Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №1

Розробка інтелектуальної інформаційної системи на основі мурашиного алгоритму

Виконав
студент групи ІТ-41ф

Новиков Д. М.

Перевірив:

доц. каф. ІСТ
Кравець П.І.

Київ 2025

Мета роботи: навчитися створювати інтелектуальні інформаційні системи на основі мурашиного алгоритму для розв'язування задач комбінаторної оптимізації.

Завдання роботи:

1. Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

2. Розробити за допомогою мов програмування програмне середовище, що реалізує метод мурашиних колоній. При програмній реалізації методу дотримуватися порядку методу мурашиних колоній.

3. Для реалізації агентів, що моделюють поведінку мурах, використовувати структуру, що має наступні поля:

- поточна позиція, в якій знаходиться агент;
- наступна позиція;
- список табу, в якому зберігаються ті пункти, в яких вже побував агент;
- кількість пунктів, що вже відвідав агент;
- масив подорожі, де зберігається послідовність пунктів, в яких побував агент;
- довжина шляху, що пройшов агент (розраховується після закінчення пошуку).

4. Передбачити можливість наглядного (графічного) відображення змін у поточних результатах роботи середовища у вигляді користувацького інтерфейсу.

5. Виділити основні етапи методу в окремі функції, що реалізуються у відповідних т-файлах:

- ініціалізація методу;
- моделювання переміщення агентів;
- вибір наступного пункту;
- оновлення шляхів;
- перезавантаження агентів.
- реалізувати окрему функцію розрахунку довжини шляху.

6. Виконати тестування розробленого програмного середовища за допомогою вирішення конкретних прикладів задачі комівояжера. Задачі (не менше трьох) для виконання тестування програми сформулювати самостійно. Вибір тестових задач обґрунтувати.

7. Порівняти одержані результати вирішення різних прикладів задачі комівояжера. Результати порівняльного аналізу звести до таблиці, попередньо розробивши систему критеріїв порівняння результатів вирішення задачі комівояжера.

8. Оформити звіт з роботи.

Хід роботи:

Вікно ACO parameters використовується для задання стартових налаштувань алгоритму (Рис. 1):

- Nodes (5..100) – кількість точок (міст) у графі;
- Ants count – кількість мурах у кожній ітерації. Більше мурах – більша вибірка/стабільність, але довше виконання;
- Iterations – кількість ітерацій алгоритму;

- Alpha (pheromone importance) – вага феромону під час вибору наступної вершини. Чим більше, тим сильніше мурахи слідують накопиченим слідам;
- Beta (distance importance) – вага оберненої відстані. Більше – віддає перевагу коротшим ребрам (поведінка більш «жадібна»);
- Evaporation rate (0..1) – коефіцієнт випаровування/затримки феромону за ітерацію. У реалізації множиться на матрицю феромонів:
  - ρ=0.6 означає, що 60% поточного рівня феромону зберігається (40% зникає).
- Q (pheromone deposit multiplier) – коефіцієнт депонування феромону на пройдених ребрах;
- Map seed (int, reproducible layout) – зерно (seed) генератора випадкових координат. Однакове значення – однакова карта;
- Start point (None or integer index) – початкова вершина:
  - None – кожна мураха стартує з випадкової вершини (поведінка за замовчуванням);
  - ціле число i – усі мурахи стартують з вершини з індексом i.
- Pheromone top % (to show top edges) – відсоток «найсильніших» ребер за феромоном, що відображаються на карті (виключно для візуалізації; на розрахунки не впливає);
- Show live animation – прапорець запуску з анімацією. Якщо зняти – алгоритм працює швидко, після чого відображається кінцевий стан;
- Кнопки Start (запуск) та Cancel (скасування).

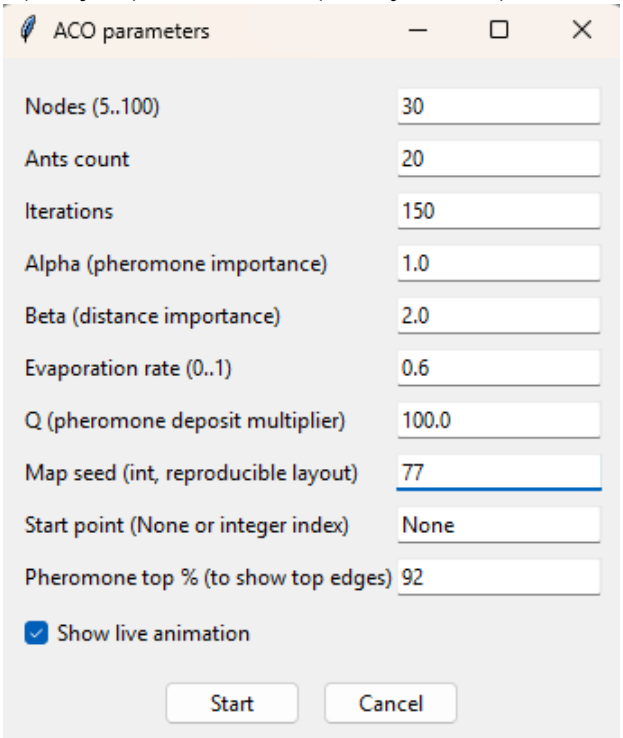


Рис 1 – Приклад ACO parameters

Після запуску відкривається фігура з чотирма панелями (Рис. 2):

1) Ліва верхня панель – Live pheromone + current iteration best path

- Жовті кола з чорним обводом – вершини (номери – індекси від 0);

- Товста чорна ламана – поточний глобально найкращий маршрут, замкнений у цикл;
- Сині напівпрозорі ребра – ребра з найбільшим рівнем феромону (відсічення за «Pheromone top %»). Товщина/насиченість відображає силу феромону (якщо увімкнено живу анімацію), бліді сині траси показують пройдені за ітерацію шляхи мурах;

2) Права верхня панель – Reference: all potential paths + distances

Показує повний граф усіх можливих ребер (світло-сірі лінії) для орієнтиру. Це довідкова панель; на роботу алгоритму не впливає.

Для $N \leq 30$ біля середин ребер додатково виводяться числові довжини (евклідова відстань).

3) Ліва нижня панель – інформаційний блок (жовте вікно)

- Iter: номер поточної/загальної ітерації;
- Current iter best: найкраща довжина серед маршрутів, знайдених у цій ітерації;
- Global best: найкраща довжина за весь час роботи.
- Best found at iter: номер ітерації, на якій вперше знайдено глобально найкращий маршрут (b-step);
- Best route (indices): перелік вершин у порядку обходу (замкнений цикл).
- Ant distances (smallest shown): короткий список найменших довжин маршруту, знайдених окремими мурахами в поточній ітерації.

4) Права нижня панель – Convergence (global best distance per iteration)

Графік збіжності:

на осі x – ітерація;

y – глобально найкраща довжина на поточній ітерації.

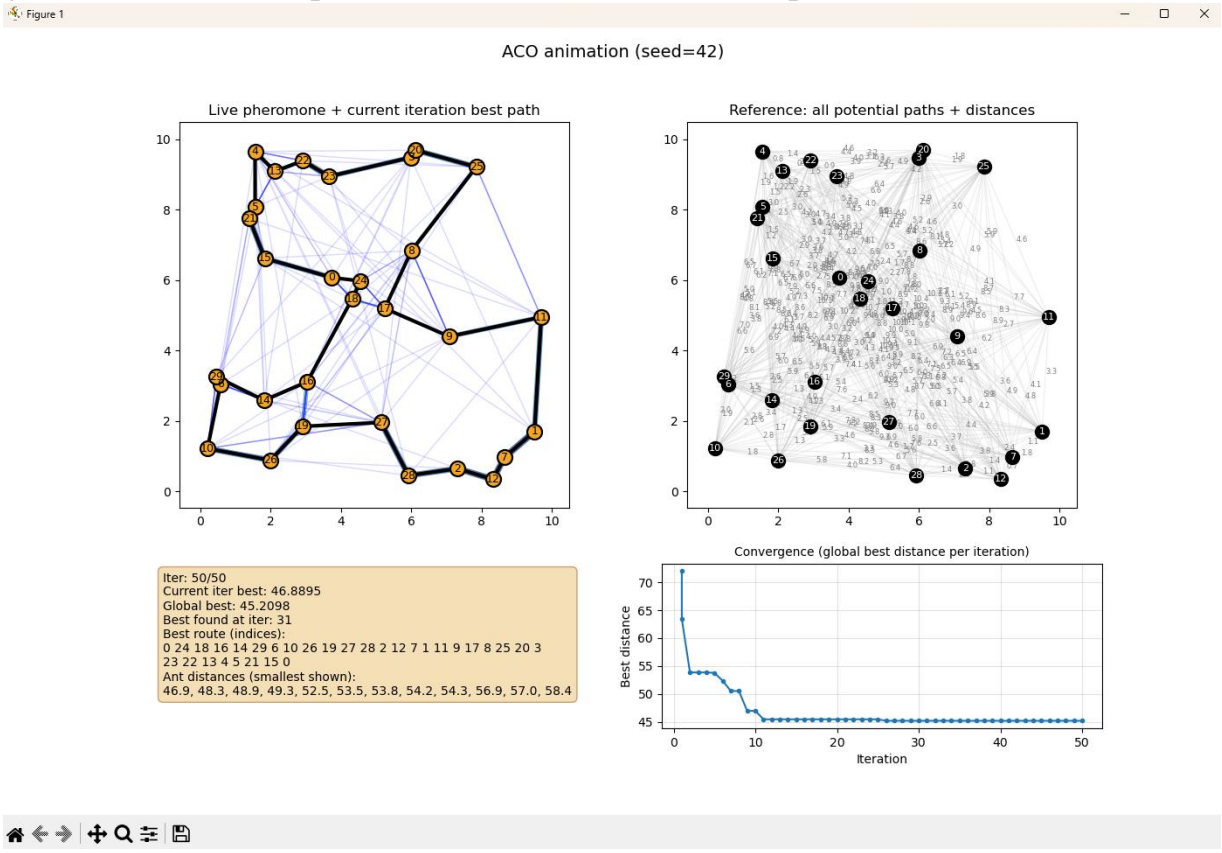


Рис 2 – Приклад основного інтерфейсу програми

Реалізований алгоритм може працювати у двох режимах.

1. Кожна мураха стартує з випадкового міста та шукає найкоротший шлях до усіх інших міст;
2. Усі мурахи стартують з одного певно вказаного міста.

Мета експериментів: порівняти якість маршрутів, швидкість збіжності та обчислювальні витрати методу мурашиних колоній (ACO) на повністю випадкових евклідових картах для різних бюджетів «мурахи * ітерації».

Умови та налаштування:

1. Мапа: випадкове розміщення точок у прямокутнику [0,10]×[0,10] із фіксованим map_seed (напр., 77).
2. Стартова вершина:
    a. start_point = None (кожна мураха стартує випадково);
    b. start_point = N/2.
3. Повторюваність: 3 запусків на кожен сценарій і розмір задачі N (різні внутрішні стани алгоритму, одна й та сама карта через map_seed).

Параметри ACO (за замовчуванням):

1. alpha = 1.0 (вага феромону);
2. beta = 2.5 (вага відстані);
3. Evaporation rate = 0.60;
4. Q = 100;
5. Розміри задачі: $N \in \{20,50,100\}$.

Показники, що фіксуються:

1. Best distance – найкраща довжина туру (менше – краще);
2. b-step – номер ітерації, на якій уперше знайдено глобально найкращий маршрут;
3. Avg – середнє значення для усіх повторів.

Сценарії експериментів:

- Швидкий пошук: швидкий нижній орієнтир; перевірка адекватності параметрів:
    - Мурахи: 10;
    - Ітерації: 50;
    - Очікування: швидке знаходження прийнятного маршруту.
- Збалансований пошук: компроміс між якістю та часом:
    - Мурахи: [0.5*N];
    - Ітерації: 100;
    - Очікування: покращення Швидкого на ~5%, b-step зазвичай у межах 40–70% ітерацій.
- Насичений пошук: максимальна якість:
    - Мурахи: min (N, 100);
    - Ітерації: 200;

- Очікування: ще 1-3% виграш проти Збалансованого, особливо на великих N (>50).

План запусків:
Для кожного N∈{20,50,100} і кожного сценарію:

- виконати 3 запусків (як для випадкових стартових точок так і для визначеної);
- зберегти: Best (мін), b-step, Avg (середній Best/b-step);

Приклад роботи:
N = 25:

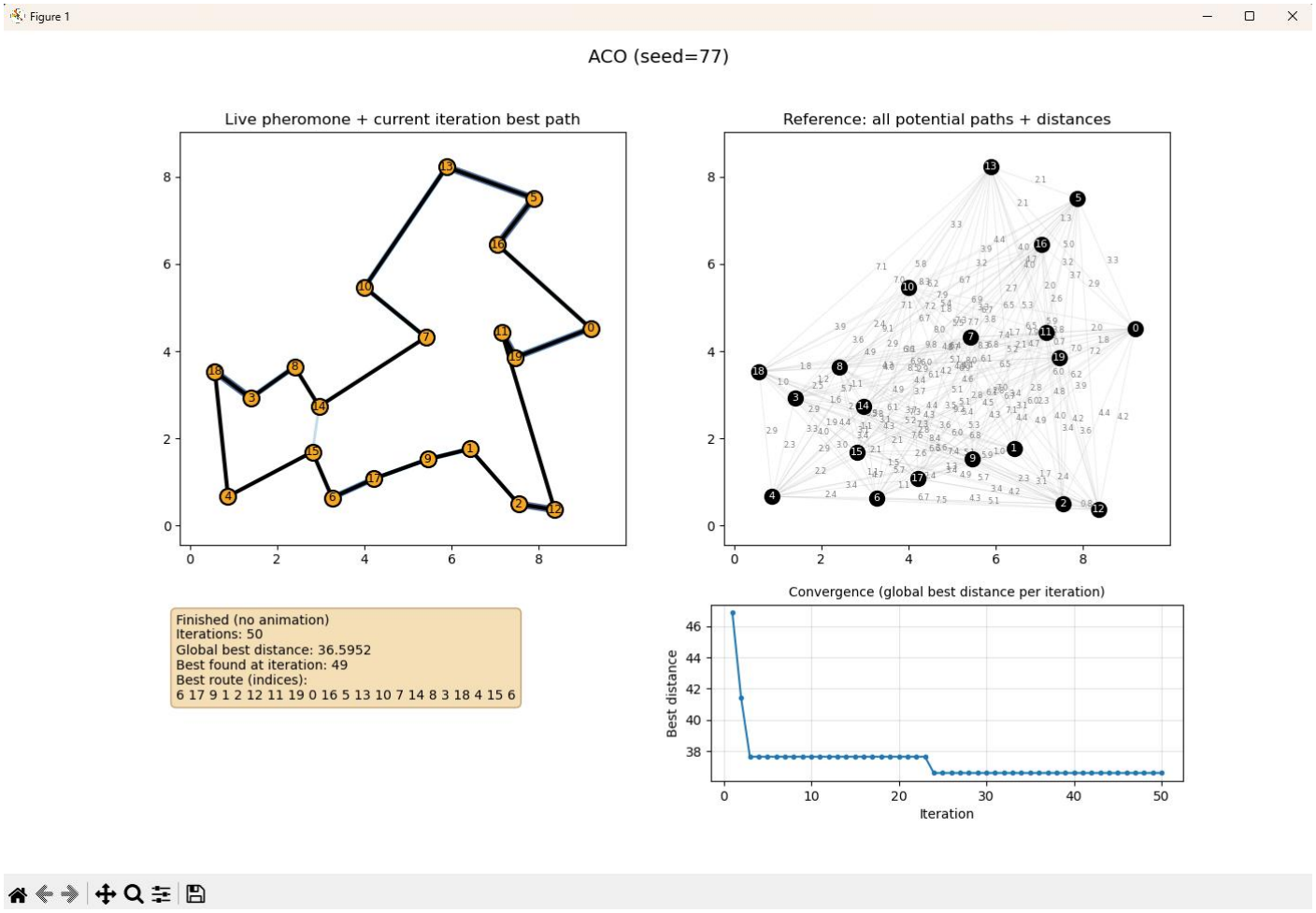
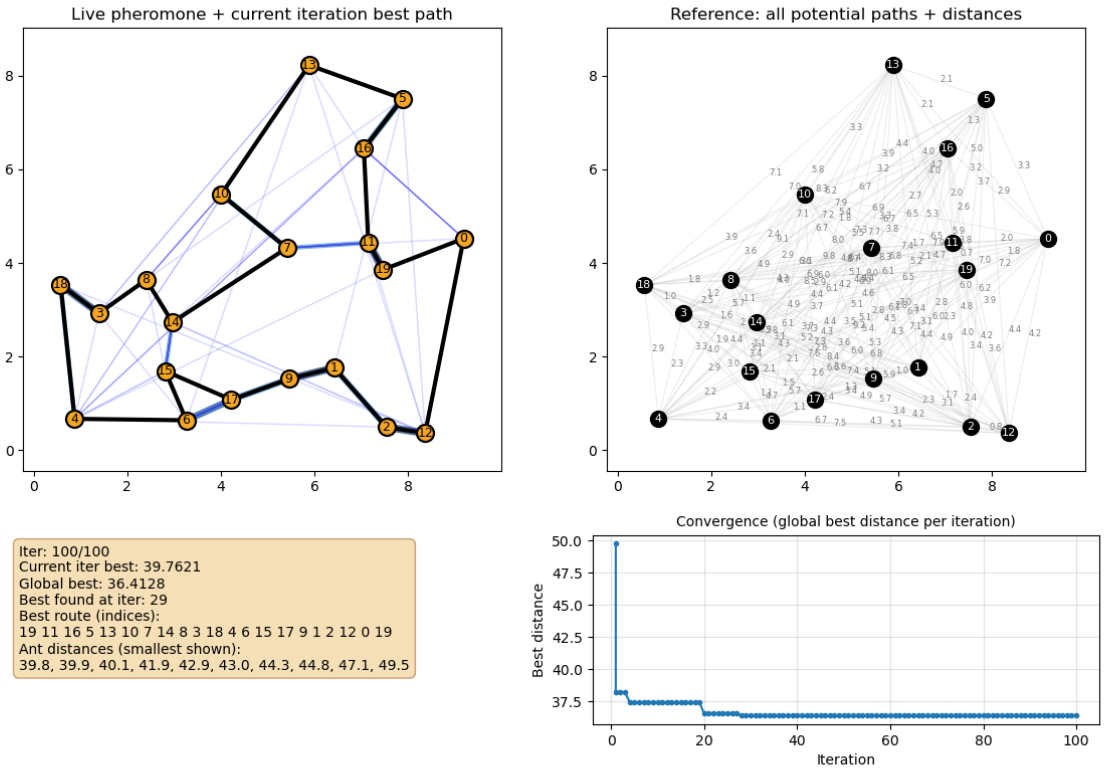


Figure 1

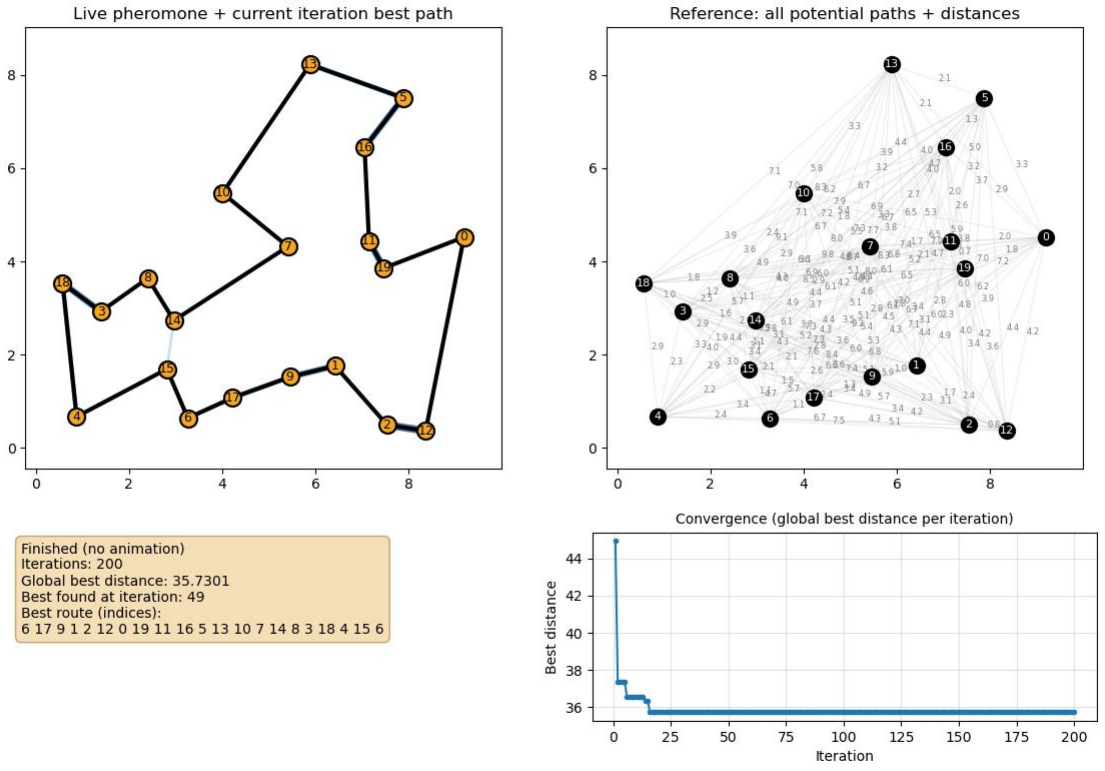
ACO animation (seed=77)



Iter: 100/100
Current iter best: 39.7621
Global best: 36.4128
Best found at iter: 29
Best route (indices):
19 11 16 5 13 10 7 14 8 3 18 4 6 15 17 9 1 2 12 0 19
Ant distances (smallest shown):
39.8, 39.9, 40.1, 41.9, 42.9, 43.0, 44.3, 44.8, 47.1, 49.5

Figure 1

ACO (seed=77, N=20, ants=100)



Finished (no animation)
Iterations: 200
Global best distance: 35.7301
Best found at iteration: 49
Best route (indices):
6 17 9 1 2 12 0 19 11 16 5 13 10 7 14 8 3 18 4 15 6

(x, y) = (3.94, 5.68)

Iteration	Швидкий (ants = 10, iterations = 50)		Збалансований (ants = 10, iterations = 100)		Насичений (ants = 100, iterations = 200)	
	Best	b-step	Best	b-step	Best	b-step
1	36.5952	49	35.7301	15	35.7301	70
2	35.7301	12	35.7301	59	35.7301	49
3	35.7301	28	36.4128	29	35.7301	24
Avg	36.0185	30	35.9577	34	35.7301	48

N = 50:

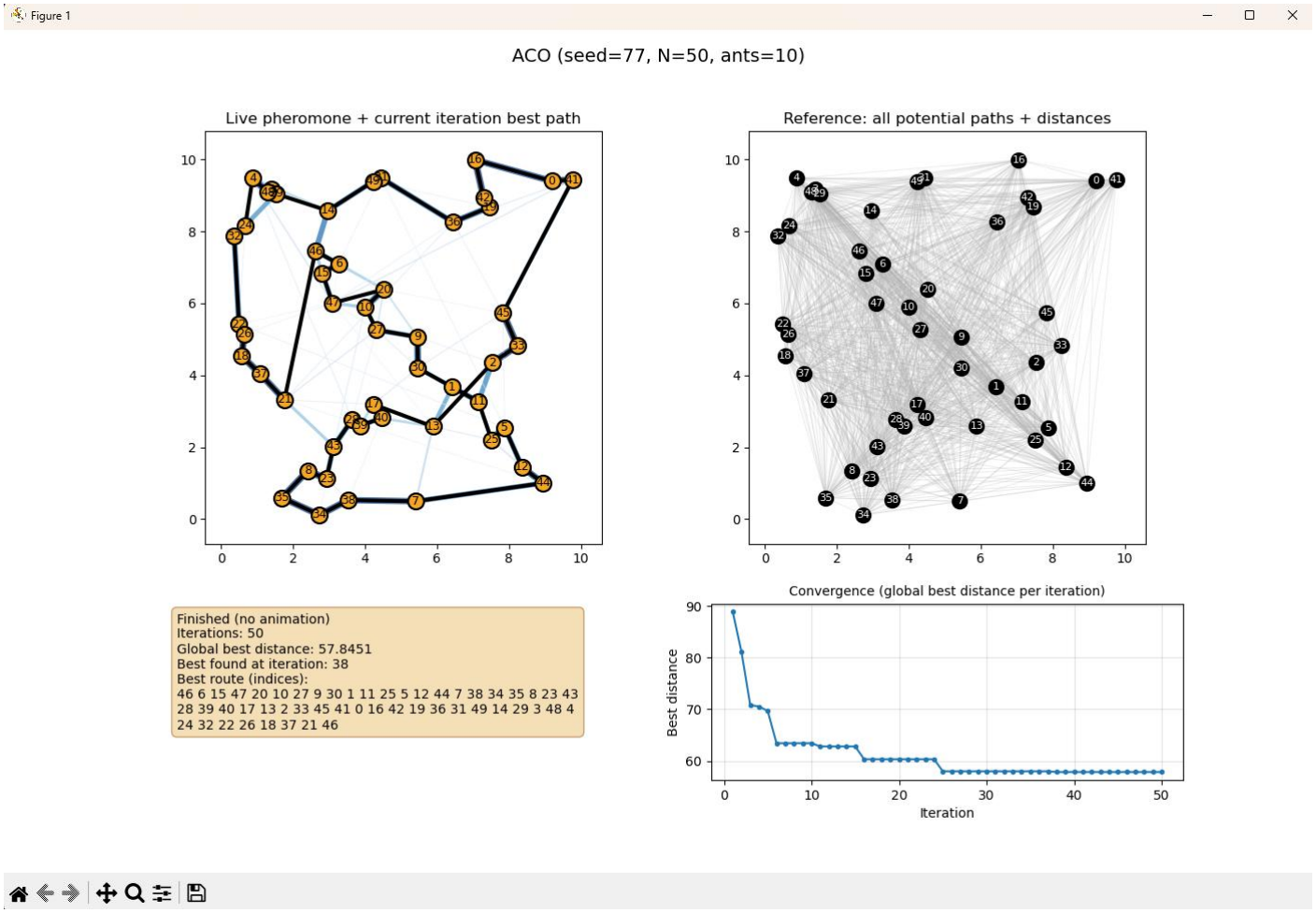
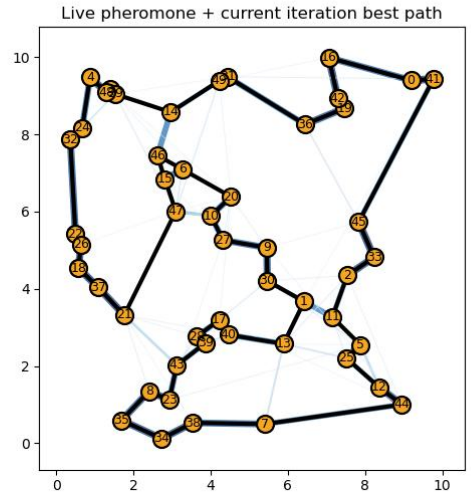
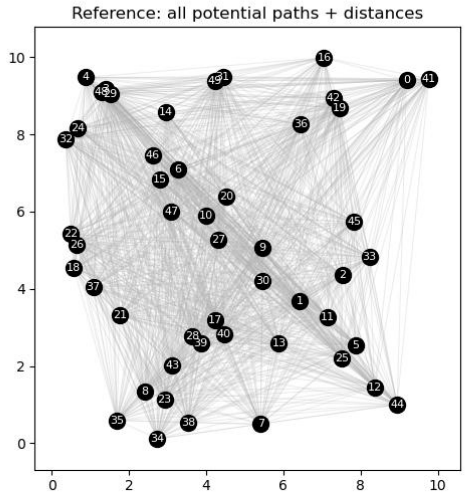


Figure 1

ACO (seed=77, N=50, ants=25)





Finished (no animation)
Iterations: 100
Global best distance: 55.4609
Best found at iteration: 70
Best route (indices):
47 15 46 6 20 10 27 9 30 1 13 40 17 28 39 43 23 8 35 34 38 7 44 12 25 5 11 2 33 45 41 0 16 42 19 36 31 49 14 29 3 48 4 24 32 22 26 18 37 21 47

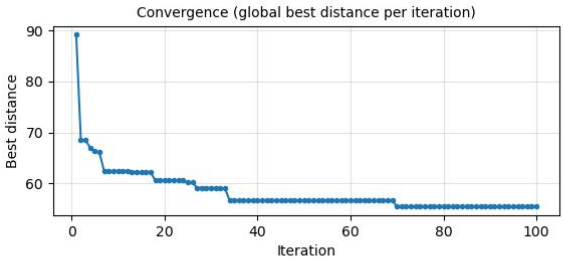
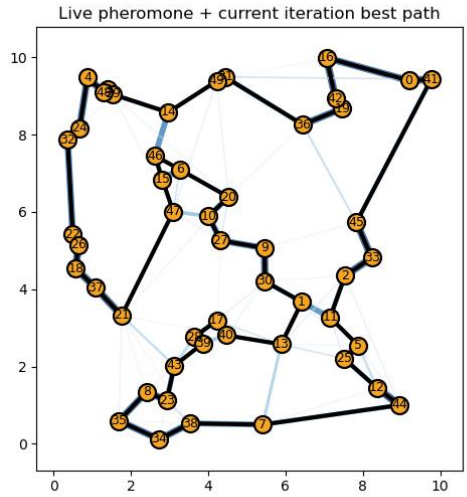
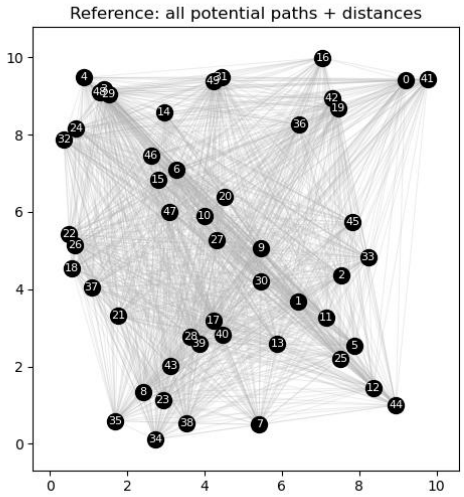


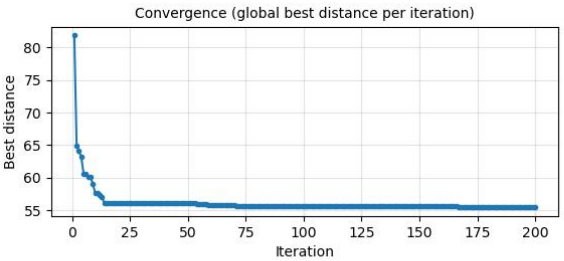
Figure 1

ACO (seed=77, N=50, ants=100)





Finished (no animation)
Iterations: 200
Global best distance: 55.4609
Best found at iteration: 181
Best route (indices):
35 8 23 43 39 28 17 40 13 1 30 9 27 10 20 6 46 15 47 21 37 18 26 22 32 24 4 48 3 29 14 49 31 36 19 42 16 0 41 45 33 2 11 5 25 12 44 7 38 34 35



Iteration	Швидкий (ants = 10, iterations = 50)		Збалансований (ants = 25, iterations = 100)		Насичений (ants = 100, iterations = 200)	
	Best	b-step	Best	b-step	Best	b-step
1	57.8451	38	55.4609	70	55.4609	181
2	55.4609	47	56.0716	54	55.4206	18
3	57.0436	40	56.203	70	55.4609	129
Avg	56.7832	42	55.9118	65	55.4475	109

N = 100:

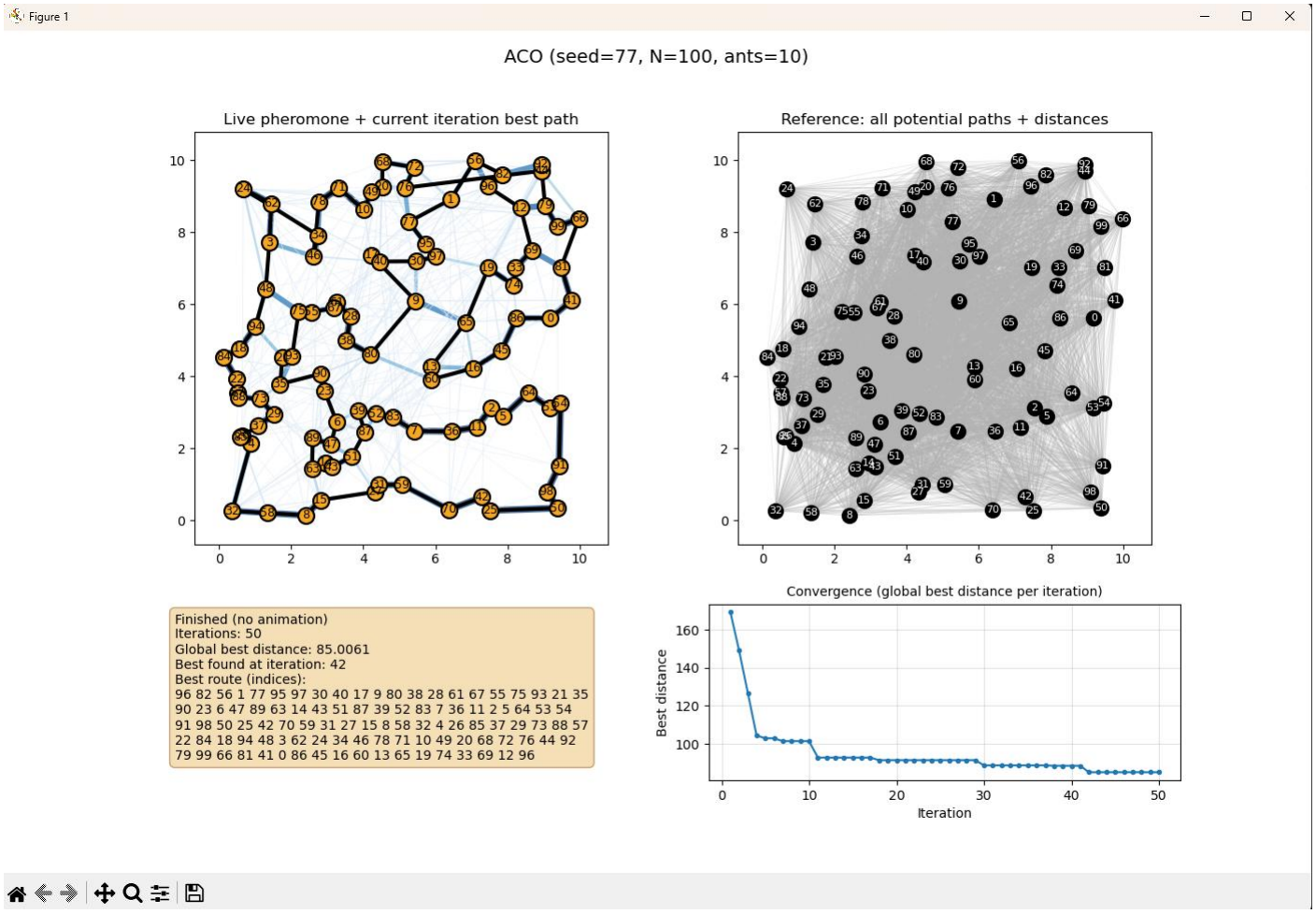
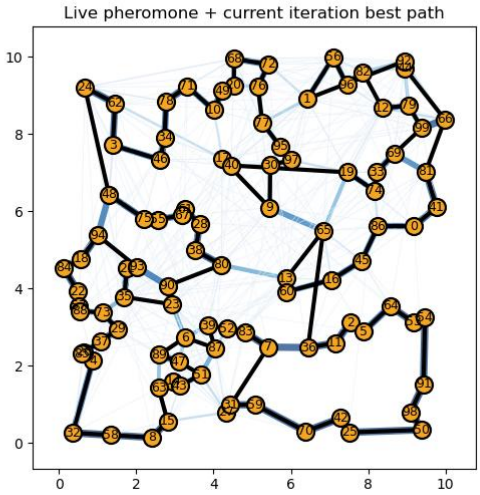
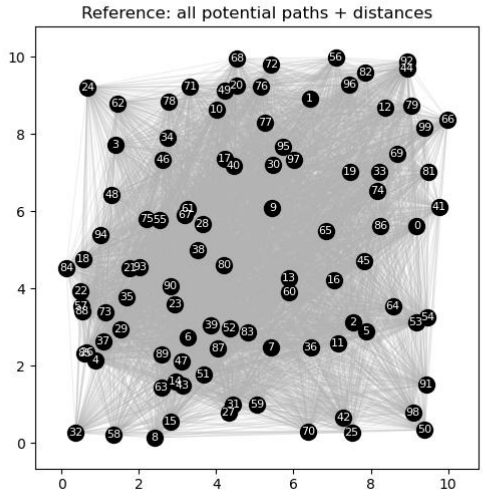


Figure 1

ACO (seed=77, N=100, ants=50)





Finished (no animation)
Iterations: 100
Global best distance: 87.1891
Best found at iteration: 80
Best route (indices):
7 83 52 39 87 6 89 47 51 43 14 63 15 8 58 32 4 85 26 37 29 73 88 57 22 84 18 94 93 21 35 23 90 80 38 28 61 67 55 75 48 24 62 3 46 34 78 71 10 49 20 68 72 76 77 95 97 30 9 17 40 19 74 33 69 99 79 12 82 96 56 1 92 44 66 81 41 0 86 45 16 60 13 65 36 11 2 5 64 53 54 91 98 50 25 42 70 59 31 27 7

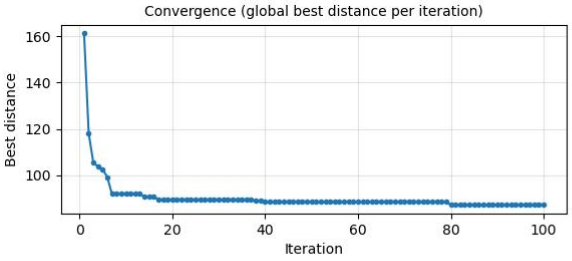
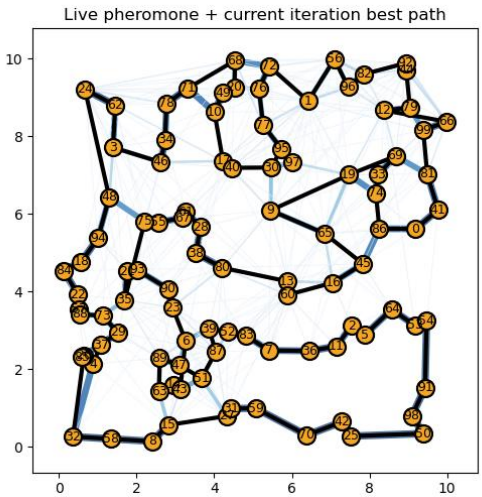
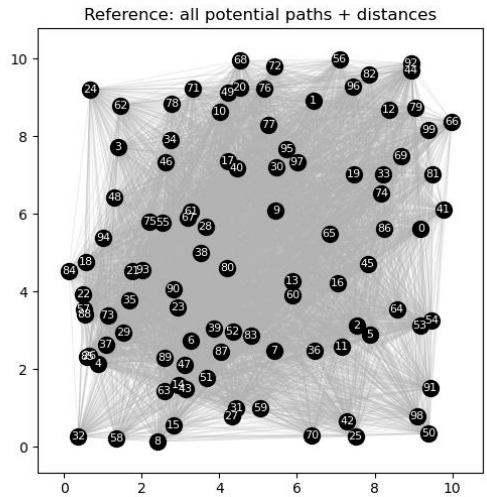


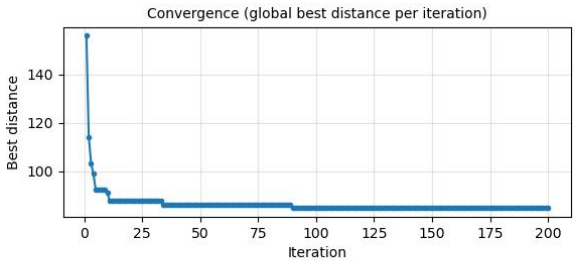
Figure 1

ACO (seed=77, N=100, ants=100)



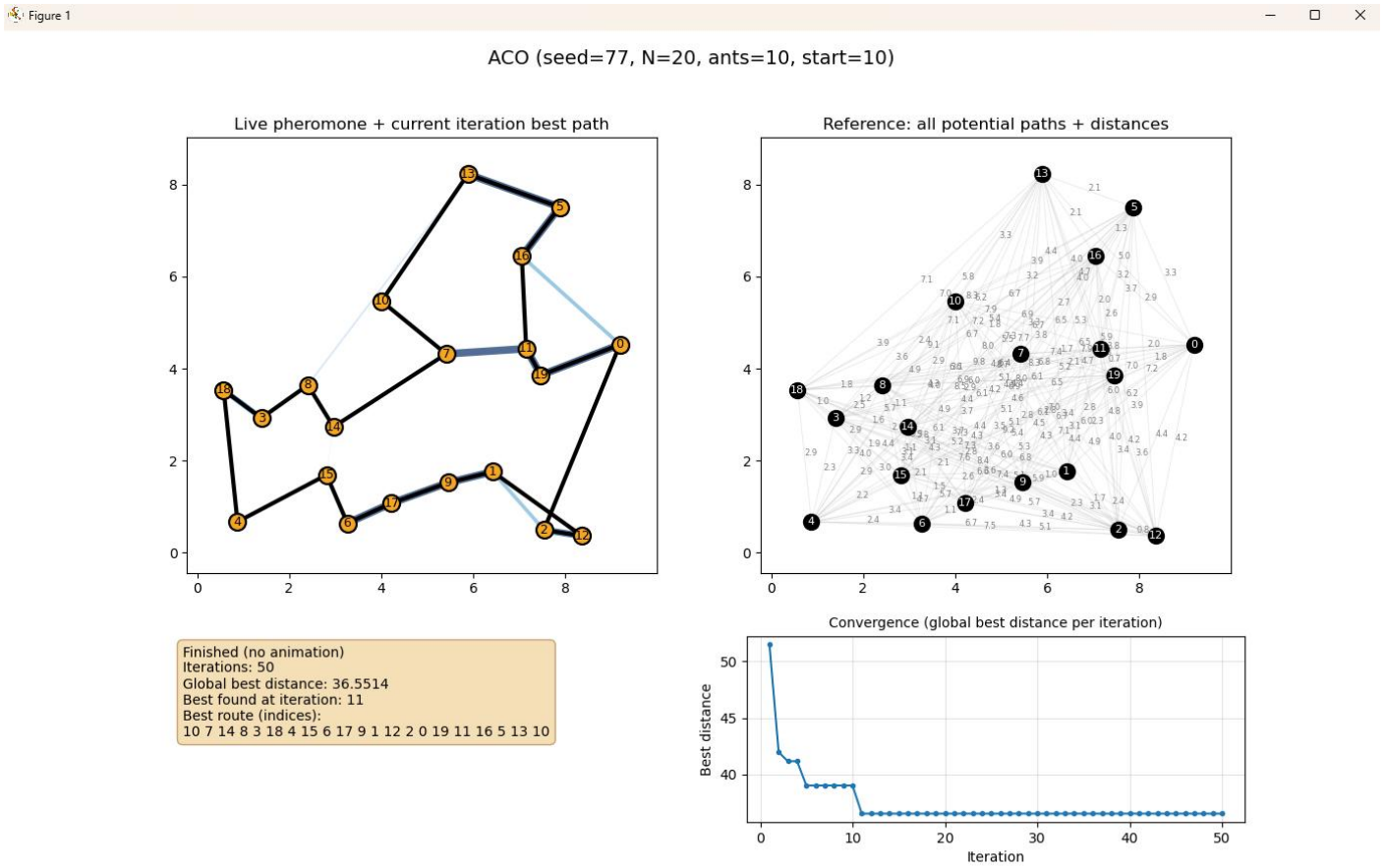


Finished (no animation)
Iterations: 200
Global best distance: 84.7460
Best found at iteration: 90
Best route (indices):
69 33 74 86 0 41 81 99 66 12 79 44 92 82 96 56 1 72 76 77 97 95 30 40 17 10 49 20 68 71 78 34 46 3 62 24 48 94 18 84 22 57 88 73 29 37 4 85 26 32 58 8 15 27 31 59 70 42 25 50 98 91 54 53 64 5 2 11 36 7 83 52 39 87 51 47 89 63 14 43 6 23 90 93 21 35 75 55 67 61 28 38 80 13 60 16 45 65 9 19 69



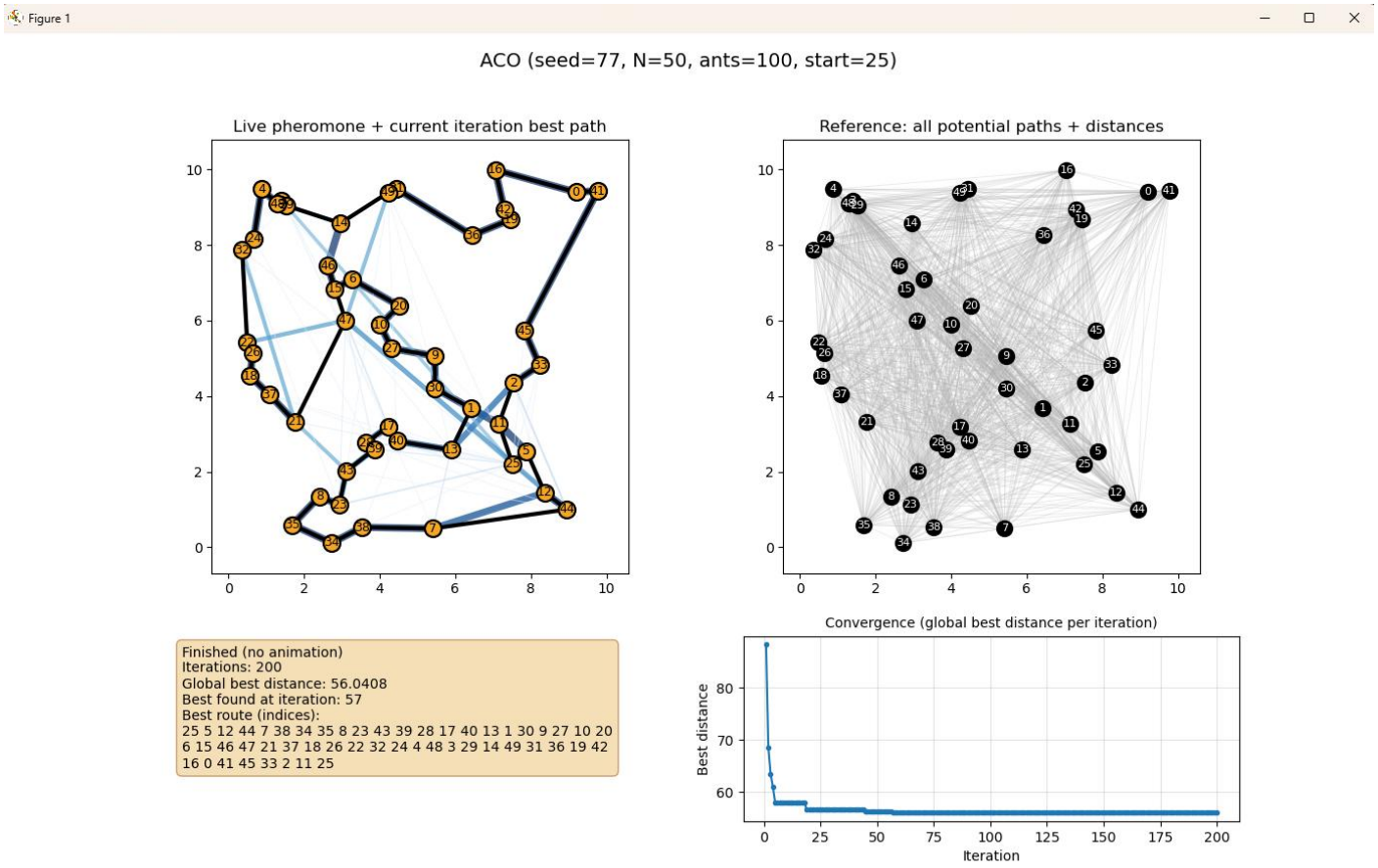
Iteration	Швидкий (ants = 10, iterations = 50)		Збалансований (ants = 50, iterations = 100)		Насичений (ants = 100, iterations = 200)	
	Best	b-step	Best	b-step	Best	b-step
1	85.0061	42	87.1891	80	84.746	90
2	88.0878	29	87.1918	100	84.8942	80
3	89.8118	33	87.7091	39	84.9061	195
Avg	87.6352	35	87.3633	73	84.8488	122

N = 20, start = 10:



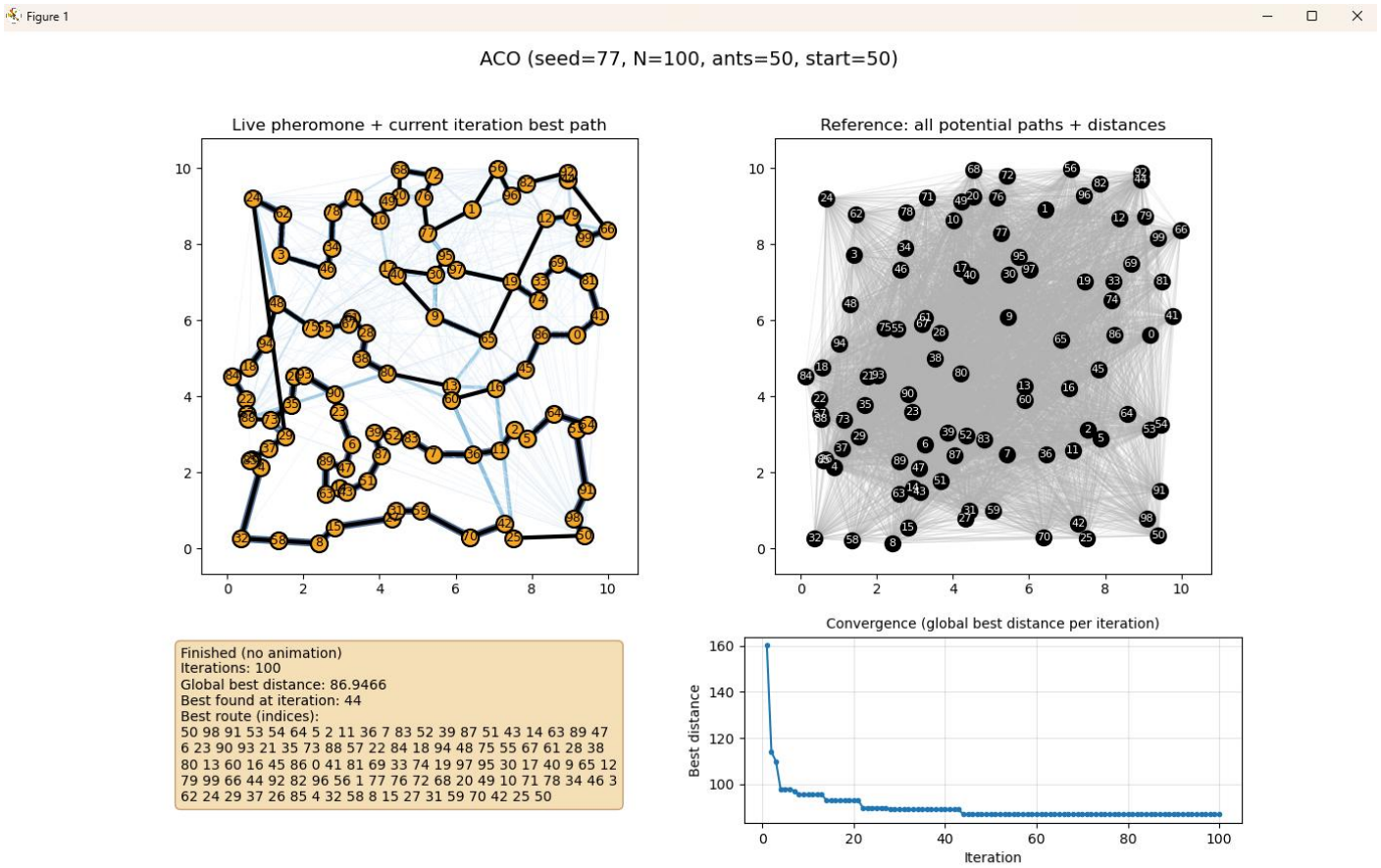
Iteration	Швидкий (ants = 10, iterations = 50)		Збалансований (ants = 10, iterations = 100)		Насичений (ants = 100, iterations = 200)	
	Best	b-step	Best	b-step	Best	b-step
1	36.5514	11	37.2252	53	35.7301	49
2	38.6315	26	36.6179	66	35.7301	23
3	36.6179	44	36.6179	46	35.7301	37
Avg	37.2669	27	36.8203	55	35.7301	36

N = 50, start = 25:



Iteration	Швидкий (ants = 10, iterations = 50)		Збалансований (ants = 25, iterations = 100)		Насичений (ants = 100, iterations = 200)	
	Best	b-step	Best	b-step	Best	b-step
1	57.2531	20	56.3117	31	56.0408	57
2	61.7583	33	56.3117	26	56.5738	47
3	59.1938	31	57.002	84	56.352	98
Avg	59.4017	28	56.5418	47	56.3222	67

N = 100, start = 50:



Iteration	Швидкий (ants = 10, iterations = 50)		Збалансований (ants = 50, iterations = 100)		Насичений (ants = 100, iterations = 200)	
	Best	b-step	Best	b-step	Best	b-step
1	91.3628	42	86.9466	44	86.451	106
2	90.4863	49	88.3772	75	87.1941	46
3	91.8219	39	88.7249	38	87.3255	73
Avg	91.2237	43	88.0162	52	86.9902	75

N	start_point	Сценарій	Ants	Iterations	Best (AVG)	b-step (AVG)
20	None	Швидкий	10	50	36.0185	30
20	None	Збалансований	10	100	35.9577	32
20	None	Насичений	100	200	35.7301	48
20	10	Швидкий	10	50	37.2669	27
20	10	Збалансований	10	100	36.8203	55
20	10	Насичений	100	200	35.7301	63
50	None	Швидкий	10	50	56.7832	42

50	None	Збалансований	25	100	55.9118	65
50	None	Насичений	100	200	55.4475	109
50	25	Швидкий	10	50	59.4017	28
50	25	Збалансований	25	100	56.5418	36
50	25	Насичений	100	200	56.3222	82
100	None	Швидкий	10	50	87.6532	35
100	None	Збалансований	50	100	87.3633	73
100	None	Насичений	100	200	84.8488	122
100	50	Швидкий	10	50	91.2237	43
100	50	Збалансований	50	100	88.0612	52
100	50	Насичений	100	200	86.9902	75

Порівняння якості

За результатом, при **N=20** отримано наступні середні найкращі довжини (менше – краще): Швидкий: 36.02; Збалансований: 35.96; Насичений: 35.73.

Приріст якості невеликий (задача мала): Насичений кращий за Швидкий приблизно на 0.8%, за Збалансований – на 0.6%.

Висновок: для малих N є границя якості; додаткові мурахи/ітерації майже не покращують результат.

При **N=50** середні найкращі довжини: Швидкий: 56.78; Збалансований: 55.91; Насичений: 55.45.

Збалансований покращує Швидкий на ±1.5%; Насичений додає ще ±0.8% (разом 2.3% від Швидкого).

Висновок: чітко видно «коліно» кривої – Збалансований дає кращий результат за помірний час.

При **N=100** середні найкращі довжини: 87.65; 87.36; 84.95.

Покращення Збалансованого відносно Швидкого невелике (±0.3%), зате Насичений додає ще ±2.8% (разом 3.1% проти Швидкого).

Висновок: на великих мапах додаткові ресурси відчутно покращують якість; Насичений має сенс, якщо важлива найкраща довжина. При більшій кількості ресурсів алгоритм довше досліджує, а прориви трапляються пізніше – це добре для великих задач, де небажана рання «жадібна» конвергенція.

Вплив фіксованої стартової вершини (порівняння середніх найкращих довжин при start=None vs start=fixed):

N=20:

- Швидкий погіршився з 36.02 до 37.27 (+3.5%);
- Збалансований – до 36.82 (+2.3%);
- Насичений – практично без змін (~35.73).

N=50:

- Швидкий +4.6% (до 59.40);
- Збалансований +1.1% (до 56.54);
- Насичений +1.6% (до 56.32).

N=100:

- Швидкий +4.1% (до 91.22);
- Збалансований +0.8% (до 88.06);
- Насичений +2.4% (до 86.99).

Висновок: фіксація старту зменшує різноманітність початкових турів і, як правило, погіршує якість (особливо – при малій кількості мурах). Для відтворюваності експериментів старт фіксувати можна, але для якості краще використовувати випадковий старт.

Висновки: за результатом виконання лабораторної роботи створено інтелектуальну інформаційну систему на основі мурашиного алгоритму для розв'язування задач комбінаторної оптимізації. Основні висновки за результатом дослідження:

- Якість монотонно покращується зі збільшенням кількості мурах та ітерацій; ефект найбільший на великих картах (N=100);
- b-step зсувається праворуч при зростанні ресурсів;
- Фіксований старт у середньому погіршує результат (до 5%); краще використовувати випадкові старти;
- Найкращий компроміс для більшості випадків – Збалансований (мурахи = N/2, 100 ітерацій): дає більшість виграшу за помірний час;
- За потреби найкращого результату на N≥60 – Насичений (до 100 мурах, 200 ітерацій), бажано з ранньою зупинкою.

Ці висновки узгоджуються з теорією ACO: збільшення кількості агентів і горизонту ітерацій підсилює накопичення корисного феромону (хоч і з пізнішими проривами); однак надмірне використання ресурсів дає зменшувану віддачу на одиницю ресурсу.

Вихідний код застосунку можна знайти за наступним посиланням на GitHub.

ДОДАТОК А

Вихідний код алгоритму

```
class ACO:
    def __init__(self, alpha=1.0, beta=2.0, evaporation=0.5, Q=100.0):
        self.alpha = alpha
        self.beta = beta
        self.evaporation = evaporation
        self.Q = Q

        self.points = None
        self.distances = None
        self.pheromones = None

        self.all_best_paths = []
        self.best_len_history = []
        self.best_iter_found = None

    def init_map(self, n_points, map_seed=None, x_range=(0, 10), y_range=(0, 10)):
        if map_seed is not None:
            np.random.seed(int(map_seed))
        else:
            np.random.seed(None)

        xs = np.random.uniform(x_range[0], x_range[1], n_points)
        ys = np.random.uniform(y_range[0], y_range[1], n_points)
        self.points = np.column_stack((xs, ys))
        self.distances = self._compute_distance_matrix(self.points)
        self.pheromones = np.ones((n_points, n_points))

        # reset history
        self.all_best_paths = []
        self.best_len_history = []
        self.best_iter_found = None
        np.random.seed(None)

    @staticmethod
    def _compute_distance_matrix(points):
        n = len(points)
        mat = np.zeros((n, n))
        for i in range(n):
            for j in range(i + 1, n):
                d = np.linalg.norm(points[i] - points[j])
                mat[i, j] = mat[j, i] = d
        return mat

    def _move_agent(self, visited, current):
        unvisited = np.where(~visited)[0]
        taus = self.pheromones[current, unvisited] ** self.alpha
        etas = 1.0 / (self.distances[current, unvisited] + 1e-12)
        probs = taus * (etas ** self.beta)
        s = probs.sum()
        if s <= 0 or np.isnan(s):
            return np.random.choice(unvisited)
        probs = probs / s
        return np.random.choice(unvisited, p=probs)

    def iteration(self, n_ants=10, start_point=None):
        n = len(self.points)
        paths = []
        lengths = []
```

```
        for _ in range(n_ants):
            visited = np.zeros(n, dtype=bool)
            current = np.random.randint(n) if start_point is None else
int(start_point)
            visited[current] = True
            path = [current]
            total = 0.0

            while not np.all(visited):
                nxt = self._move_agent(visited, current)
                total += self.distances[current, nxt]
                path.append(nxt)
                visited[nxt] = True
                current = nxt

            total += self.distances[path[-1], path[0]]  # close tour
            paths.append(path)
            lengths.append(total)

        # evaporate
        self.pheromones *= self.evaporation

        # deposit
        for path, L in zip(paths, lengths):
            deposit = self.Q / (L + 1e-12)
            for i in range(len(path) - 1):
                a, b = path[i], path[i + 1]
                self.pheromones[a, b] += deposit
                self.pheromones[b, a] += deposit
            a, b = path[-1], path[0]
            self.pheromones[a, b] += deposit
            self.pheromones[b, a] += deposit

        # get iteration's best
        idx_best = int(np.argmin(lengths))
        iter_best_len = lengths[idx_best]
        iter_best_path = paths[idx_best]

        # determine global best
        global_best_len = self.best_len_history[-1] if self.best_len_history else
np.inf
        if iter_best_len < global_best_len:
            # new global best
            self.all_best_paths.append(iter_best_path)
            self.best_len_history.append(iter_best_len)
            self.best_iter_found = len(self.best_len_history)  # 1-based
        else:
            # keep previous best
            if self.all_best_paths:
                self.all_best_paths.append(self.all_best_paths[-1])
            else:
                self.all_best_paths.append(iter_best_path)
            self.best_len_history.append(global_best_len)

        return {
            "paths": paths,
            "lengths": lengths,
            "iter_best_path": iter_best_path,
            "iter_best_len": iter_best_len,
            "global_best_len": self.best_len_history[-1],
            "best_iter_found": self.best_iter_found,
```

```
        "pheromones": self.pheromones.copy(),
    }
```

ДОДАТОК Б

Контрольні питання

1. В чому полягає основна ідея методу мурашиних колоній?
   Імітувати колективну поведінку мурах гарантуючи близько оптимальні рішення: багато простих агентів паралельно будують розв'язки, залишають на ребрах феромон, який підсилює хороші компоненти рішень; феромон випаровується, щоб не зациклюватися на випадкових траєкторіях.
2. Для розв'язання яких задач використовується метод мурашиних колоній?
   Переважно для комбінаторної оптимізації: задача комівояжера (TSP), маршрутизація транспортних засобів, розклади, розфарбування графів тощо.
3. В який умовах особливо ефективне використання мурашиних алгоритмів?
   Де потрібно швидко оцінювати часткові рішення (локальні кроки), потрібен паралельний/стохастичний пошук і прийнятна приблизність. Також добре підходить для складних великих задач, де не обов'язково отримувати найкраще рішення (через складність обчислювання), а достатньо наближений результат (NP-повні задачі).
4. Порівняйте метод мурашиних колоній з іншими оптимізаційними методами.
   Мурашині алгоритми гнучкі та здатні знаходити глобальні оптимуми, порівняно з класичними методами (наприклад, градієнтними методами), є менш чутливими до локальних мінімумів.
5. Що являє собою навколишнє середовище в методі мурашиних колоній?
   Це граф задачі (вузли та ребра) з метрикою вартостей (відстані/штрафи) та матриця феромонів, яка зберігає колективну пам'ять про корисні компоненти.
6. З якою метою використовується список табу?
   Містить уже відвідані (заборонені) вершини/ресурси, щоб уникнути повторень і забезпечити побудову коректного часткового рішення (наприклад, у TSP – кожне місто відвідується один раз).
7. Яким чином розташовуються вузли в списку поточної подорожі?
   Вузли в списку поточної подорожі розташовуються залежно від феромонів та ймовірностей вибору (та які ще не в списку табу) в кожному етапі пошуку шляху.
8. Що таке феромон? Яке його призначення в методі мурашиних колоній?
   Феромон – це умовний «сигнал», який агенти залишають на ребрах чи ділянках маршруту. У ACO він відображає корисність шляху: що кращий маршрут, то більше феромону на ньому, і тим вища ймовірність, що інші агенти підуть цим же шляхом у майбутньому. З часом феромон слабшає, щоб система не застрягала на випадкових рішеннях.
9. Проаналізуйте послідовність виконання методу мурашиних колоній.
   Послідовність виконання методу:
   a. вирівнюємо початковий рівень феромону і задаємо параметри;
   b. кожна мураха крок за кроком будує повний розв'язок, обираючи наступну вершину з урахуванням феромону і близькості;
   c. після побудови турів оцінюємо їх довжини, оновлюємо глобально найкращий маршрут і запам'ятовуємо ітерацію його першої появи;

d. “випаровуємо” феромон, зменшуючи його рівень всюди;
e. “відкладаємо” новий феромон на ребрах кращих маршрутів, підсилюючи їх;
f. повторюємо кроки будування, оцінки й оновлення феромону до вичерпання ітерацій або до критерію зупинки.

10. Як розраховується кількість феромону, що було залишено на кожній грані шляху і-го агенту?

Кількість феромону, що залишає агент на кожному ребрі свого маршруту, визначається якістю його маршруту: що коротший або кращий тур, то більший внесок феромону він робить на всіх ребрах, якими пройшов. Таким чином хороші рішення отримують відчутніше “підсилення”.

11. Поясніть особливості модифікації методу мурашиних колоній при вирішенні таких задач комбінаторної оптимізації, як задача комівояжера, задача оптимізації маршрутів вантажівок, задача розфарбування графа, квадратична задача про призначення, задача оптимізації сіткових графіків, задача календарного планування.

Особливості адаптації методу під різні задачі:

a. Задача комівояжера: вершини – міста, ребра – відстані між ними. Мурахи будують замкнений тур, відвідуючи кожне місто рівно один раз. Табу-список забороняє повертатись у вже відвідані міста;
b. Оптимізація маршрутів вантажівок: є кілька маршрутів і обмеження на місткість та час. Дозволені кроки враховують завантаження авто і часові вікна клієнтів. Феромон підсилює вдалі вставки з урахуванням логістичних обмежень. Поділ простору на підзадачі: простір маршруту може бути поділений на кілька підзадач, кожна з яких відповідає за один вантажівку (мурахи шукають оптимальні маршрути для кожної вантажівки, враховуючи кількість вантажів та обмеження по часі або навантаженню);
c. Розфарбування графа: кожній вершині призначаємо колір так, щоб сусідні вершини мали різні кольори. Компоненти рішення – пари “вершина-колір”, а феромон підказує, які призначення частіше приводили до коректного мінімального набору кольорів;
d. Квадратична задача призначення: потрібно зіставити об’єкти з місцями розміщення з урахуванням парних взаємодій. Мурахи поступово будують відповідності “об’єкт-місце”, фіксуючи найперспективніші варіанти феромоном. Феромони оновлюються таким чином, щоб найкращі призначення отримували більше феромонів, підвищуючи ймовірність вибору цих варіантів іншими мурахами;
e. Оптимізація сіткових графіків: планування робіт із залежностями. Мурахи формують допустимі послідовності робіт, феромон концентрується на критичних ділянках, які скорочують загальний термін (враховуються додаткові обмеження, як час, навантаження чи інші фактори, що впливають на вибір шляху);
f. Календарне планування: формування розкладів на машинах або ресурсах. Кроками є рішення “яку операцію куди і коли ставити”. Феромон накопичується на вдалих розміщеннях (враховуються часові обмеження

для кожного завдання, а також ресурси, доступні для виконання), а прості правила диспетчеризації допомагають швидше будувати хороші розклади. Загальна ідея в усіх випадках одна: агенти крок за кроком будують прийнятне рішення, корисні частини підсилюються феромоном, непотрібні – забуваються, а процес триває, доки не отримаємо якісний результат або досягнуто точку зупинки.