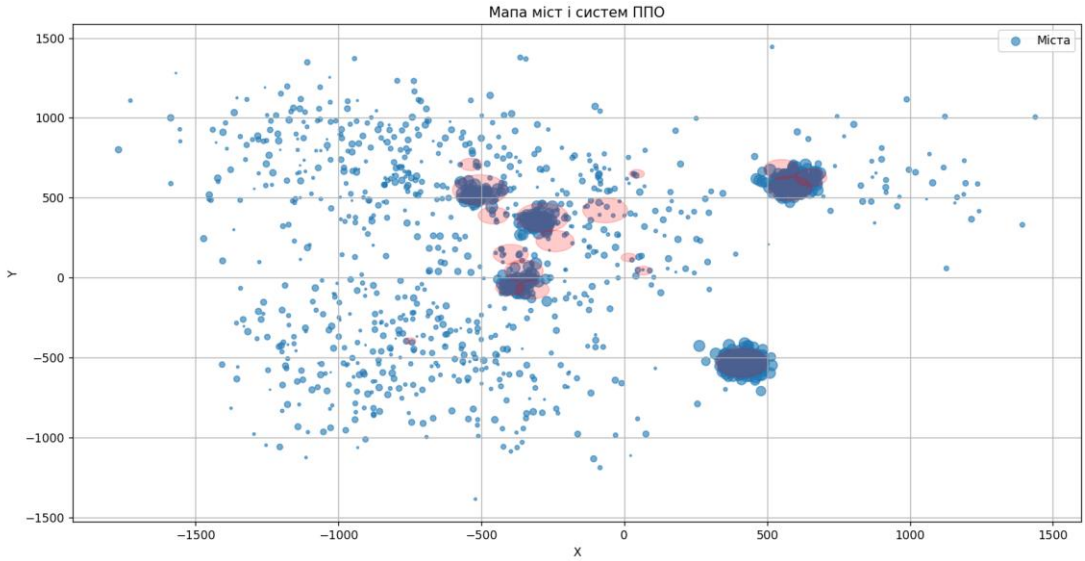
Накрито 753 міста. Розмір популяції: 10 та ймовірність мутації 0.2.



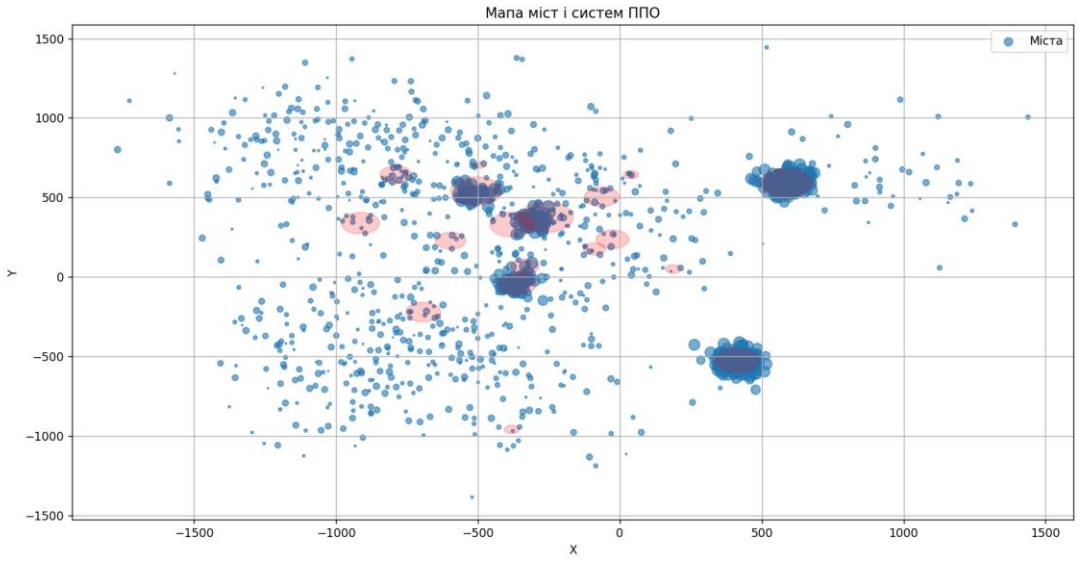
Накрито 802 міста. Розмір популяції 20, ймовірність мутації 0.1.



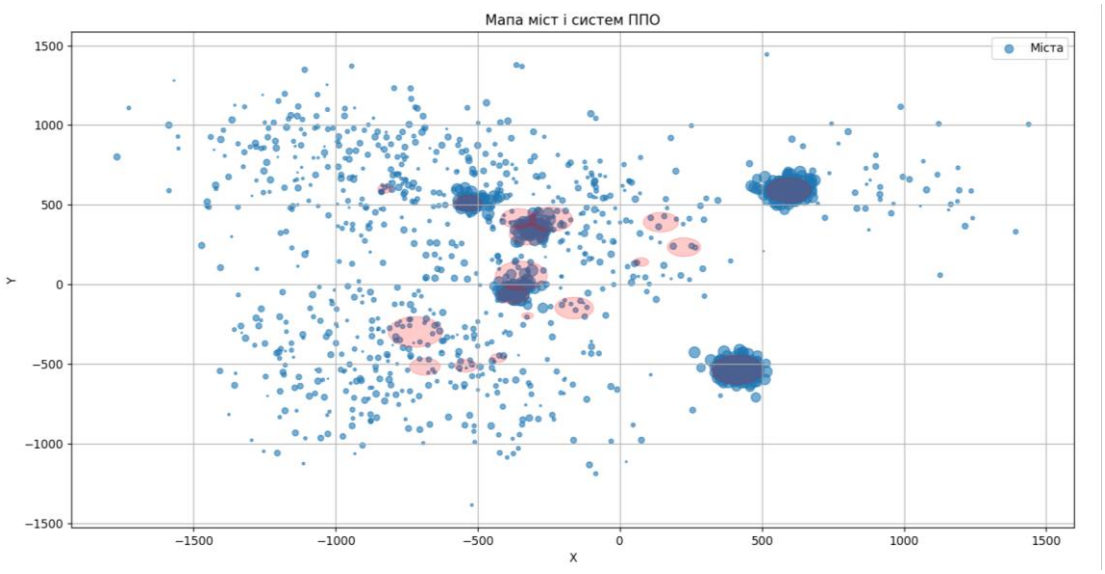
Накрито 773 міста. Розмір популяції 30, ймовірність мутації 0.1.



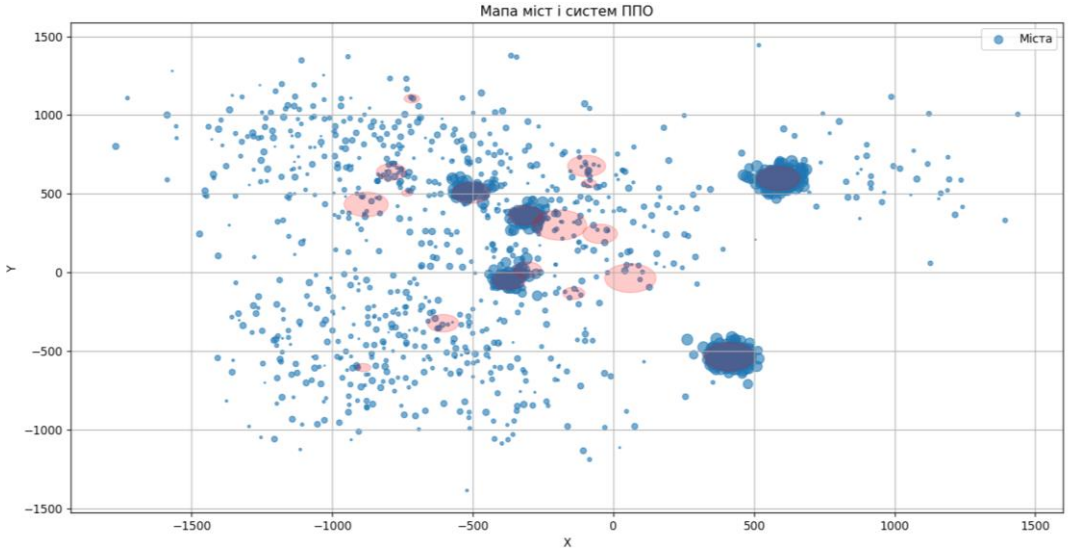
Накрито 794 міста. Розмір популяції 25, ймовірність мутації 0.1.



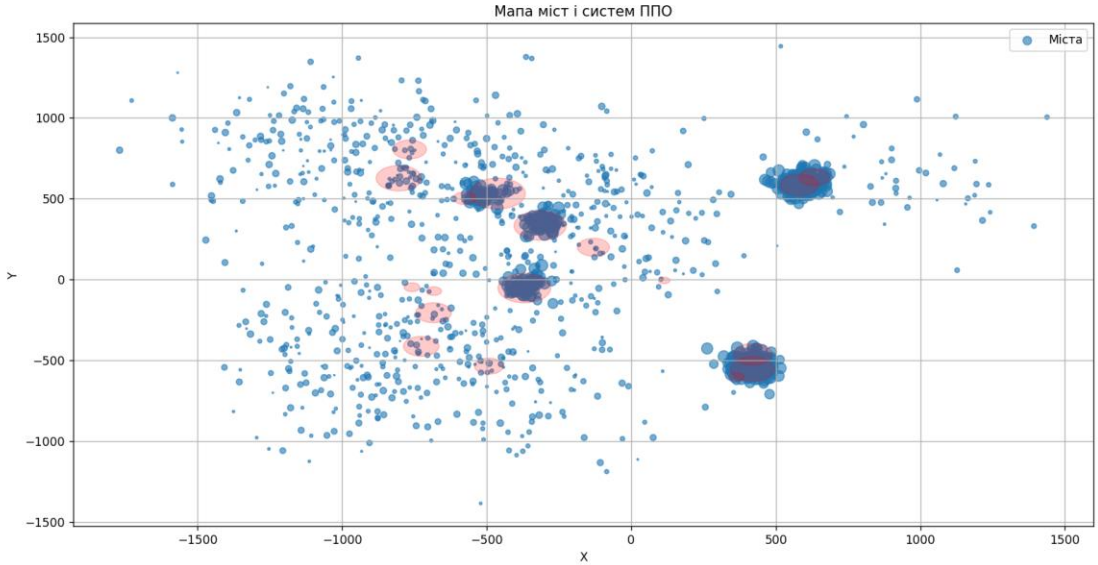
Накрито 785 міст. Розмір популяції 25, ймовірність мутації 0.2.



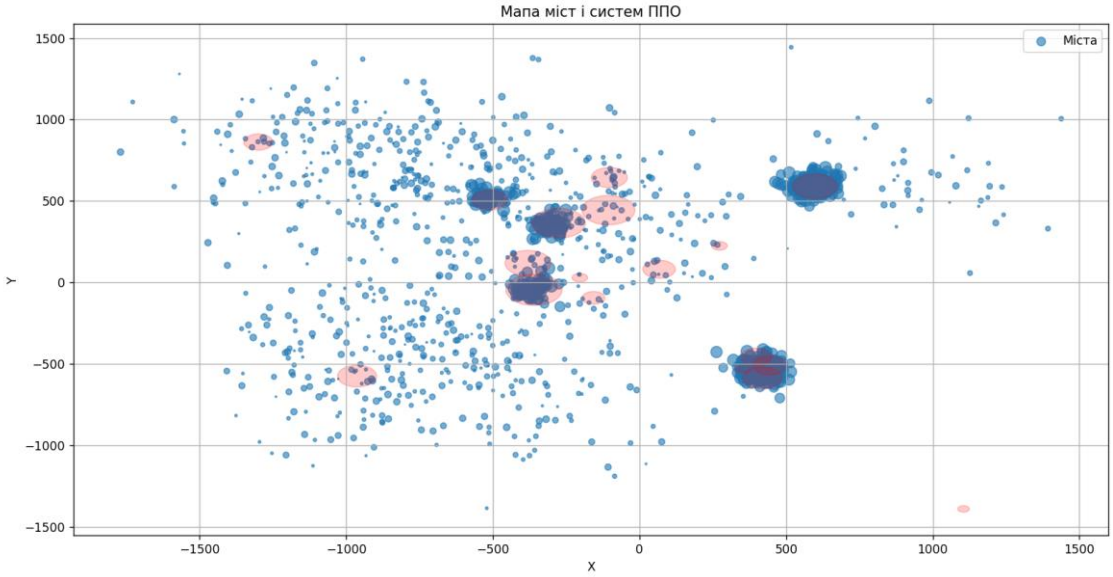
Накрито 773 міста. Розмір популяції 25, ймовірність мутації 0.15.



Накрито 772 міста. Розмір популяції 20, ймовірність мутації 0.15.



Накрито 782 міста. Розмір популяції 20, ймовірність мутації 0.1.



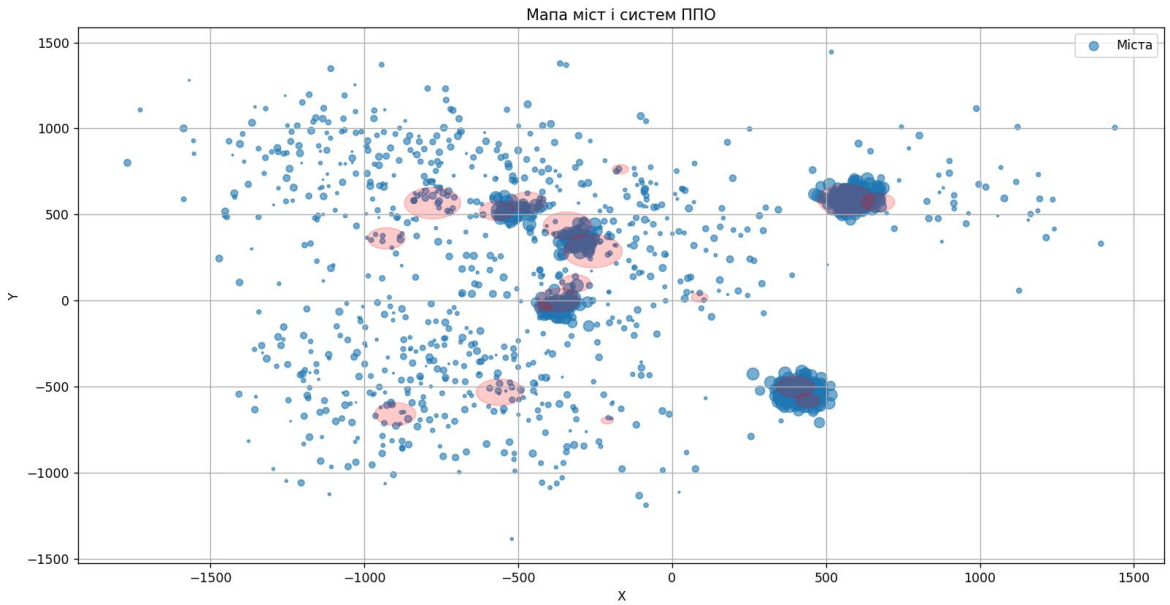
Накрито 775 міст. Розмір популяції 20, ймовірність схрещування 0.05.

Для методу селекції “Топ найкращих” оптимальний розмір популяції становить 20-25 хромосом, а ймовірність мутації 0.1.

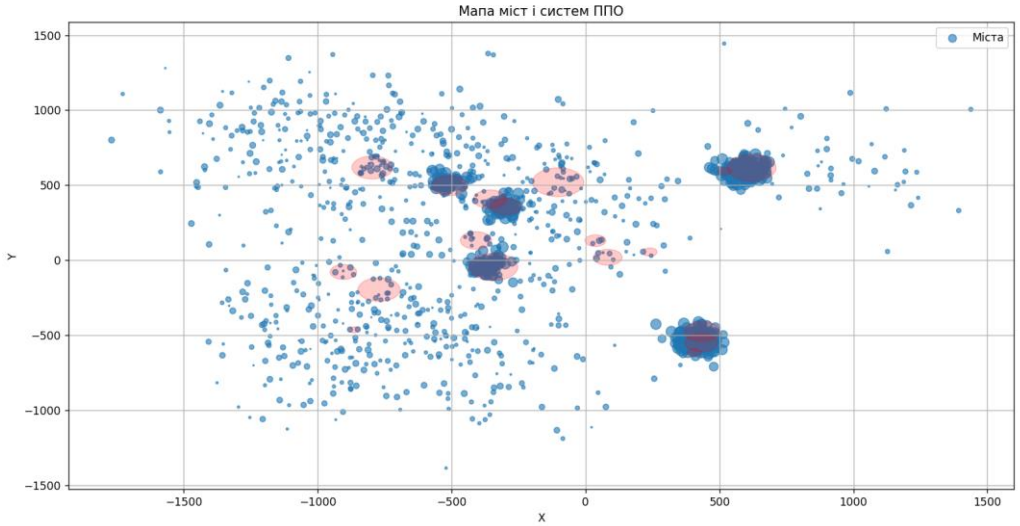
Тепер використаємо стратегію селекції “ Турнірна селекція”:

Параметри:

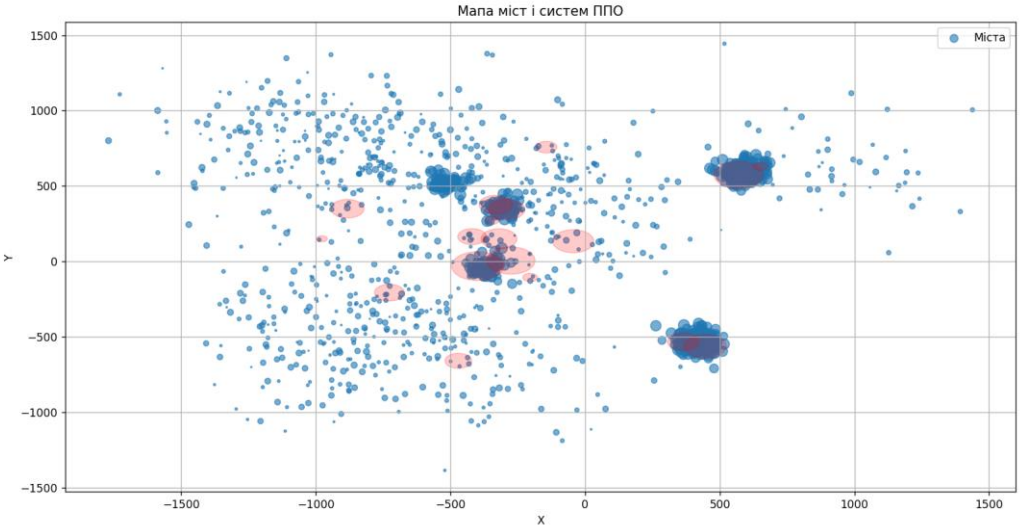
- Схрещування: панміксія, комбінація двох стратегій (усереднення та комбінування координат), ймовірність: 0.9
- Мутація: 1 – одну з координат генеруємо рандомно, ймовірність: 0.1
- Селекція: Турнірна селекція



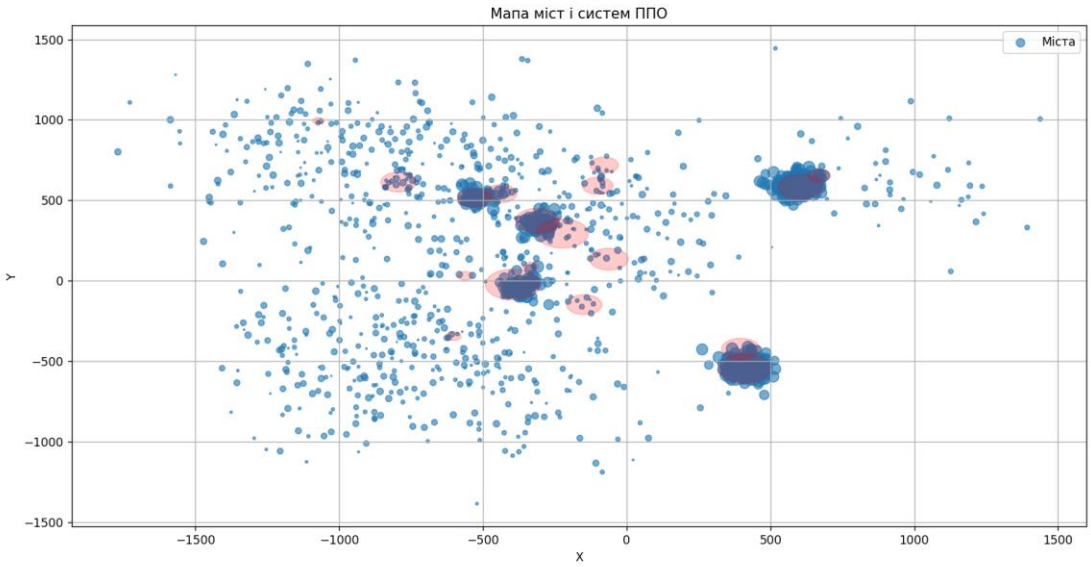
Накрито 710 міст. Розмір популяції 20



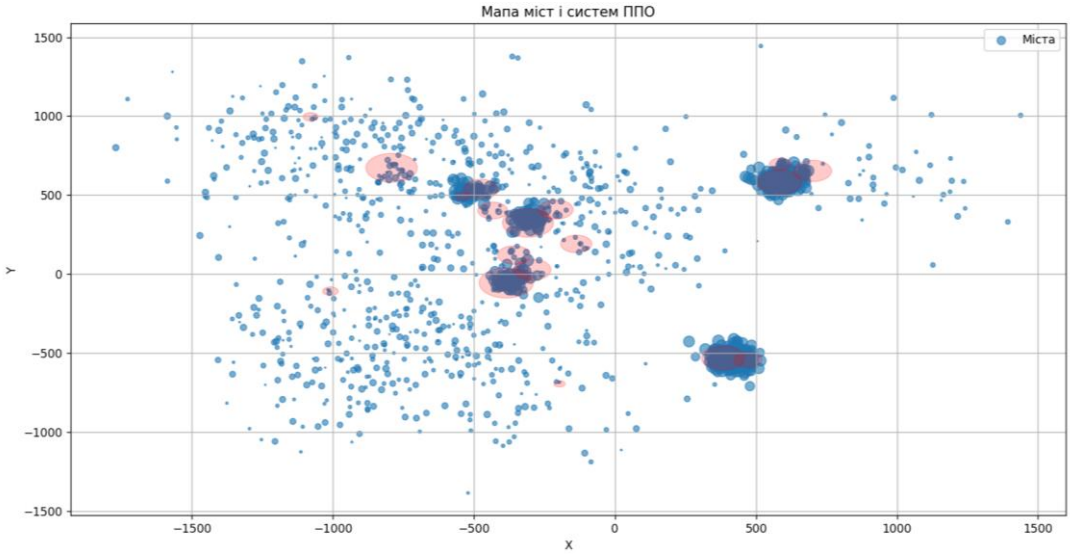
Накрито 759 міст. Розмір популяції 50, ймовірність мутації 0.2



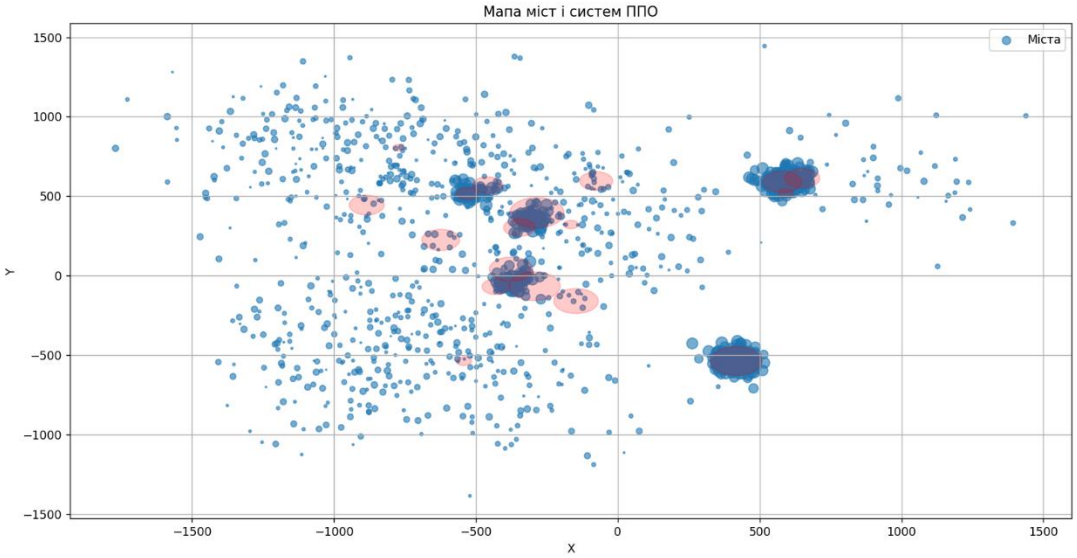
Накрито 697 міст. Розмір популяції 50, ймовірність мутації 0.1



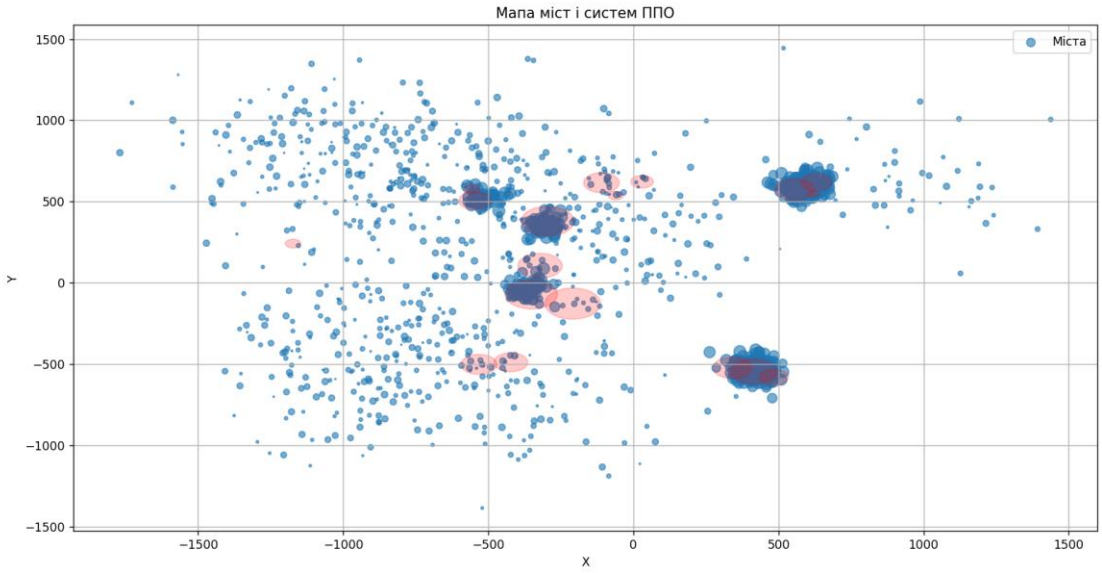
Накрито 778 міст. Розмір популяції 50, ймовірність мутації 0.3



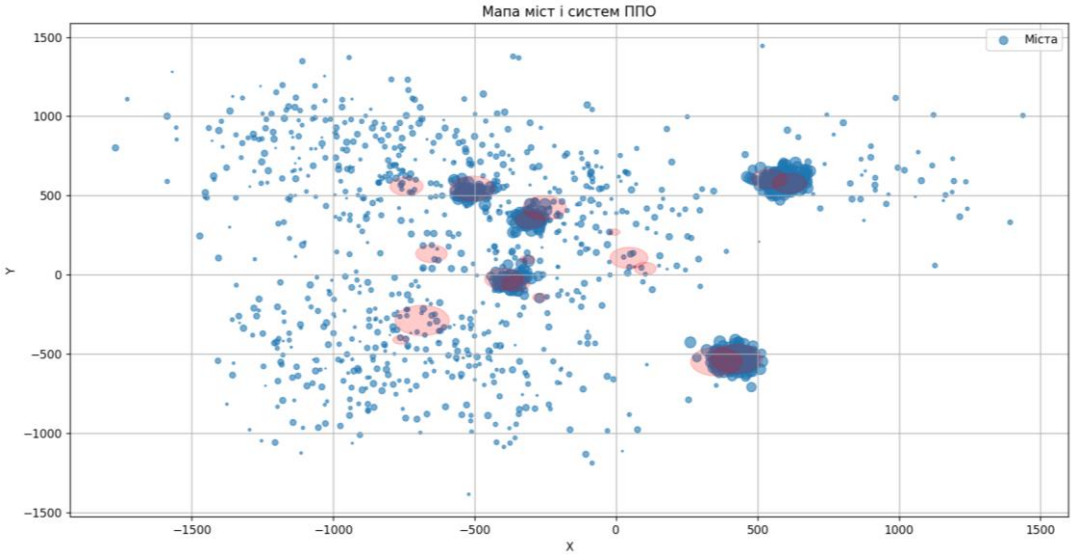
Накрито 752 міста. Розмір популяції 20, ймовірність мутації 0.3.



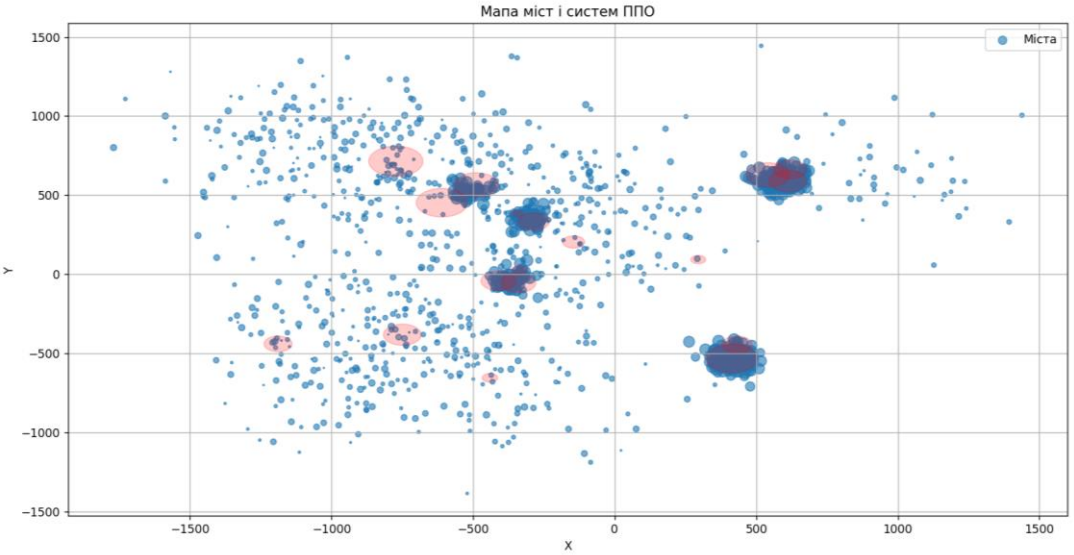
Накрито 759 міст. Розмір популяції 35, ймовірність схрещування 0.3.



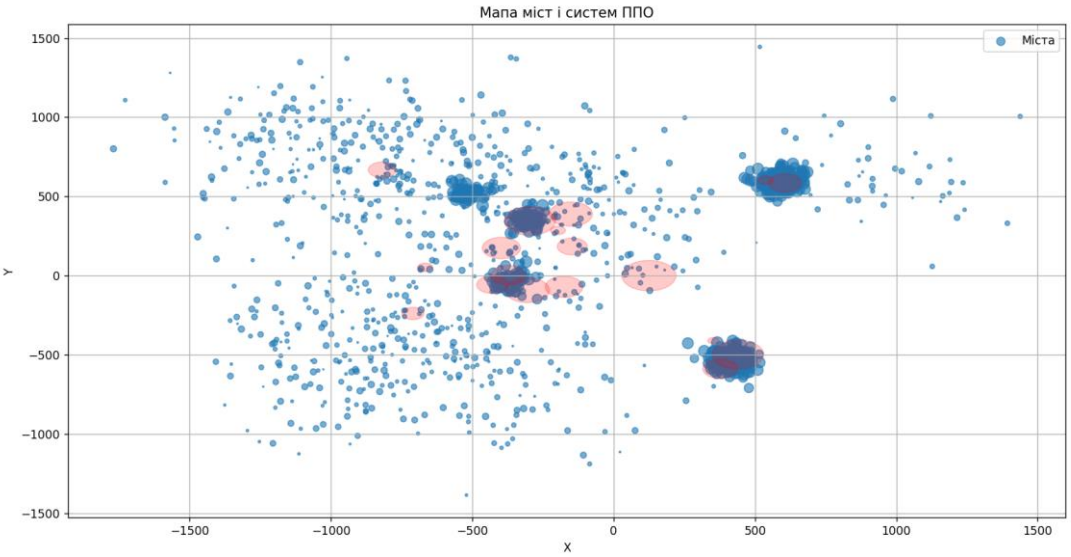
Накрито 747 міст. Розмір популяції 100, ймовірність мутації 0.3.



Накрито 726 міст. Розмір популяції 100, ймовірність мутації 0.2.



Накрито 763 міста. Розмір популяції 50, ймовірність мутації 0.25.



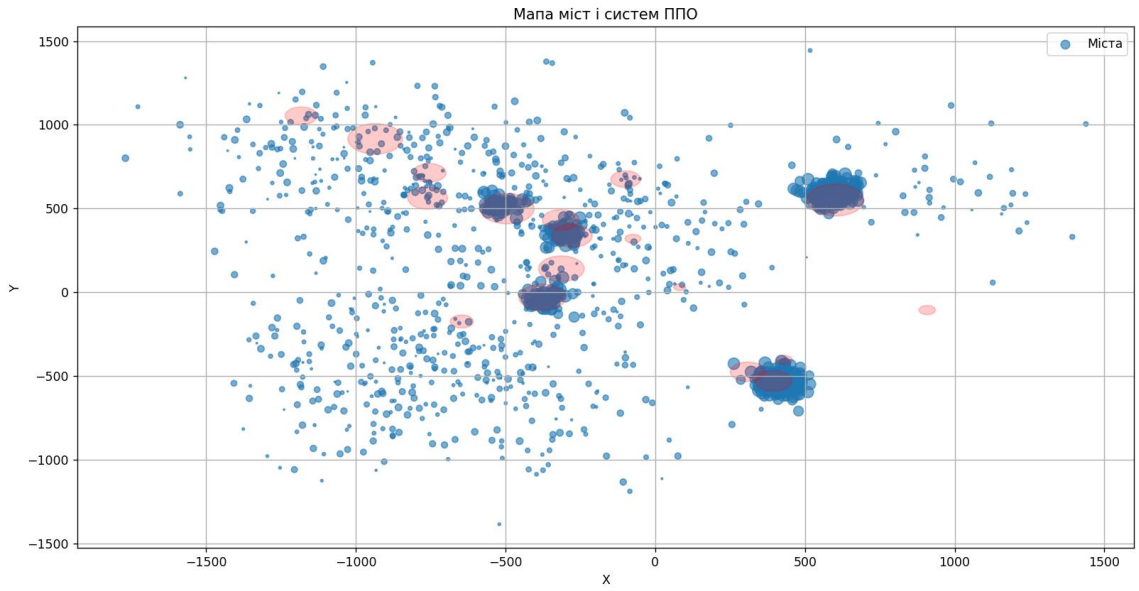
Накрито 689 міст. Розмір популяції 50, ймовірність мутації 0.4.

Отже для методу селекції “Турнірна селекція” з методом вибору батьків панміксія оптимальний розмір популяції становить 50 хромосом, а ймовірність мутації 0.3.

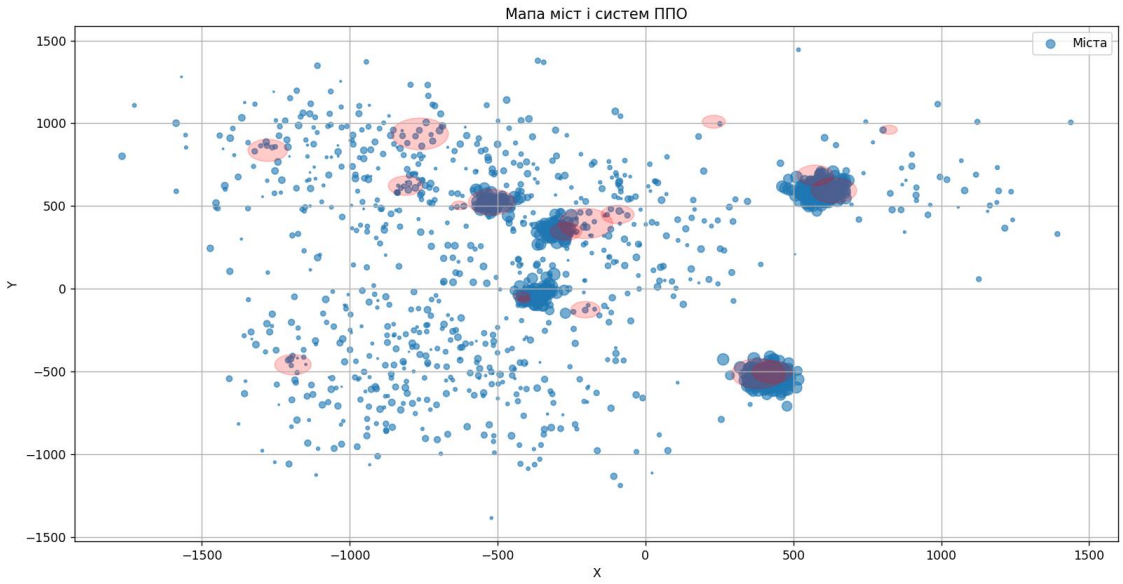
Спробуємо використати стратегію вибору батьків “Інбрідинг”:

Параметри:

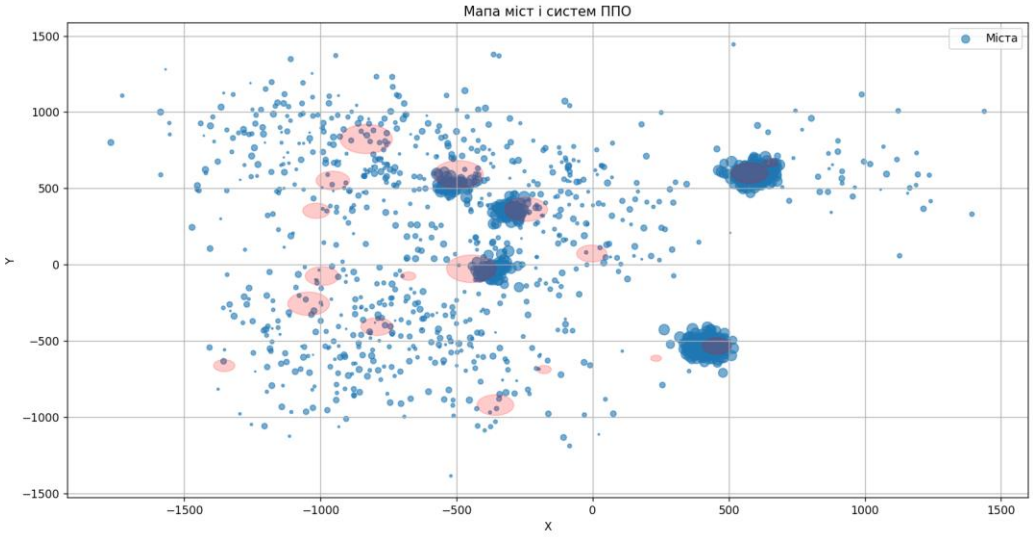
- Схрещування: інбрідінг, комбінація двох стратегій (усереднення та комбінування координат), ймовірність: 0.9
- Мутація: 1 – одну з координат генеруємо рандомно, ймовірність: 0.3
- Селекція: Турнірна селекція



Накрито 683 міста. Розмір популяції 50, ймовірність мутації 0.3.

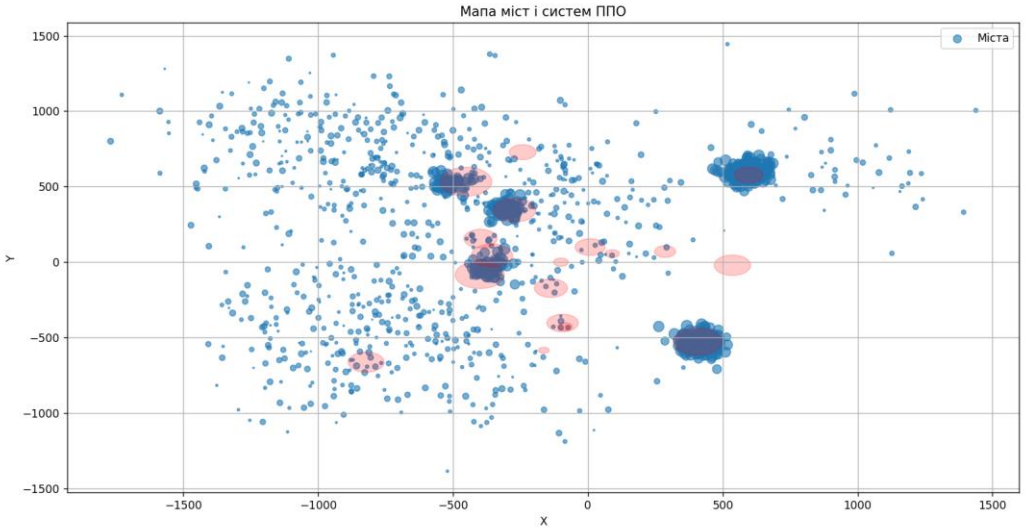


Накрито 621 місто. Розмір популяції 20, ймовірність мутації 0.3.

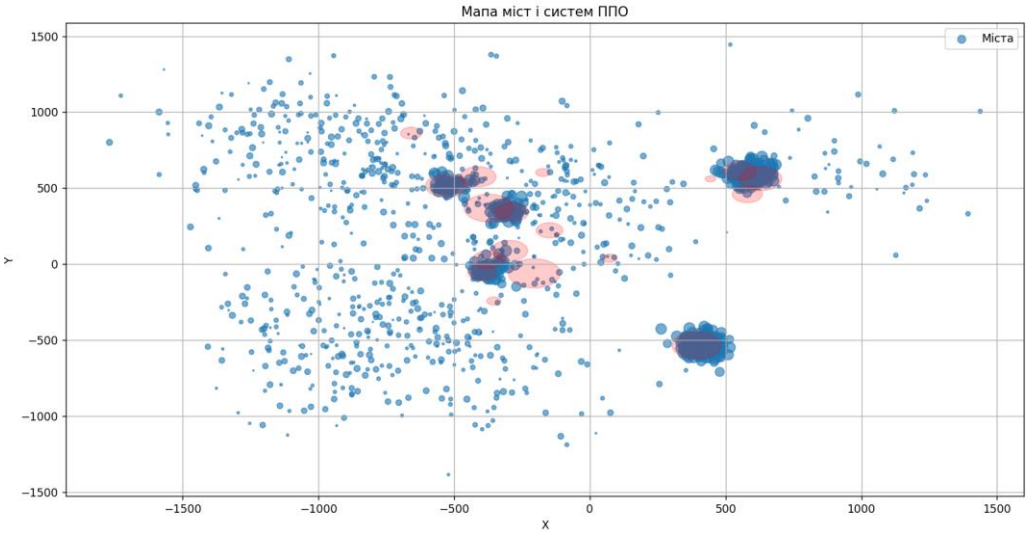


Накрито 525 міст. Розмір популяції 50, ймовірність мутації 0.2.

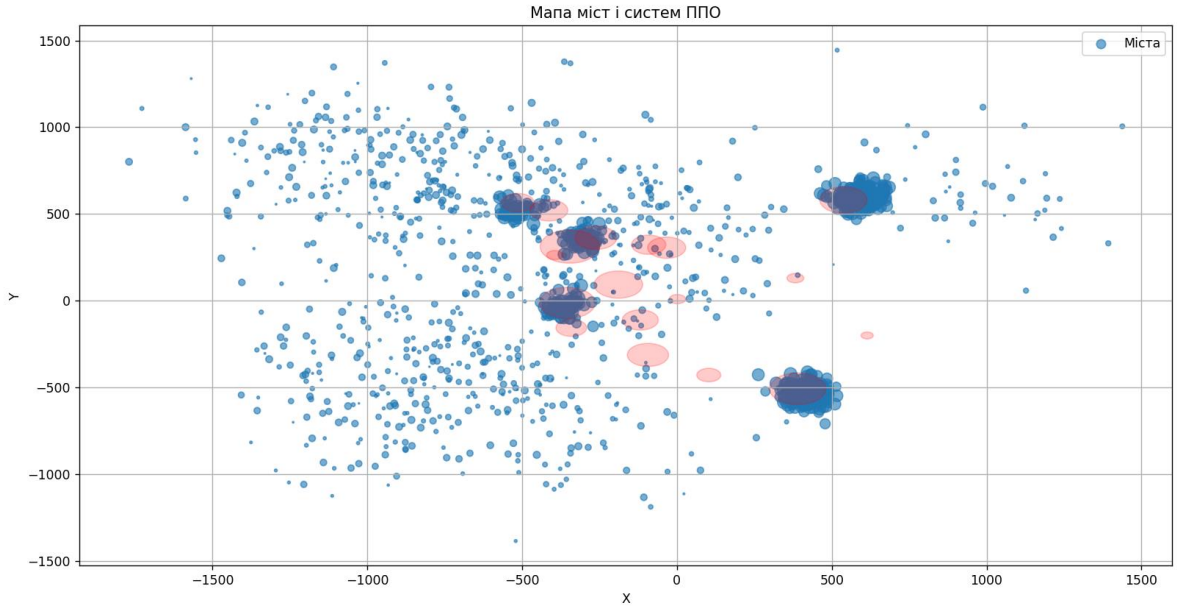
Спробуємо застосувати стратегію “Аутбрідінг”:



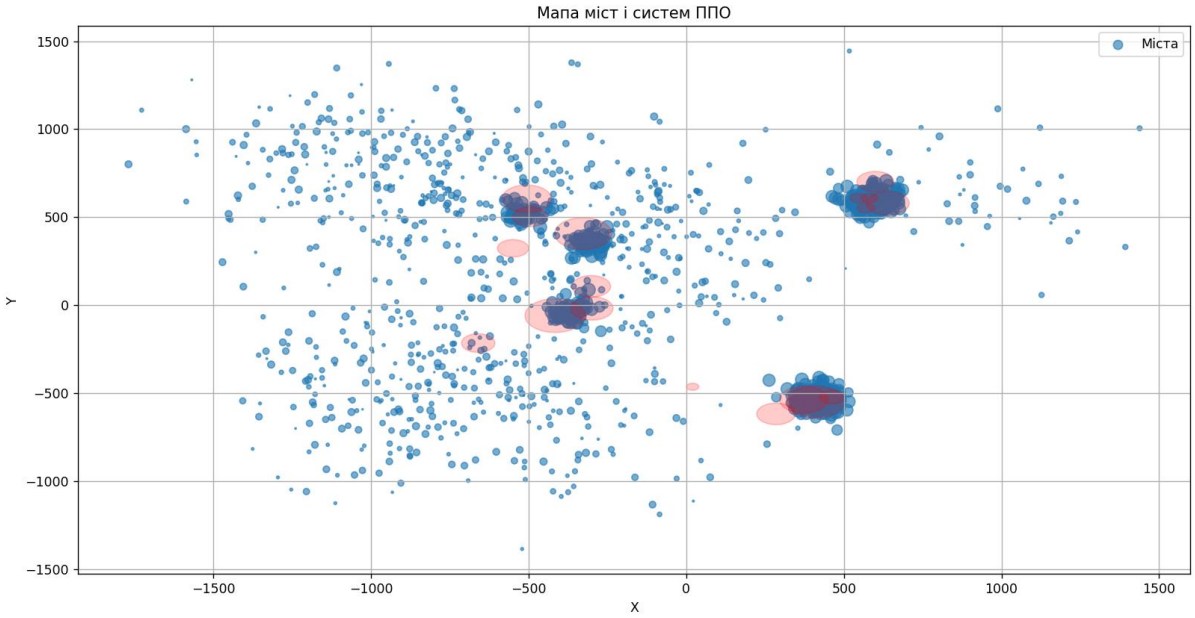
Накрито 662 міста. Розмір популяції 50, ймовірність мутації 0.3.



Накрито 721 місто. Розмір популяції 20, ймовірність мутації 0.3.



Накрито 608 міст. Розмір популяції 50, ймовірність мутації 0.2.



Покрито 695 міст. Розмір популяції 20, ймовірність мутації 0.2.

В результаті тестування було визначено, що найкраща стратегія для схрещування – це комбінація двох стратегій (перша – взяти координату x від першого батька, а координату y від другого, а друга – усереднити координати першого та другого батька).

Найкращий результат було досягнуто при стратегії вибору батьків “Панміксія”. Оптимальний розмір популяції та оптимальна ймовірність мутації залежить від обраного методу селекції.

Краще працює мутація, при якій одна координата генерується випадково.

Зберемо отримані результати в таблиці:

Метод селекції	Стратегія схрещування	Розмір популяції	Ймовірність мутації	Кількість покритих міст
Топ найкращих	Панміксія	20	0.1	**802**
Топ найкращих	Панміксія	25	0.1	794
Топ найкращих	Панміксія	20	0.2	788
Топ найкращих	Панміксія	25	0.2	785
Топ найкращих	Панміксія	20	0.1	782
Топ найкращих	Панміксія	50	0.2	782
Топ найкращих	Панміксія	100	0.1	780
Топ найкращих	Панміксія	20	0.05	775
Топ найкращих	Панміксія	30	0.1	773
Топ найкращих	Панміксія	25	0.15	773
Топ найкращих	Панміксія	20	0.15	772
Топ найкращих	Панміксія	10	0.2	753
Топ найкращих	Аутбрідінг	20	0.1	739
Топ найкращих (мутація 2)	Аутбрідінг	100	0.1	588/681
Турнірна селекція	Панміксія	50	0.3	**778**
Турнірна селекція	Панміксія	50	0.2	763
Турнірна селекція	Панміксія	50	0.2	759
Турнірна селекція	Панміксія	35	0.3	759
Турнірна селекція	Аутбрідінг	20	0.1	754
Турнірна селекція	Аутбрідінг	20	0.1	752
Турнірна селекція	Панміксія	20	0.3	752
Турнірна селекція	Панміксія	100	0.3	747
Турнірна селекція	Панміксія	100	0.2	726
Турнірна селекція	Аутбрідінг	20	0.3	721
Турнірна селекція	Панміксія	20	0.1	710
Турнірна селекція	Панміксія	50	0.1	697
Турнірна селекція	Аутбрідінг	20	0.2	695
Турнірна селекція	Панміксія	50	0.4	689
Турнірна селекція	Інбрідінг	50	0.3	683
Турнірна селекція	Аутбрідінг	50	0.3	662

Турнірна селекція	Аутбрідінг	20	0.1	643
Турнірна селекція	Інбрідінг	20	0.3	621
Турнірна селекція	Аутбрідінг	50	0.2	608
Турнірна селекція	Інбрідінг	50	0.2	525
Метод рулетки	Аутбрідінг	20	0.2	601
Метод рулетки	Аутбрідінг	20	0.2	607
Метод рулетки	Аутбрідінг	20	0.2	581

З таблиці видно, що найкращі результати алгоритм дав саме з стратегією вибору батьків панміксія, стратегією селекції Топ найкращих, розміром популяції 20 та ймовірністю мутації 0.1.

Тепер порівняємо найкращі результати, отримані методом грубої сили, із результатами, отриманими за допомогою генетичного алгоритму:

Брутфорс	Генетичний
763	802
761	794
759	788
739	785
735	782

Бачимо, що генетичному алгоритму вдалося підібрати кращий набір координат для розміщення систем ППО. Але він витратив значно більше часу та виконав значно більшу кількість обчислень цільової функції.

Такий результат пояснюється набором вхідних даних. Брутфорс добре працює, оскільки на карті є райони, де розміщення ППО буде найоптимальнішим. За 100 або 1000 спроб одна точка з високою ймовірністю влучить у цей район і алгоритм збереже ці координати.

Отже для даного набору даних генетичний алгоритм не дає дуже сильного виграшу, але при цьому при чітко підібраних параметрах знаходить більш оптимальні координати.

Висновок: у процесі виконання лабораторної роботи я реалізував генетичний алгоритм, а також різні стратегії схрещувань, мутацій, вибору батьків, селекції. Під час тестування стало зрозуміло, що для генетичного алгоритму критичним є те, як правильно підібрані параметри, оскільки розмір популяції, ймовірність мутації, обрана стратегія схрещування та інші параметри призводять до кардинально різних результатів. Склалося враження, що можна підбирати нескінченну кількість комбінацій стратегій та параметрів, оскільки немає якоїсбь універсальної комбінації. Наприклад, для стратегії селекції “Турнірна селекція” та “Топ найкращих” найкращі значення дають зовсім різні набори параметрів. Так для отримання хорошого результату при “Турнірній селекціхії” ймовірність мутації має бути висока 0.3, а для стратегії “Топ найкращих” таке значення вже призведе до погіршення результату. Я старався підібрати найбільш оптимальні значення, і отримав хороші результати, але можна продовжувати експерементувати далі до нескінченості.