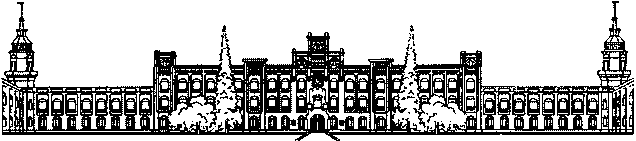
****

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 |  | Перевірив: |
|  |  |  |
| Новиков Д. М. |  | 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).

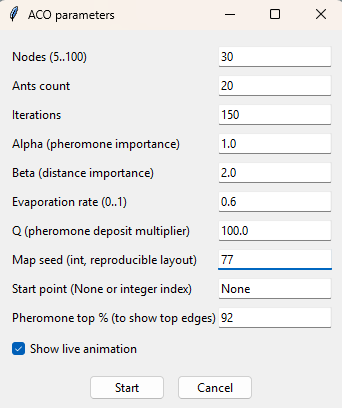


Рис 1 – Приклад ACO parameters

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≤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.

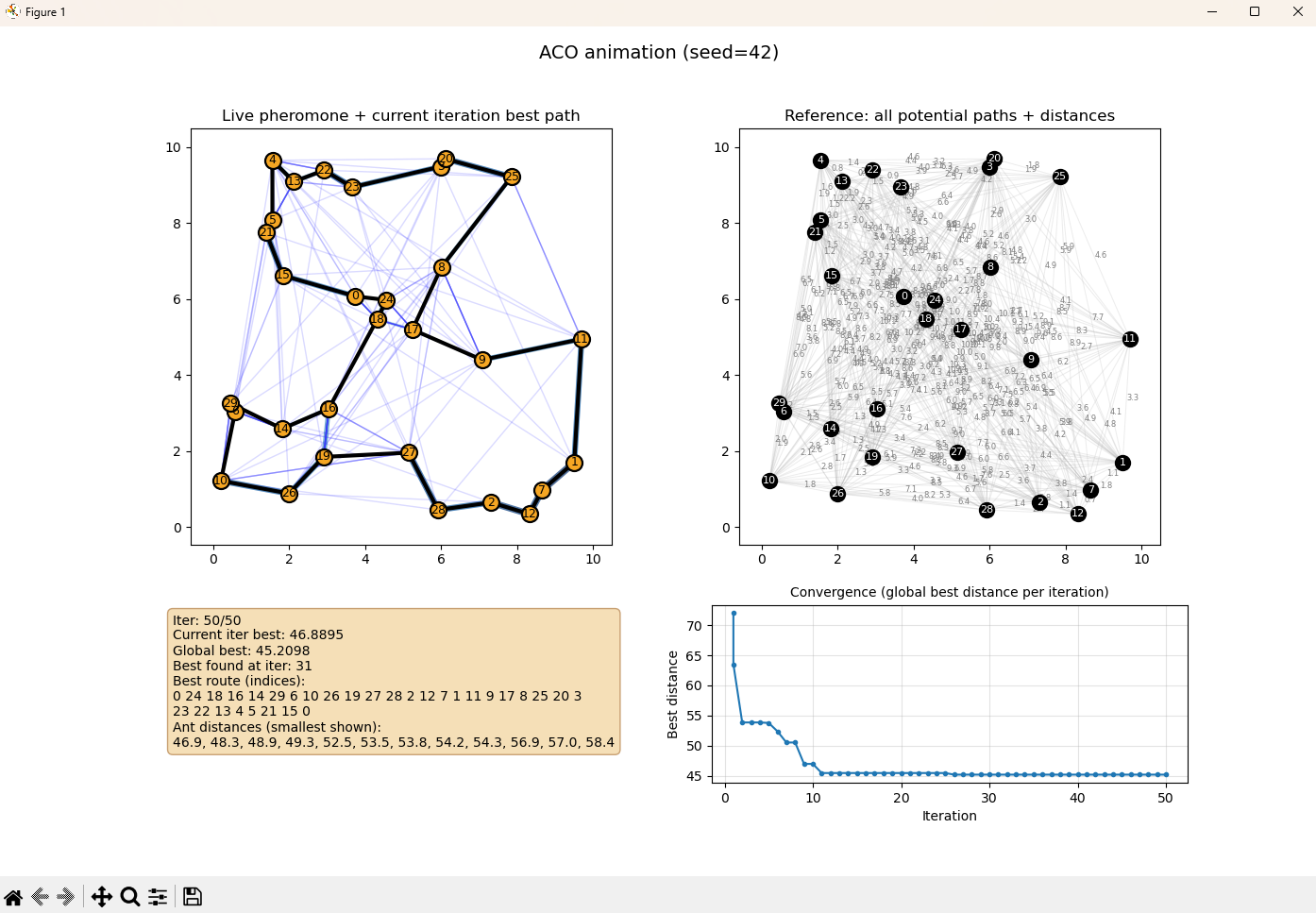


Рис 2 – Приклад основного інтерфейсу програми

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:
   1. start\_point = None (кожна мураха стартує випадково);
   2. 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∈{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).

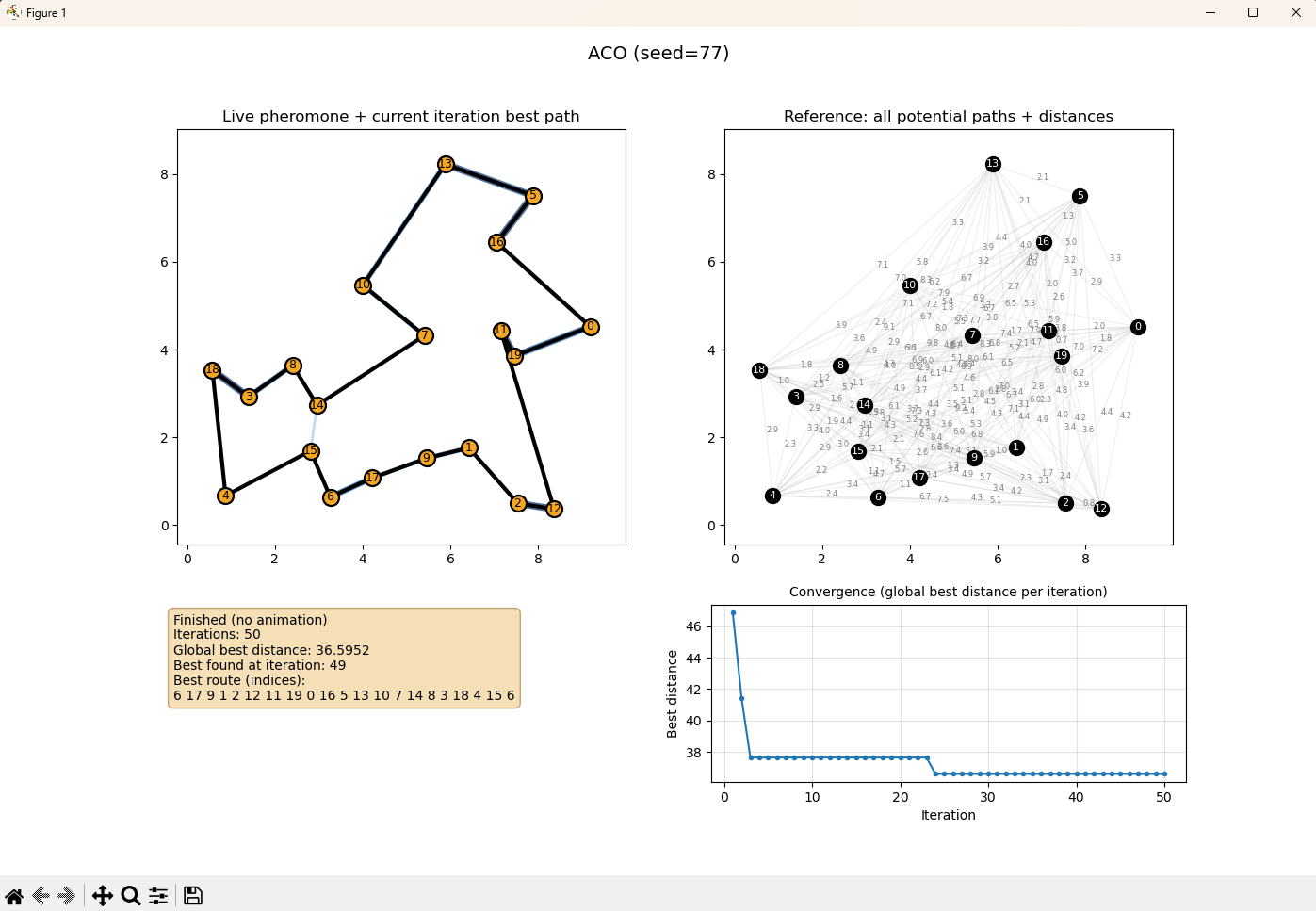
План запусків:

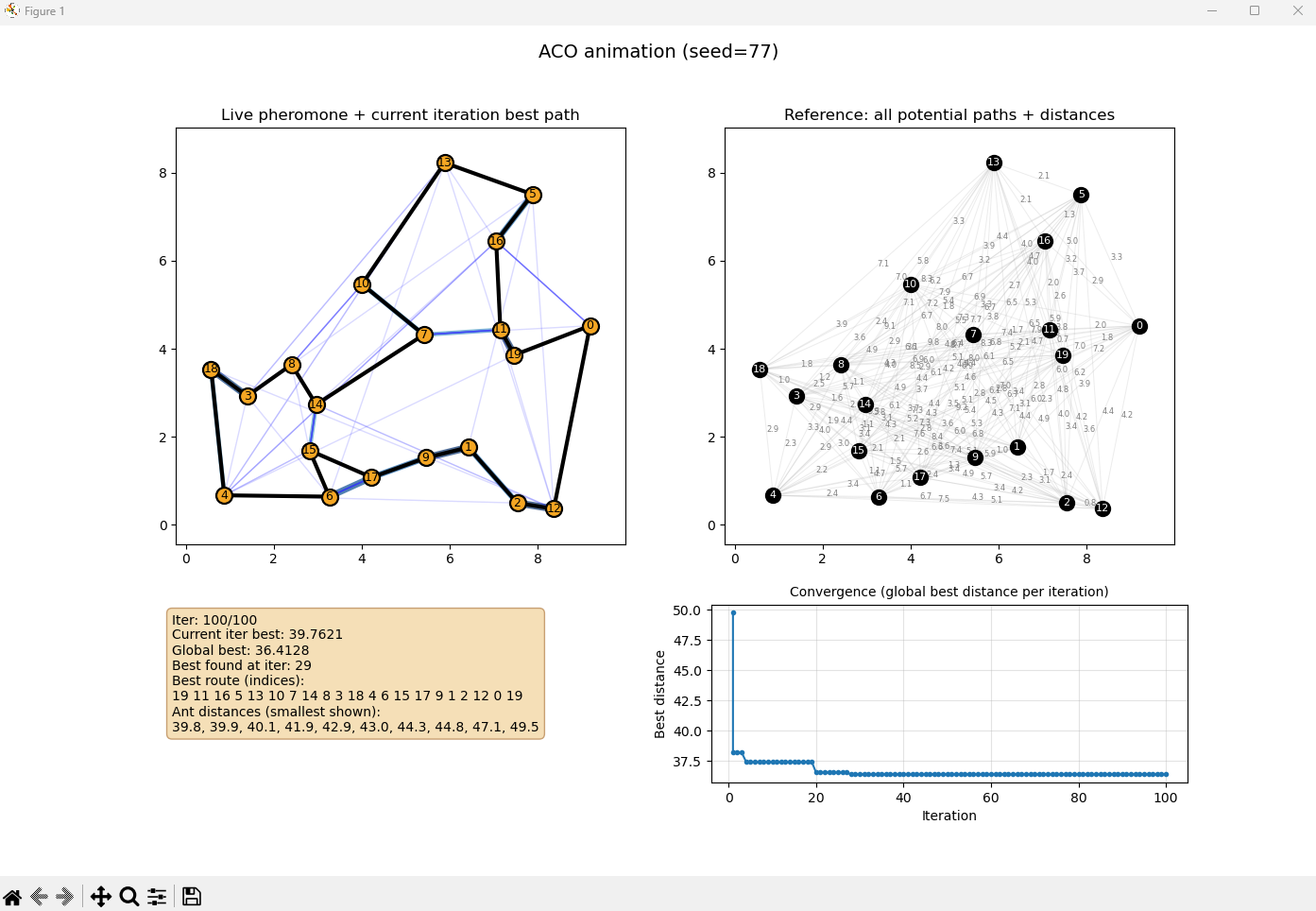
Для кожного N∈{20,50,100} і кожного сценарію:

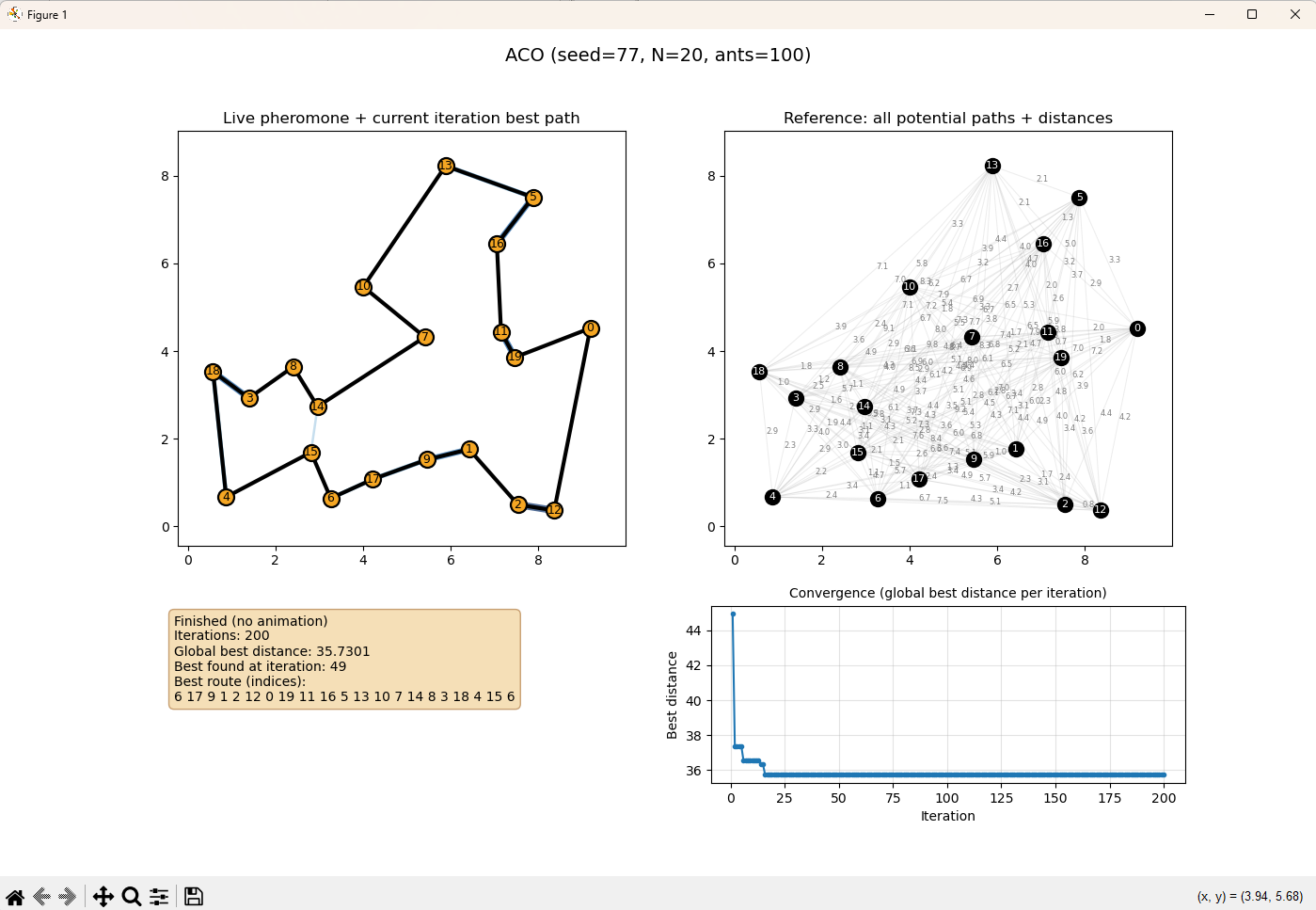
* виконати 3 запусків (як для випадкових стартових точок так і для визначеної);
* зберегти: Best (мін), b-step, Avg (середній Best/b-step);

Приклад роботи:

N = 25:

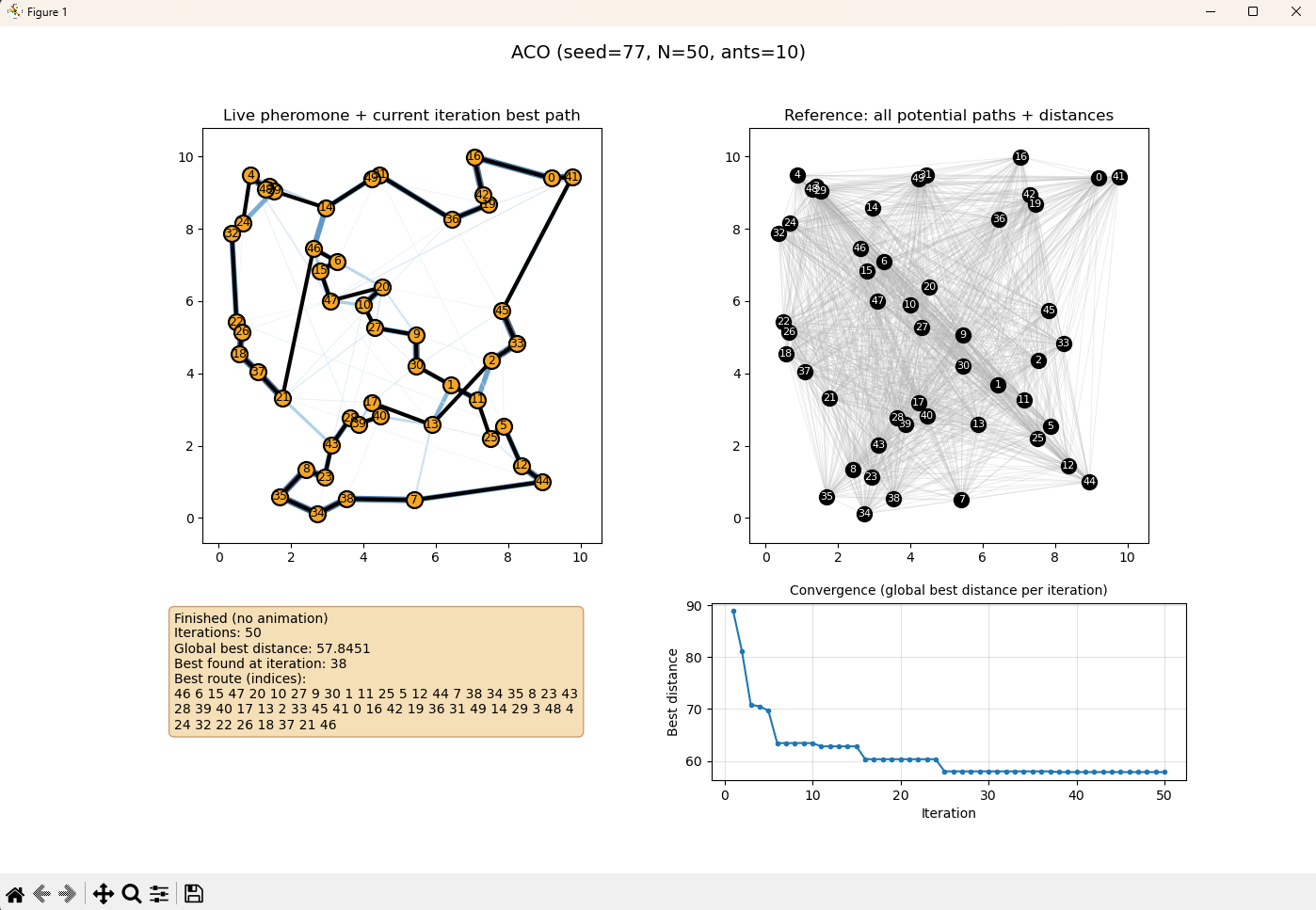


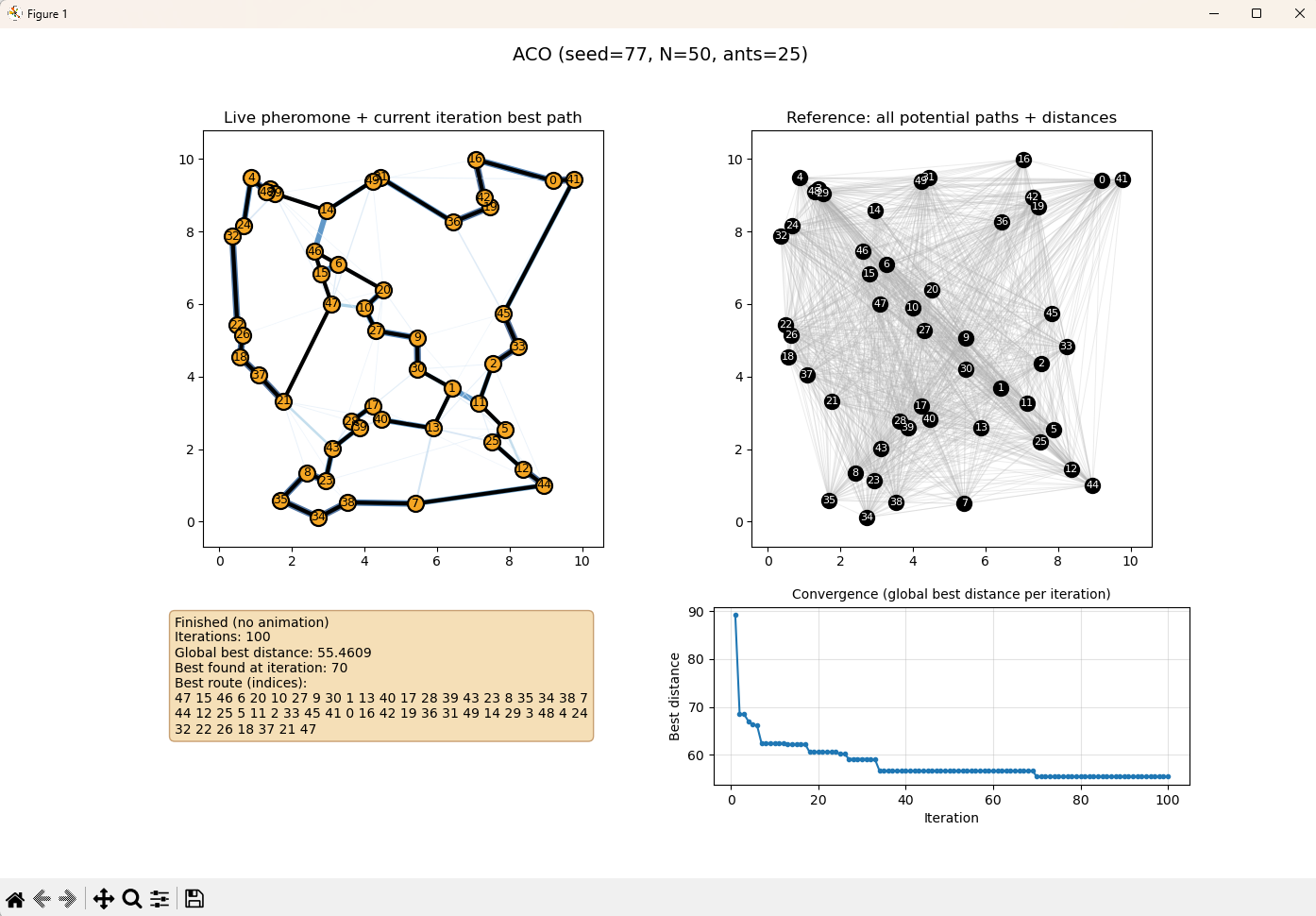


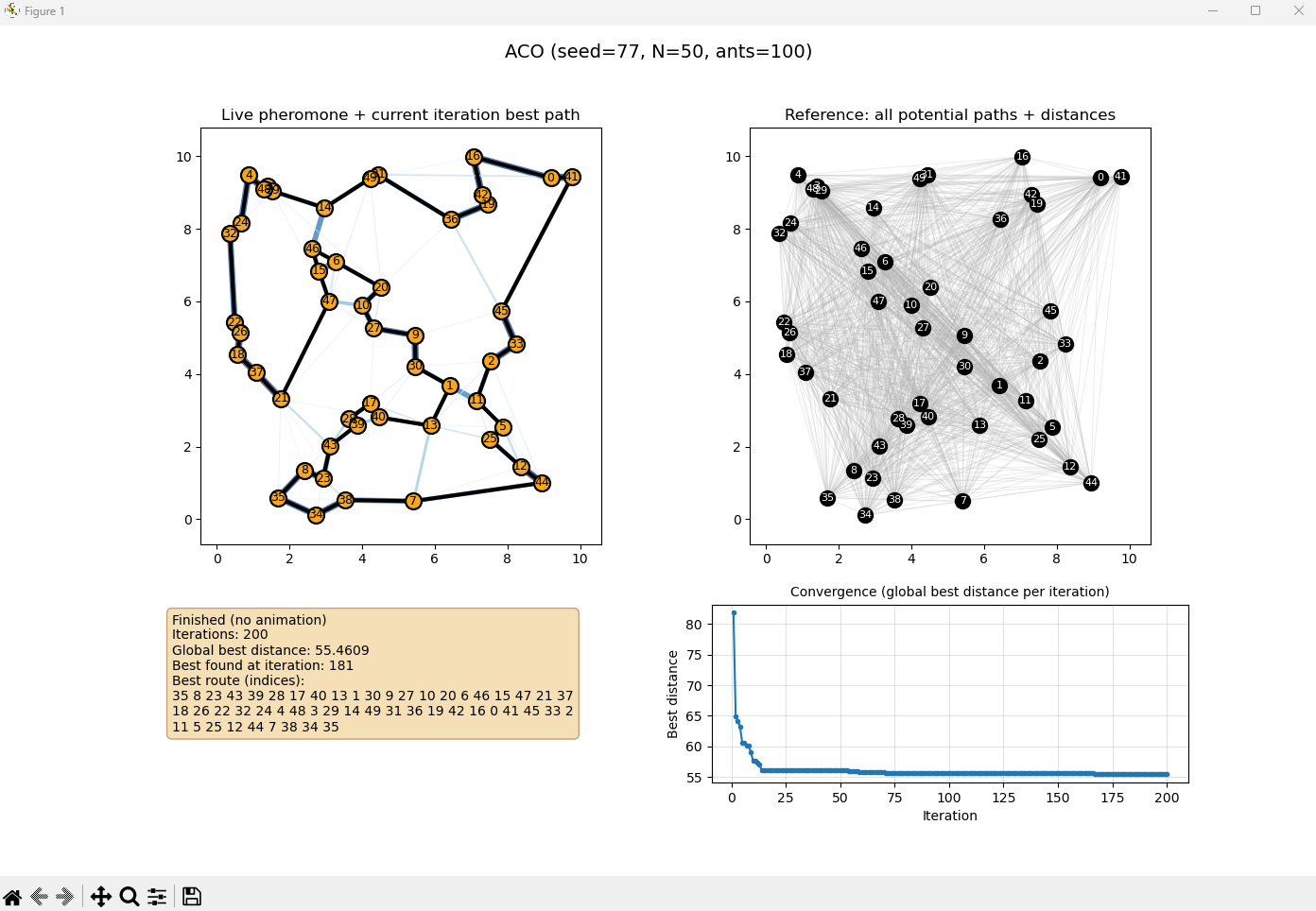


|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Швидкий (ants = 10, iterations = 50) | | Збалансований (ants = 10, iterations = 100) | | Насичений (ants = 100, iterations = 200) | |
| Best | b-step | Best | b-step | Best | b-step |
| 1 | 36.5952 | 49 | 35.7301 | 15 | 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:

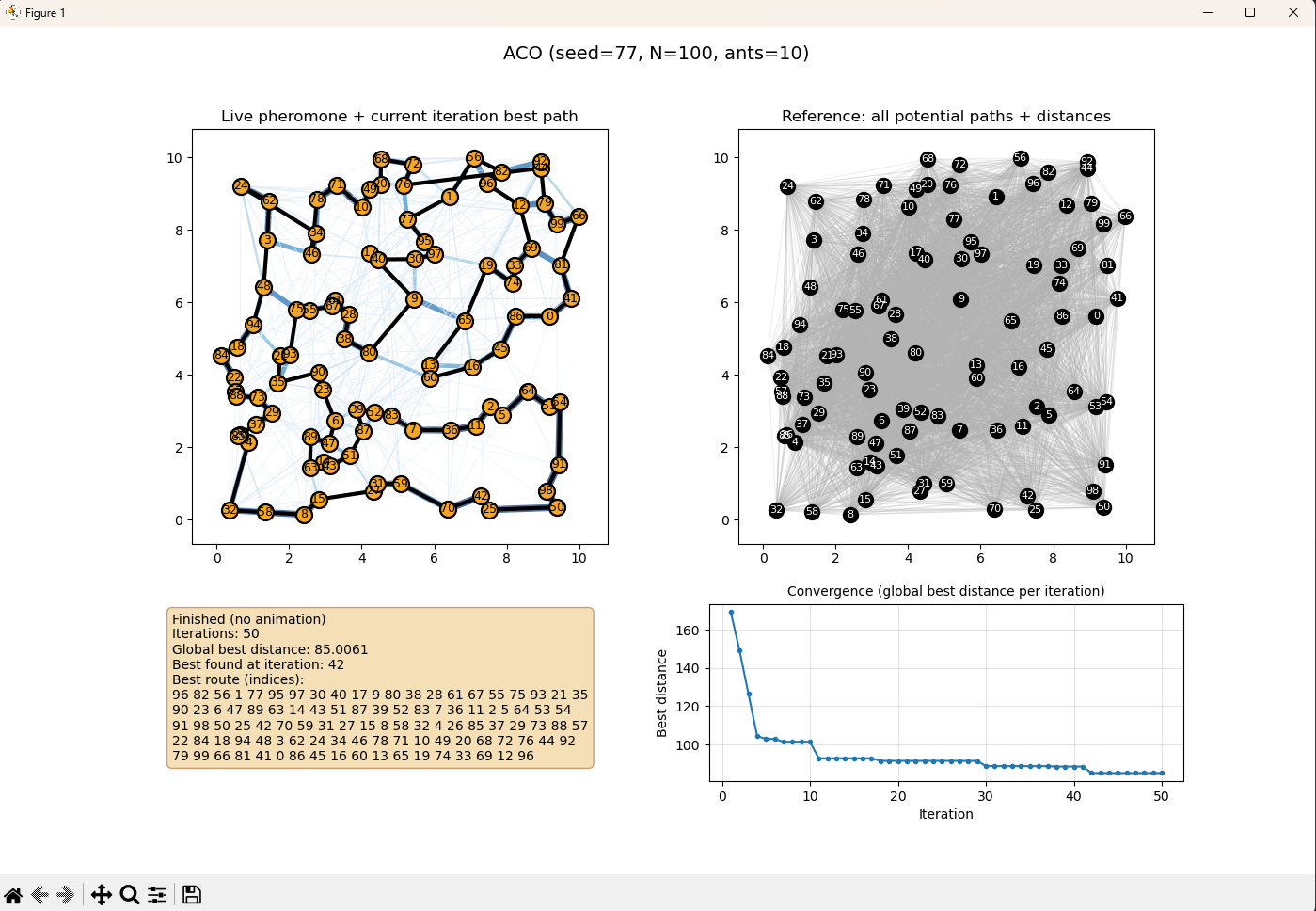


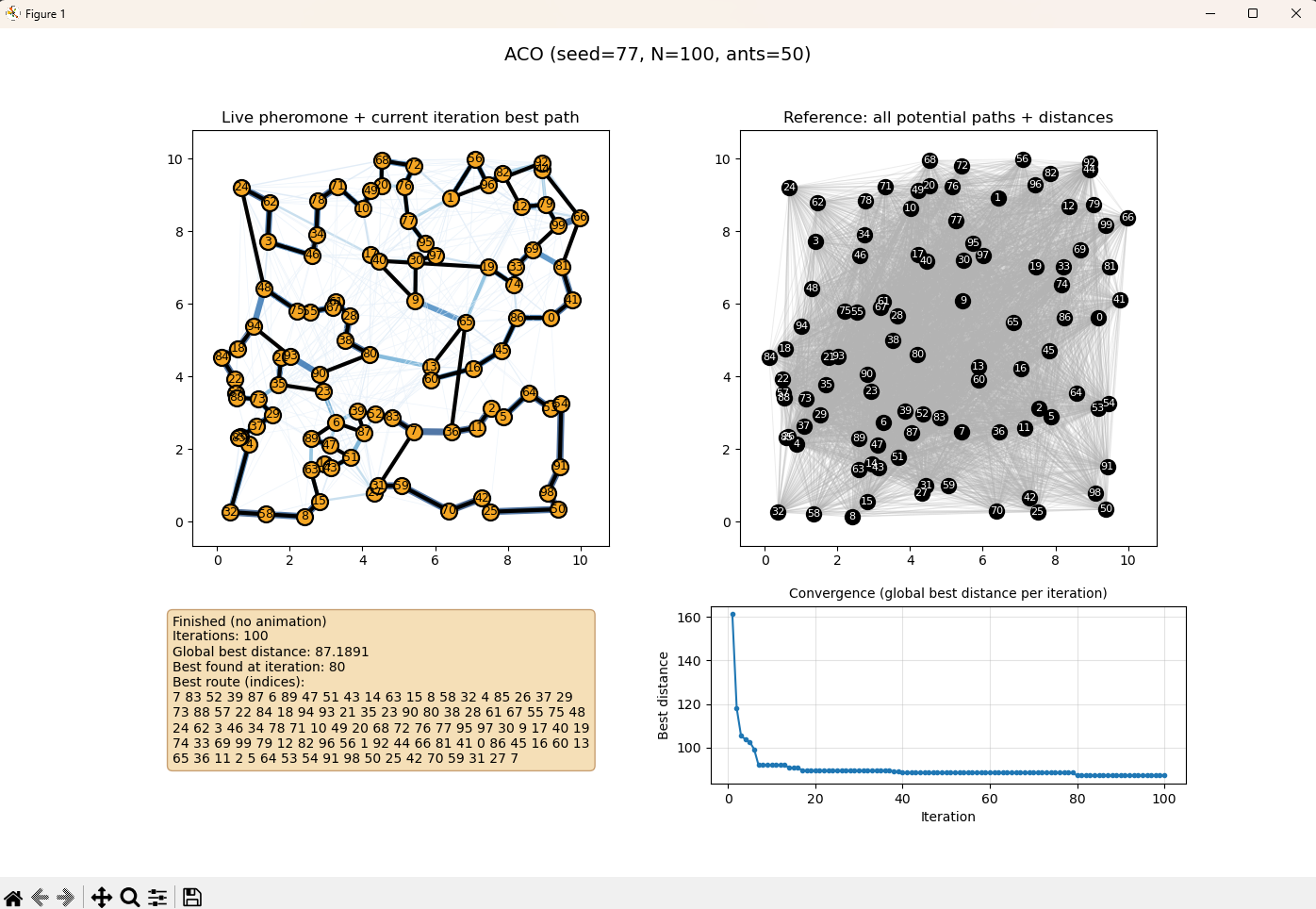


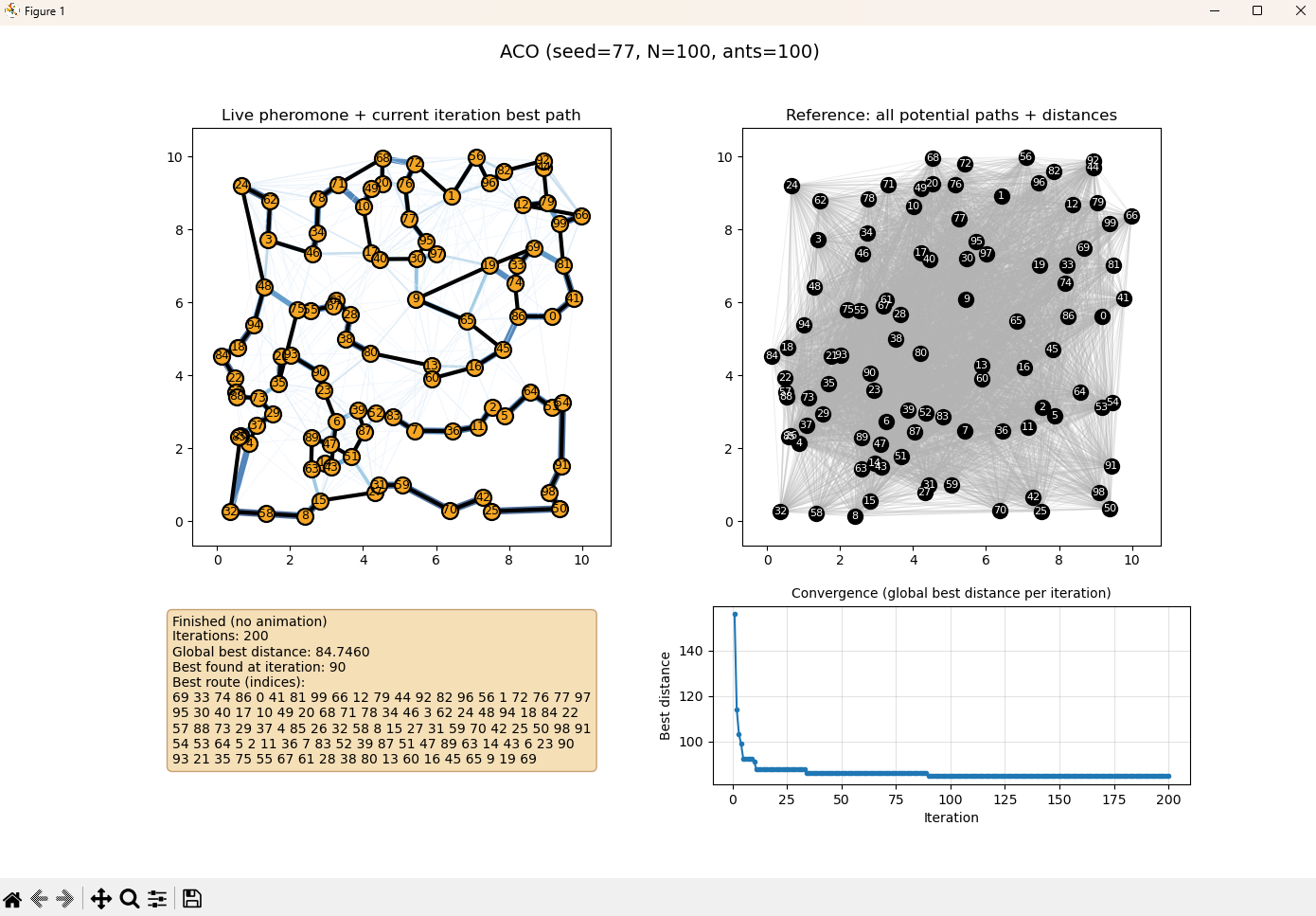


|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Швидкий (ants = 10, iterations = 50) | | Збалансований (ants = 25, iterations = 100) | | Насичений (ants = 100, iterations = 200) | |
| Best | b-step | Best | b-step | Best | b-step |
| 1 | 57.8451 | 38 | 55.4609 | 70 | 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:

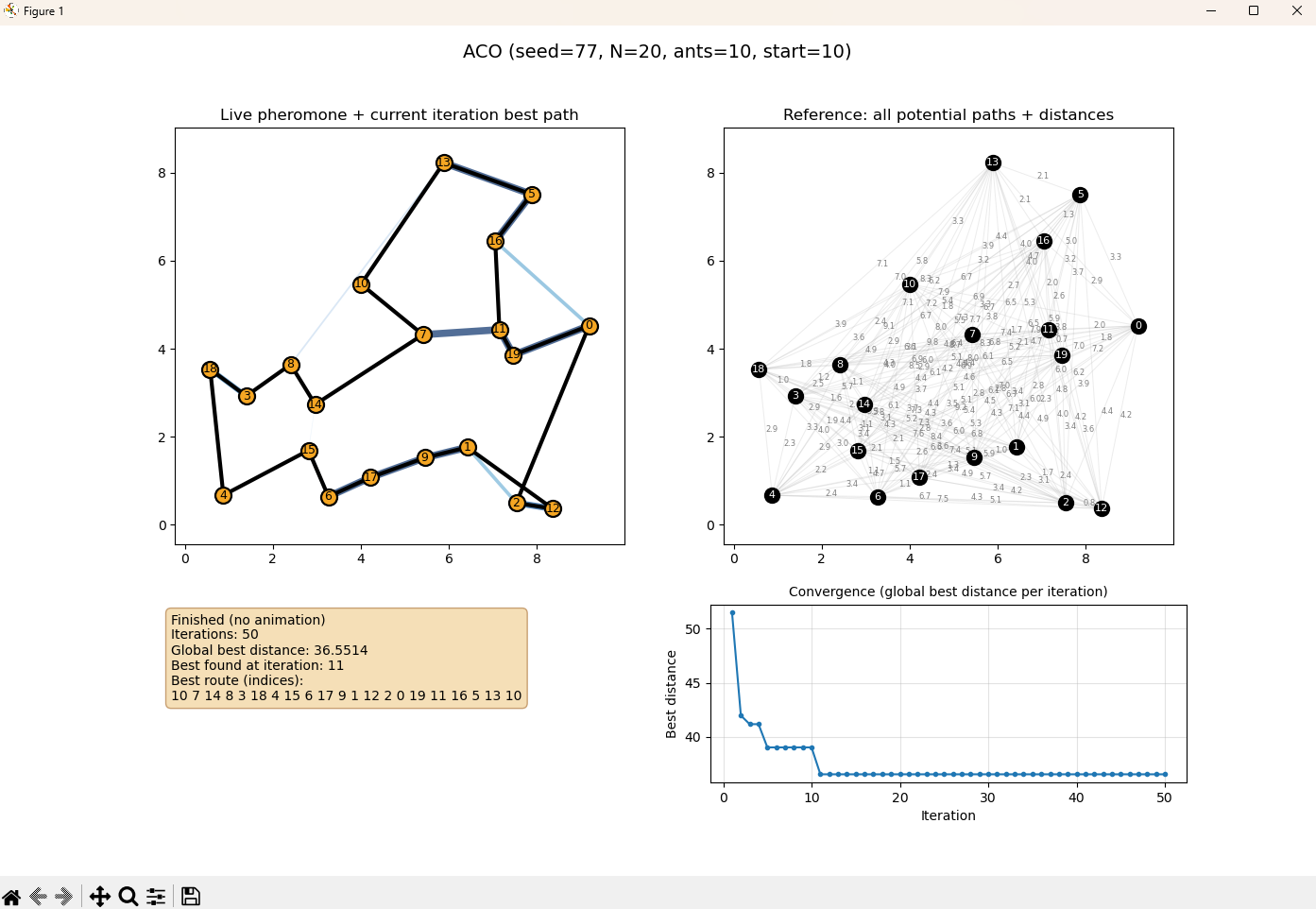






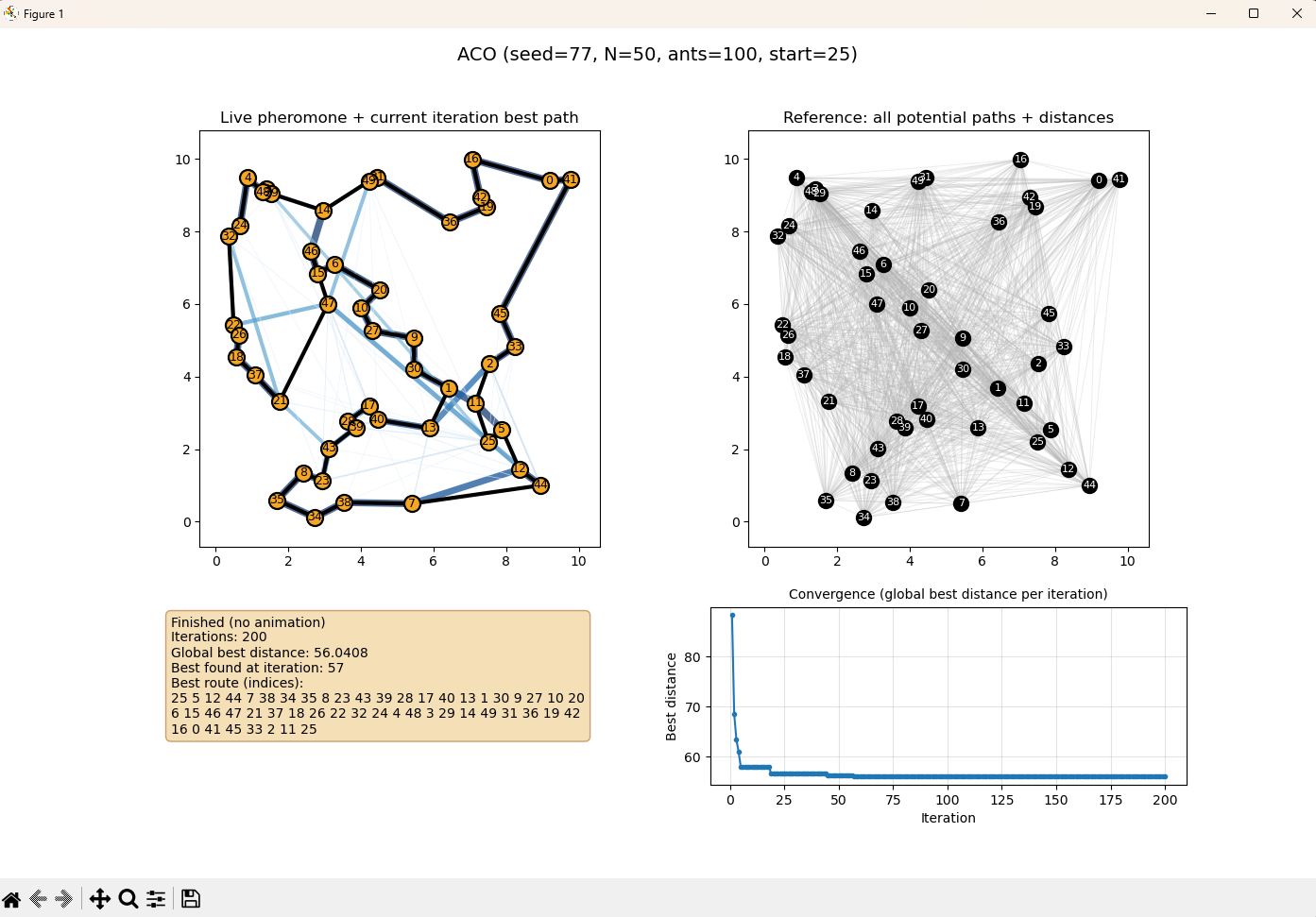
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Швидкий (ants = 10, iterations = 50) | | Збалансований (ants = 50, iterations = 100) | | Насичений (ants = 100, iterations = 200) | |
| Best | b-step | Best | b-step | Best | b-step |
| 1 | 85.0061 | 42 | 87.1891 | 80 | 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:



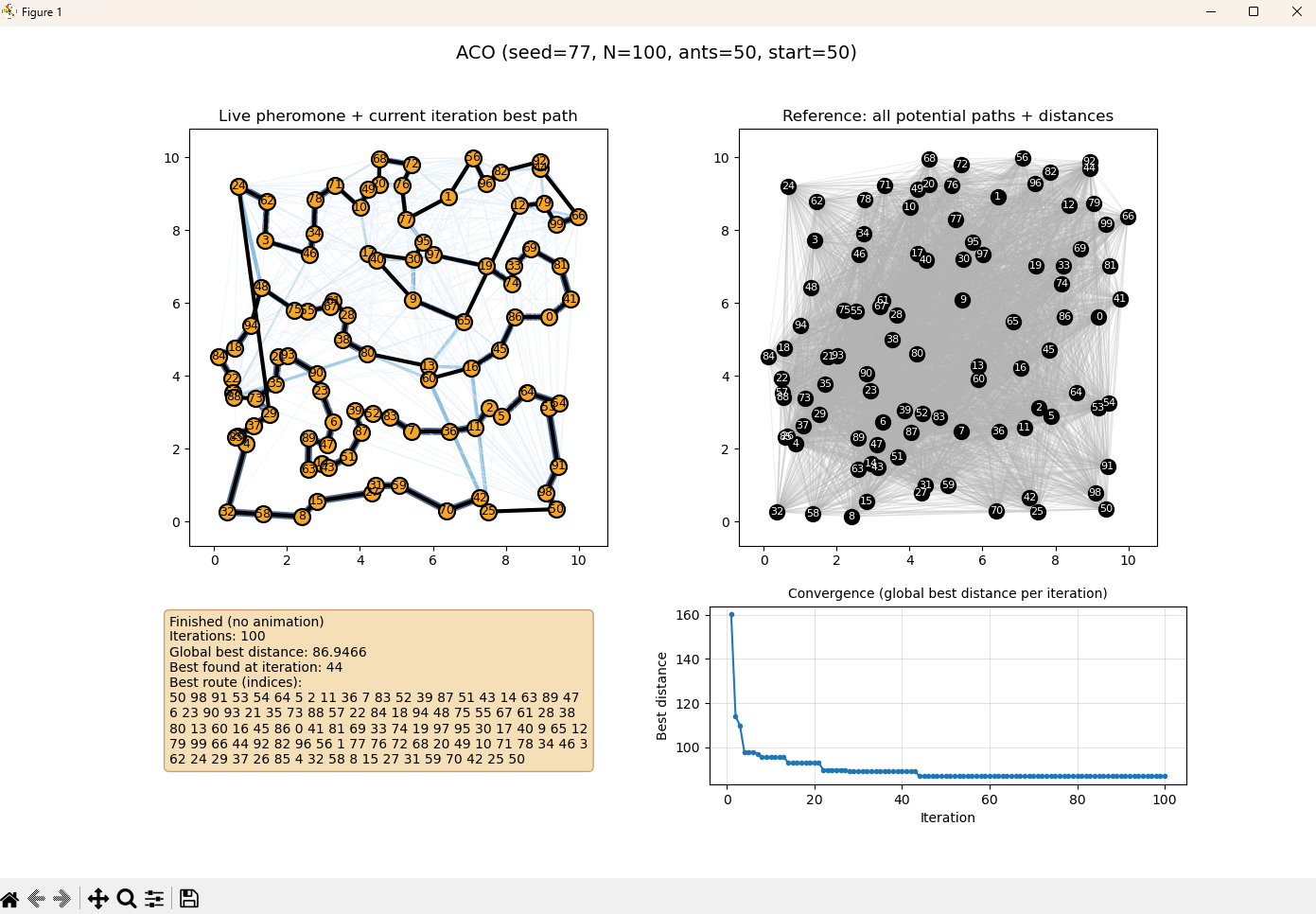
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Швидкий (ants = 10, iterations = 50) | | Збалансований (ants = 10, iterations = 100) | | Насичений (ants = 100, iterations = 200) | |
| Best | b-step | Best | b-step | Best | b-step |
| 1 | 36.5514 | 11 | 37.2252 | 53 | 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](https://github.com/JokerFunny/MCCIIS/blob/main/Lab1/aco_gui.py).