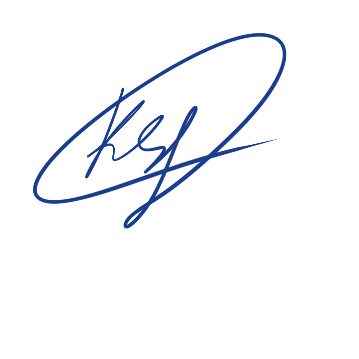
**Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

**Лабораторная работа №3**

Муравьиный алгоритм.

Выполнил:

Студент группы КБ-211

Коренев Д.Н.

Принял:

Твердохлеб В.В

Оглавление

[Задание 3](#_Toc179595081)

[Код алгоритма 6](#_Toc179595082)

[Интерфейс приложения 10](#_Toc179595083)

[Результаты исследования влияния параметров алгоритма 12](#_Toc179595084)

[Вывод 15](#_Toc179595085)

[Приложения 17](#_Toc179595086)

*Цель работы:* Изучение и исследование параметров муравьиного алгоритма на примере решения задачи коммивояжера.

Задание

1. Реализовать windows-приложение, позволяющее решить задачу   
коммивояжера для графа с количеством вершин не менее 20, с использованием   
муравьиного алгоритма. Параметры алгоритма задаются пользователем.

**Блок-схема алгоритма решения задачи комивояжера с использованием муравьиного алгоритма**

Блок-схемы алгоритма представлены на рисунках 1-3.

Изображение выглядит как текст, снимок экрана, Шрифт, графический дизайн

Автоматически созданное описание   
Рисунок 1. Блок-схема решения задачи комивояжера с использованием муравьиного алгоритма.

Изображение выглядит как текст, снимок экрана, Шрифт, Печать

Автоматически созданное описание   
Рисунок 2. Блок-схема процедуры simulateAnts().

Изображение выглядит как текст, снимок экрана, Шрифт, графический дизайн

Автоматически созданное описание   
Рисунок 3. Блок-схема процедуры updateTrails().

Код алгоритма

Полный код программы в приложении 1. Код реализации алгоритма:

Python

class City:

    def \_\_init\_\_(self, x=0, y=0):

        self.x = x

        self.y = y

class Ant:

    def \_\_init\_\_(self, max\_cities):

        self.curCity = 0

        self.nextCity = -1

        self.tourLength = 0.0

        self.path = [-1] \* max\_cities

        self.pathIndex = 0

        self.tabu = [0] \* max\_cities

def getRand(max\_value):

    return random.randint(0, max\_value)

def getSRand():

    return random.random()

def init(cities, ants, distance, pheromone,

         max\_cities, max\_distance, init\_pheromone):

    for from\_city in range(max\_cities):

        cities[from\_city].x = getRand(max\_distance)

        cities[from\_city].y = getRand(max\_distance)

        for to\_city in range(max\_cities):

            distance[from\_city][to\_city] = 0.0

            pheromone[from\_city][to\_city] = init\_pheromone

    for from\_city in range(max\_cities):

        for to\_city in range(max\_cities):

            if from\_city != to\_city and distance[from\_city][to\_city] == 0.0:

                xd = abs(cities[from\_city].x - cities[to\_city].x)

                yd = abs(cities[from\_city].y - cities[to\_city].y)

                distance[from\_city][to\_city] = math.sqrt(xd \* xd + yd \* yd)

                distance[to\_city][from\_city] = distance[from\_city][to\_city]

    to\_city = 0

    for ant in ants:

        if to\_city == max\_cities:

            to\_city = 0

        ant.curCity = to\_city

        to\_city += 1

        ant.path = [-1] \* max\_cities

        ant.pathIndex = 1

        ant.path[0] = ant.curCity

        ant.nextCity = -1

        ant.tourLength = 0.0

        ant.tabu = [0] \* max\_cities

        ant.tabu[ant.curCity] = 1

def restartAnts(ants, best, bestIndex, max\_cities):

    to\_city = 0

    for ant in ants:

        if ant.tourLength < best:

            best = ant.tourLength

            bestIndex = ants.index(ant)

        ant.nextCity = -1

        ant.tourLength = 0.0

        ant.path = [-1] \* max\_cities

        ant.pathIndex = 1

        if to\_city == max\_cities:

            to\_city = 0

        ant.curCity = to\_city

        to\_city += 1

        ant.path[0] = ant.curCity

        ant.tabu = [0] \* max\_cities

        ant.tabu[ant.curCity] = 1

    return best, bestIndex

def antProduct(from\_city, to\_city, pheromone, distance, alpha, beta):

    try:

        return (pheromone[from\_city][to\_city] \*\* alpha) \

            \* ((1.0 / distance[from\_city][to\_city]) \*\* beta)

    except ZeroDivisionError:

        return 0.0

def selectNextCity(ant\_index, ants, pheromone, distance,

                   alpha, beta, max\_cities):

    from\_city = ants[ant\_index].curCity

    denom = sum(antProduct(from\_city, to\_city, pheromone, distance, alpha, beta)

                for to\_city in range(max\_cities) \

                    if ants[ant\_index].tabu[to\_city] == 0)

    if denom == 0.0:

        # Fallback: randomly select an unvisited city

        unvisited\_cities = [to\_city for to\_city in range(

            max\_cities) if ants[ant\_index].tabu[to\_city] == 0]

        return random.choice(unvisited\_cities)

    while True:

        to\_city = random.randint(0, max\_cities - 1)

        if ants[ant\_index].tabu[to\_city] == 0:

            p = antProduct(from\_city, to\_city, pheromone,

                           distance, alpha, beta) / denom

            if getSRand() < p:

                break

    return to\_city

def simulateAnts(ants, pheromone, distance, alpha, beta, max\_cities):

    moving = 0

    for k in range(len(ants)):

        if ants[k].pathIndex < max\_cities:

            ants[k].nextCity = selectNextCity(

                k, ants, pheromone, distance, alpha, beta, max\_cities)

            ants[k].tabu[ants[k].nextCity] = 1

            ants[k].path[ants[k].pathIndex] = ants[k].nextCity

            ants[k].pathIndex += 1

            ants[k].tourLength += distance[ants[k].curCity][ants[k].nextCity]

            if ants[k].pathIndex == max\_cities:

                ants[k].tourLength += distance[ants[k].path[max\_cities - 1]

                                               ][ants[k].path[0]]

            ants[k].curCity = ants[k].nextCity

            moving += 1

    return moving

def updateTrails(ants, pheromone, distance, rho, qval, max\_cities):

    global init\_pheromone

    for from\_city in range(max\_cities):

        for to\_city in range(max\_cities):

            if from\_city != to\_city:

                pheromone[from\_city][to\_city] \*= (1.0 - rho)

                if pheromone[from\_city][to\_city] < 0.0:

                    pheromone[from\_city][to\_city] = init\_pheromone

    for ant in ants:

        for i in range(max\_cities):

            if i < max\_cities - 1:

                from\_city = ant.path[i]

                to\_city = ant.path[i + 1]

            else:

                from\_city = ant.path[i]

                to\_city = ant.path[0]

            pheromone[from\_city][to\_city] += (qval / ant.tourLength)

            pheromone[to\_city][from\_city] = pheromone[from\_city][to\_city]

    for from\_city in range(max\_cities):

        for to\_city in range(max\_cities):

            pheromone[from\_city][to\_city] \*= rho

def main():

    global init\_pheromone, is\_running, \

        data\_blob, start\_time, best\_length\_x, best\_length\_y

    cities = [City() for \_ in range(data\_blob['max\_cities'])]

    ants = [Ant(data\_blob['max\_cities']) for \_ in range(data\_blob['max\_ants'])]

    distance = [[0.0] \* data\_blob['max\_cities']

                for \_ in range(data\_blob['max\_cities'])]

    pheromone = [[init\_pheromone] \* data\_blob['max\_cities']

                 for \_ in range(data\_blob['max\_cities'])]

    best = float('inf')

    bestIndex = 0

    init\_pheromone = data\_blob['init\_pheromone']

    random.seed()

    init(cities, ants, distance, pheromone,

         data\_blob['max\_cities'], data\_blob['max\_distance'], init\_pheromone)

    if not os.path.exists("out"):

        os.makedirs("out")

    curTime = 0

    while curTime < data\_blob['max\_time']:

        if not is\_running:

            break

        curTime += 1

        if simulateAnts(ants, pheromone, distance, data\_blob['alpha'], \

                        data\_blob['beta'], data\_blob['max\_cities']) == 0:

            updateTrails(ants, pheromone, distance, data\_blob['rho'], \

                         data\_blob['qval'], data\_blob['max\_cities'])

            if curTime != data\_blob['max\_time']:

                best, bestIndex = restartAnts(

                    ants, best, bestIndex, data\_blob['max\_cities'])

2. Интерфейс приложения должен обеспечивать визуализацию процесса   
поиска кратчайшего пути, а также ввод параметров алгоритма.

Интерфейс приложения

Изображение выглядит как снимок экрана, текст, диаграмма, линия

Автоматически созданное описание  
Рисунок 4. Интерфейс программы. Первые секунды работы алгоритма.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание  
Рисунок 5. Результат при начальных указанных параметрах.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание  
Рисунок 6. Результат при увеличенном времени работы алгоритма.

Также программа создает файл с результатом

best\_solution.txt

9 > 12 > 6 > 10 > 15 > 2 > 19 > 4 > 7 > 18 > 13 > 5 > 16 > 3 > 8 > 11 > 17 > 1 > 0 > 14 > 9

3. Исследовать алгоритм на сходимость. При каких значения параметров   
или их комбинациях, алгоритм сходится наилучшим образом. Итоги и   
параметры экспериментов зафиксировать в табличном виде.

Результаты исследования влияния параметров алгоритма

Результаты исследования приведены в таблице 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изменяемы параметр | Кол-во городов | Кол-во муравьев | Максимальное расстояние | Начальный уровень феромонов | Альфа | Бета | Ро | Q | Максимальное время | Результат | | | |
| Время, с | Лучшая длина | Решение | Комментарий |
| Кол-во городов | 1 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | - | - | 0 | Нельзя решить задачу с одним городом в графе |
| 5 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 3.92 | 239.84 | 1 | С увеличением количества городов выше максимального времени алгоритм теряет сходимость |
| 10 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 2.08 | 270.88 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.08 | 442.75 | 1 |
| 50 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 0.74 | 650.46 | 1 |
| 100 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 0.84 | inf | 0 |
| 200 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 0.36 | inf | 0 |
| Кол-во муравьев | 20 | 1 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.13 | 391.16 | 1 | Изменение кол-ва муравьев не влияет на сходимость, но при увеличении их количества выше числа городов значительно увеличивается время вычисления |
| 20 | 5 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.32 | 395.62 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.14 | 358.14 | 1 |
| 20 | 20 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.74 | 440.39 | 1 |
| 20 | 50 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 12.92 | 415.1 | 1 |
| 20 | 100 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 14.41 | 376.35 | 1 |
| 20 | 200 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 17.35 | 432.64 | 1 |
| 20 | 2000 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 73.36 | 328.67 | 1 |
| Максимальное растояние | 20 | 10 | 1 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.63 | 16 | 1 | Изменение максимального не влияет на сходимость |
| 20 | 10 | 20 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.63 | 74.18 | 1 |
| 20 | 10 | 50 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.58 | 195.81 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.59 | 352.43 | 1 |
| 20 | 10 | 500 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.86 | 2061.79 | 1 |
| 20 | 10 | 2000 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.81 | 7540.06 | 1 |
| Начальный уровень феромонов | 20 | 10 | 100 | 0 | 1 | 5 | 0.5 | 100 | 1000 | 11.83 | 410.36 | 1 | Лучшая длина на старте около 1000 |
| 20 | 10 | 100 | 0.1 | 1 | 5 | 0.5 | 100 | 1000 | 11.55 | 375.86 | 1 | Лучшая длина на старте около 450. Других изменений не замечено |
| 20 | 10 | 100 | 0.5 | 1 | 5 | 0.5 | 100 | 1000 | 11.51 | 419.96 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.72 | 432.7 | 1 |
| 20 | 10 | 100 | 1.5 | 1 | 5 | 0.5 | 100 | 1000 | 11.96 | 417.1 | 1 |
| 20 | 10 | 100 | 2 | 1 | 5 | 0.5 | 100 | 1000 | 11.65 | 358.46 | 1 |
| 20 | 10 | 100 | 10 | 1 | 5 | 0.5 | 100 | 1000 | 11.43 | 425.35 | 1 |
| 20 | 10 | 100 | 100 | 1 | 5 | 0.5 | 100 | 1000 | 11.34 | 459.6 | 1 |
| 20 | 10 | 100 | 1000 | 1 | 5 | 0.5 | 100 | 1000 | 11.82 | 379.27 | 1 |
| 20 | 10 | 100 | 0 | 1 | 5 | 0.5 | 100 | 100 | 1.21 | 432.85 | 1 | На графе много остаточного феромона, его кол-во учеличивается с увеличением начального феромона |
| 20 | 10 | 100 | 0.1 | 1 | 5 | 0.5 | 100 | 100 | 1.18 | 455.8 | 1 |
| 20 | 10 | 100 | 0.5 | 1 | 5 | 0.5 | 100 | 100 | 1.16 | 445.08 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.14 | 397.09 | 1 |
| 20 | 10 | 100 | 1.5 | 1 | 5 | 0.5 | 100 | 100 | 1.13 | 357.31 | 1 |
| 20 | 10 | 100 | 2 | 1 | 5 | 0.5 | 100 | 100 | 1.28 | 386.74 | 1 |
| 20 | 10 | 100 | 10 | 1 | 5 | 0.5 | 100 | 100 | 1.15 | 417.16 | 1 |
| 20 | 10 | 100 | 100 | 1 | 5 | 0.5 | 100 | 100 | 1.24 | 424.8 | 1 |
| 20 | 10 | 100 | 1000 | 1 | 5 | 0.5 | 100 | 100 | 1.22 | 344.2 | 1 |
| Альфа | 20 | 10 | 100 | 1 | -1 | 5 | 0.5 | 100 | 100 | 1.23 | 482.27 | 1 | На графе много остаточного феромона |
| 20 | 10 | 100 | 1 | 0 | 5 | 0.5 | 100 | 100 | 1.18 | 433.63 | 1 |
| 20 | 10 | 100 | 1 | 0.5 | 5 | 0.5 | 100 | 100 | 1.1 | 431.85 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.1 | 369.73 | 1 | С увеличением параметра альфа остается меньше остаточного феромнона на графе |
| 20 | 10 | 100 | 1 | 2 | 5 | 0.5 | 100 | 100 | 1.29 | 395.74 | 1 |
| 20 | 10 | 100 | 1 | 10 | 5 | 0.5 | 100 | 100 | 1.17 | 378.57 | 1 |
| 20 | 10 | 100 | 1 | 100 | 5 | 0.5 | 100 | 100 | 1.24 | 431.4 | 1 |
| 20 | 10 | 100 | 1 | 1000 | 5 | 0.5 | 100 | 100 | 1.1 | 401.34 | 1 |
| Бета | 20 | 10 | 100 | 1 | 1 | -5 | 0.5 | 100 | 100 | 1.21 | 1149.64 | 1 | С уменьшением параметра бета увеличивается найденное лучшее расстояние |
| 20 | 10 | 100 | 1 | 1 | 0 | 0.5 | 100 | 100 | 1.12 | 947.67 | 1 |
| 20 | 10 | 100 | 1 | 1 | 0.5 | 0.5 | 100 | 100 | 1.09 | 678.09 | 1 |
| 20 | 10 | 100 | 1 | 1 | 1 | 0.5 | 100 | 100 | 1.09 | 536.05 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.16 | 444.62 | 1 |
| 20 | 10 | 100 | 1 | 1 | 10 | 0.5 | 100 | 100 | 1.13 | 419.11 | 1 |
| 20 | 10 | 100 | 1 | 1 | 100 | 0.5 | 100 | 100 | 1.12 | 369.32 | 1 |
| 20 | 10 | 100 | 1 | 1 | 500 | 0.5 | 100 | 100 | 1.14 | 756.44 | 1 | После определенного порога алгоритму перестало хватать времени чтобы получить лучшее решение |
| 20 | 10 | 100 | 1 | 1 | 1000 | 0.5 | 100 | 100 | 1.16 | 876.5 | 1 |
| Ро | 20 | 10 | 100 | 1 | 1 | 5 | -2 | 100 | 100 | 1.2 | 393.82 | 1 | Стартовое решение остается финальным. Граф полностью заполнен феромоном |
| 20 | 10 | 100 | 1 | 1 | 5 | -0.5 | 100 | 100 | 1.22 | 429.08 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0 | 100 | 100 | 1.14 | 492.77 | 1 | Стартовое решение остается финальным. Граф не отрисовывается |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.21 | 345.53 | 1 | Нет особенностей |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.7 | 100 | 100 | 1.17 | 400.38 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 1 | 100 | 100 | 1.27 | 403.25 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 2 | 100 | 100 | 1.26 | 397.01 | 1 | На графе много остаточного феромона |
| 20 | 10 | 100 | 1 | 1 | 5 | 10 | 100 | 100 | 1.17 | 466.37 | 1 |
| Q | 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | -100 | 100 | 1.32 | 372.38 | 1 | Для значений феромона каждую итерацию используются случайные числа |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | -10 | 100 | 1.17 | 394.11 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 0 | 100 | 1.13 | 445.28 | 1 | Уровень феромона не снижается |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 1 | 100 | 1.26 | 421.9 | 1 | Замедление снижения феромона по сравн. с нормальной работой |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 10 | 100 | 1.2 | 342.78 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.28 | 397.54 | 1 | Нормальная работа алгоритма |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 1000 | 100 | 1.24 | 394.47 | 1 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 2000 | 100 | 1.2 | 402.3 | 1 |
| Максимальное время | 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | -1 | - | - | 0 | Алгоритм не выполняется |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 0 | - | - | 0 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 10 | - | - | 0 |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 100 | 1.36 | 413.89 | 1 | Нормальная работа алгоритма |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 1000 | 11.4 | 419.67 | 1 | Нормальная работа алгоритма. С увеличением максимального времени уменьшается кол-во остаточного феромона |
| 20 | 10 | 100 | 1 | 1 | 5 | 0.5 | 100 | 5000 | 58.41 | 391.38 | 1 |

Таблица 1. Результаты исследования.

Вывод

В рамках данной работы мы исследовали параметры муравьиного алгоритма на примере решения задачи коммивояжера. Для этого нами было разработано приложение, позволяющее визуализировать процесс поиска кратчайшего пути и изменять параметры алгоритма. Основная цель исследования заключалась в определении наилучших комбинаций параметров для достижения сходимости алгоритма и нахождения оптимальных решений.

Анализ результатов экспериментов показал, что с увеличением количества городов время работы алгоритма увеличивается, а качество найденного решения снижается. Например, при 50 городах лучшая длина пути составила 650.46, тогда как при 10 городах — 270.88. Это говорит о том, что для больших графов алгоритм требует более длительного времени на вычисление. Также мы обнаружили, что увеличение количества муравьев выше числа городов не способствует значительному улучшению решения, но существенно замедляет время выполнения алгоритма. Например, при 2000 муравьях алгоритм нашёл путь за 73.36 секунд, что почти в 6 раз дольше по сравнению с 10 муравьями.

Исследование влияния параметров альфа и бета показало, что увеличение значения альфа снижает количество остаточного феромона на графе, что улучшает сходимость алгоритма. Параметр бета, напротив, напрямую влияет на качество решения, и при его увеличении (например, до значения 5) найденное лучшее расстояние уменьшилось до 444.62, что подтвердило значимость данного параметра для поиска оптимального пути.

Мы также пришли к выводу, что начальный уровень феромонов не оказывает решающего влияния на сходимость алгоритма, но его изменение может помочь оптимизировать процесс. При высоком уровне феромонов (например, 1000) алгоритм быстрее сходится, но остаточный феромон на графе остаётся значительным, что замедляет процесс дальнейших итераций.

На основе проведённых экспериментов, мы рекомендуем для задачи коммивояжера с количеством вершин около 20 использовать параметр альфа в диапазоне 1-2 и бета — около 5, чтобы минимизировать остаточный феромон и одновременно улучшить результативность поиска.

Приложения

Приложение 1. Полный код программы.

<https://github.com/Kseen715/ai-foundations-lr/tree/main/lr3-ant-alg>

Python

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

# Python 3.12.6

import time

import dearpygui.dearpygui as dpg

import networkx as nx

from matplotlib import pyplot as plt

from tabulate import tabulate

from matplotlib.backends.backend\_agg import FigureCanvasAgg

from PIL import Image

import os

import random

import math

import threading

import numpy as np

from matplotlib.colors import LinearSegmentedColormap

from matplotlib import use as use\_backend

use\_backend('Agg')

# Global variables

process = None

is\_running = False

init\_pheromone = 0

data\_blob = {

    'max\_cities': None,

    'max\_ants': None,

    'max\_distance': None,

    'init\_pheromone': None,

    'alpha': None,

    'beta': None,

    'rho': None,

    'qval': None,

    'max\_time': None,

    'networkx\_seed': None,

    'stop\_on\_change': None,

}

default\_data\_blob = {

    'max\_cities': 20,

    'max\_ants': 10,

    'max\_distance': 100,

    'init\_pheromone': 1.0,

    'alpha': 1.0,

    'beta': 5.0,

    'rho': 0.5,

    'qval': 100,

    'max\_time': 100,

    'networkx\_seed': 100,

    'stop\_on\_change': True,

}

best\_length\_y = []

best\_length\_x = []

start\_time = 0

# Constants

TEXTURE\_FACTOR = 500

TEXTURE\_WIDTH = TEXTURE\_FACTOR

TEXTURE\_HEIGHT = TEXTURE\_FACTOR

NETWOKX\_SIZE\_DELIMITER = 500

DEFAULT\_RES\_WIDTH\_COEF = 3

DEFAULT\_RES\_HEIGHT\_COEF = 3

DEFAULT\_RES\_FACTOR = 250

DEFAULT\_RES\_WIDTH = int(DEFAULT\_RES\_WIDTH\_COEF \* DEFAULT\_RES\_FACTOR)

DEFAULT\_RES\_HEIGHT = int(DEFAULT\_RES\_HEIGHT\_COEF \* DEFAULT\_RES\_FACTOR)

class City:

    def \_\_init\_\_(self, x=0, y=0):

        self.x = x

        self.y = y

class Ant:

    def \_\_init\_\_(self, max\_cities):

        self.curCity = 0

        self.nextCity = -1

        self.tourLength = 0.0

        self.path = [-1] \* max\_cities

        self.pathIndex = 0

        self.tabu = [0] \* max\_cities

def getRand(max\_value):

    return random.randint(0, max\_value)

def getSRand():

    return random.random()

def init(cities, ants, distance, pheromone,

         max\_cities, max\_distance, init\_pheromone):

    for from\_city in range(max\_cities):

        cities[from\_city].x = getRand(max\_distance)

        cities[from\_city].y = getRand(max\_distance)

        for to\_city in range(max\_cities):

            distance[from\_city][to\_city] = 0.0

            pheromone[from\_city][to\_city] = init\_pheromone

    for from\_city in range(max\_cities):

        for to\_city in range(max\_cities):

            if from\_city != to\_city and distance[from\_city][to\_city] == 0.0:

                xd = abs(cities[from\_city].x - cities[to\_city].x)

                yd = abs(cities[from\_city].y - cities[to\_city].y)

                distance[from\_city][to\_city] = math.sqrt(xd \* xd + yd \* yd)

                distance[to\_city][from\_city] = distance[from\_city][to\_city]

    to\_city = 0

    for ant in ants:

        if to\_city == max\_cities:

            to\_city = 0

        ant.curCity = to\_city

        to\_city += 1

        ant.path = [-1] \* max\_cities

        ant.pathIndex = 1

        ant.path[0] = ant.curCity

        ant.nextCity = -1

        ant.tourLength = 0.0

        ant.tabu = [0] \* max\_cities

        ant.tabu[ant.curCity] = 1

def restartAnts(ants, best, bestIndex, max\_cities):

    to\_city = 0

    for ant in ants:

        if ant.tourLength < best:

            best = ant.tourLength

            bestIndex = ants.index(ant)

        ant.nextCity = -1

        ant.tourLength = 0.0

        ant.path = [-1] \* max\_cities

        ant.pathIndex = 1

        if to\_city == max\_cities:

            to\_city = 0

        ant.curCity = to\_city

        to\_city += 1

        ant.path[0] = ant.curCity

        ant.tabu = [0] \* max\_cities

        ant.tabu[ant.curCity] = 1

    return best, bestIndex

def antProduct(from\_city, to\_city, pheromone, distance, alpha, beta):

    try:

        return (pheromone[from\_city][to\_city] \*\* alpha) \

            \* ((1.0 / distance[from\_city][to\_city]) \*\* beta)

    except ZeroDivisionError:

        return 0.0

def selectNextCity(ant\_index, ants, pheromone, distance,

                   alpha, beta, max\_cities):

    from\_city = ants[ant\_index].curCity

    denom = sum(antProduct(from\_city, to\_city, pheromone, distance, alpha, beta)

                for to\_city in range(max\_cities) \

                    if ants[ant\_index].tabu[to\_city] == 0)

    if denom == 0.0:

        # Fallback: randomly select an unvisited city

        unvisited\_cities = [to\_city for to\_city in range(

            max\_cities) if ants[ant\_index].tabu[to\_city] == 0]

        return random.choice(unvisited\_cities)

    while True:

        to\_city = random.randint(0, max\_cities - 1)

        if ants[ant\_index].tabu[to\_city] == 0:

            p = antProduct(from\_city, to\_city, pheromone,

                           distance, alpha, beta) / denom

            if getSRand() < p:

                break

    return to\_city

def simulateAnts(ants, pheromone, distance, alpha, beta, max\_cities):

    moving = 0

    for k in range(len(ants)):

        if ants[k].pathIndex < max\_cities:

            ants[k].nextCity = selectNextCity(

                k, ants, pheromone, distance, alpha, beta, max\_cities)

            ants[k].tabu[ants[k].nextCity] = 1

            ants[k].path[ants[k].pathIndex] = ants[k].nextCity

            ants[k].pathIndex += 1

            ants[k].tourLength += distance[ants[k].curCity][ants[k].nextCity]

            if ants[k].pathIndex == max\_cities:

                ants[k].tourLength += distance[ants[k].path[max\_cities - 1]

                                               ][ants[k].path[0]]

            ants[k].curCity = ants[k].nextCity

            moving += 1

    return moving

def updateTrails(ants, pheromone, distance, rho, qval, max\_cities):

    global init\_pheromone

    for from\_city in range(max\_cities):

        for to\_city in range(max\_cities):

            if from\_city != to\_city:

                pheromone[from\_city][to\_city] \*= (1.0 - rho)

                if pheromone[from\_city][to\_city] < 0.0:

                    pheromone[from\_city][to\_city] = init\_pheromone

    for ant in ants:

        for i in range(max\_cities):

            if i < max\_cities - 1:

                from\_city = ant.path[i]

                to\_city = ant.path[i + 1]

            else:

                from\_city = ant.path[i]

                to\_city = ant.path[0]

            pheromone[from\_city][to\_city] += (qval / ant.tourLength)

            pheromone[to\_city][from\_city] = pheromone[from\_city][to\_city]

    for from\_city in range(max\_cities):

        for to\_city in range(max\_cities):

            pheromone[from\_city][to\_city] \*= rho

def emitDataFile(cities, ants, ant\_index):

    with open("out/cities.dat", "w") as fp:

        for city in cities:

            fp.write(f"{city.x} {city.y}\n")

    with open("out/solution.dat", "w") as fp:

        for city\_index in ants[ant\_index].path:

            fp.write(f"{cities[city\_index].x} {cities[city\_index].y}\n")

        fp.write(f"{cities[ants[ant\_index].path[0]].x} {

                 cities[ants[ant\_index].path[0]].y}\n")

def main():

    global init\_pheromone, TEXTURE\_WIDTH, TEXTURE\_HEIGHT, is\_running, \

        data\_blob, start\_time, best\_length\_x, best\_length\_y

    best\_length\_x.clear()

    best\_length\_y.clear()

    cities = [City() for \_ in range(data\_blob['max\_cities'])]

    ants = [Ant(data\_blob['max\_cities']) for \_ in range(data\_blob['max\_ants'])]

    distance = [[0.0] \* data\_blob['max\_cities']

                for \_ in range(data\_blob['max\_cities'])]

    pheromone = [[init\_pheromone] \* data\_blob['max\_cities']

                 for \_ in range(data\_blob['max\_cities'])]

    best = float('inf')

    bestIndex = 0

    init\_pheromone = data\_blob['init\_pheromone']

    random.seed()

    init(cities, ants, distance, pheromone,

         data\_blob['max\_cities'], data\_blob['max\_distance'], init\_pheromone)

    if not os.path.exists("out"):

        os.makedirs("out")

    start\_time = time.time()

    curTime = 0

    while curTime < data\_blob['max\_time']:

        if not is\_running:

            break

        curTime += 1

        if simulateAnts(ants, pheromone, distance, data\_blob['alpha'], \

                        data\_blob['beta'], data\_blob['max\_cities']) == 0:

            updateTrails(ants, pheromone, distance, data\_blob['rho'], \

                         data\_blob['qval'], data\_blob['max\_cities'])

            if curTime != data\_blob['max\_time']:

                best, bestIndex = restartAnts(

                    ants, best, bestIndex, data\_blob['max\_cities'])

            # PLOTTING

            cmap = LinearSegmentedColormap.from\_list(

                'custom\_cmap', ['#FFA500', '#FF0000', '#00FF00'])

            # plot the pheromone matrix as networkx graph

            max\_cities = data\_blob['max\_cities']

            networkx\_seed = data\_blob['networkx\_seed']

            texture\_res\_factor = TEXTURE\_FACTOR / NETWOKX\_SIZE\_DELIMITER

            # Compute radius based on max\_cities

            size\_factor = math.sqrt(

                math.log(max\_cities) / (math.pi \* max\_cities))

            G = nx.random\_geometric\_graph(

                max\_cities, radius=1, seed=networkx\_seed)

            pos = nx.get\_node\_attributes(G, "pos")

            fig = plt.figure()

            limits = plt.axis("off")  # turn off axis

            fig.tight\_layout(pad=0)

            # resize plot in pixels

            fig.set\_size\_inches(

                TEXTURE\_WIDTH / fig.get\_dpi(), TEXTURE\_WIDTH / fig.get\_dpi())

            for i in range(data\_blob['max\_cities']):

                G.add\_node(i, weight=0.4)

            for i in range(data\_blob['max\_cities']):

                for j in range(i + 1, data\_blob['max\_cities']):

                    G.add\_edge(i, j, weight=pheromone[i][j])

            # Extract weights

            weights = [G[u][v]['weight'] for u, v in G.edges()]

            # Normalize weights to range [0, 1]

            norm\_weights = np.array(weights) / max(weights)

            # Scale weights with texture\_res\_factor

            widths = norm\_weights \* texture\_res\_factor \* 4

            # Map normalized weights to colors

            edge\_colors = [cmap(w) for w in norm\_weights]

            # Draw the graph with weights as edge widths

            nx.draw\_networkx(G, pos, with\_labels=True, width=widths, \

                             edge\_color=edge\_colors,

                             node\_size=size\_factor

                             \* 1000 \* (texture\_res\_factor \*\* 2),

                             node\_color='#88AAFF',

                             font\_size=size\_factor

                             \* 50 \* (texture\_res\_factor),

                             font\_family='monospace')

            canvas = FigureCanvasAgg(fig)

            canvas.draw()

            width, height = fig.get\_size\_inches() \* fig.get\_dpi()

            pixel\_data = canvas.buffer\_rgba()

            # normalize pixel data to 0-1

            pixel\_data = [x / 255 for x in bytearray(pixel\_data)]

            dpg.set\_value("texture\_tag", pixel\_data)

            plt.close(fig)

            \_time = float(time.time() - start\_time)

            best\_length\_y.append(best)

            best\_length\_x.append(\_time)

            if len(best\_length\_x) > 0:

                dpg.set\_value("series\_best\_length", [

                              best\_length\_x, best\_length\_y])

                dpg.fit\_axis\_data("y\_axis\_best\_length")

                dpg.fit\_axis\_data("x\_axis\_best\_length")

                dpg.set\_value("temp\_text", f"{best:.2f}")

            dpg.set\_value("shown\_iteration\_text", f"{curTime}")

            if -1 in ants[bestIndex].path:

                dpg.set\_value("is\_solution\_found", "No")

            else:

                dpg.set\_value("is\_solution\_found", "Yes")

            # min\_y = min(best\_length\_y)

            max\_y = max(best\_length\_y)

            dpg.set\_axis\_limits("y\_axis\_best\_length", 0, max\_y + 0.01 \* max\_y)

        \_time = float(time.time() - start\_time)

        dpg.set\_value("time\_text", f"{\_time:.2f}")

        dpg.set\_value("iteration\_text", f"{curTime}")

        # /PLOTTING

        # print(f"Time is {curTime} ({best})")

    # print(f"Best tour length: {best}")

    best\_solution = ""

    if -1 in ants[bestIndex].path:

        # if -1 in solution, print error

        # print("Solution was not found")

        best\_solution = "Solution was not found"

    else:

        # print best solution as 12 > 23 > 234 > ...

        # print("Best solution:")

        for city\_index in ants[bestIndex].path:

            best\_solution += f"{city\_index} > "

        best\_solution += f"{ants[bestIndex].path[0]}"

        # print(best\_solution)

    # Save best solution to file

    with open("out/best\_solution.txt", "w") as file:

        file.write(best\_solution)

        emitDataFile(cities, ants, bestIndex)

    stop()

def update\_layout():

    port\_width = dpg.get\_viewport\_client\_width()

    port\_height = dpg.get\_viewport\_client\_height()

    inputs\_height = 26

    image\_size = min(port\_width, port\_height - inputs\_height \* 10)

    plots\_width = port\_width - image\_size - 20

    plots\_count = 2

    plots\_height = image\_size / plots\_count - plots\_count + 1  # / 3.031

    inputs\_width = min(max(port\_width \* 0.15, 100), 500)

    outputs\_width = min(max(port\_width \* 0.15, 100), 500)

    if dpg.does\_item\_exist("image"):

        dpg.configure\_item("image", width=image\_size, height=image\_size)

    if dpg.does\_item\_exist("num\_cities"):

        dpg.configure\_item("num\_cities", width=inputs\_width)

    if dpg.does\_item\_exist("num\_ants"):

        dpg.configure\_item("num\_ants", width=inputs\_width)

    if dpg.does\_item\_exist("max\_distance"):

        dpg.configure\_item("max\_distance", width=inputs\_width)

    if dpg.does\_item\_exist("init\_pheromone"):

        dpg.configure\_item("init\_pheromone", width=inputs\_width)

    if dpg.does\_item\_exist("alpha"):

        dpg.configure\_item("alpha", width=inputs\_width)

    if dpg.does\_item\_exist("beta"):

        dpg.configure\_item("beta", width=inputs\_width)

    if dpg.does\_item\_exist("rho"):

        dpg.configure\_item("rho", width=inputs\_width)

    if dpg.does\_item\_exist("qval"):

        dpg.configure\_item("qval", width=inputs\_width)

    if dpg.does\_item\_exist("max\_time"):

        dpg.configure\_item("max\_time", width=inputs\_width)

    if dpg.does\_item\_exist("temp\_text"):

        dpg.configure\_item("temp\_text", width=outputs\_width)

    if dpg.does\_item\_exist("iteration\_text"):

        dpg.configure\_item("iteration\_text", width=outputs\_width)

    if dpg.does\_item\_exist("shown\_iteration\_text"):

        dpg.configure\_item("shown\_iteration\_text", width=outputs\_width)

    if dpg.does\_item\_exist("time\_text"):

        dpg.configure\_item("time\_text", width=outputs\_width)

    if dpg.does\_item\_exist("networkx\_seed"):

        dpg.configure\_item("networkx\_seed", width=outputs\_width)

    if dpg.does\_item\_exist("is\_solution\_found"):

        dpg.configure\_item("is\_solution\_found", width=outputs\_width)

    if dpg.does\_item\_exist("plot\_best\_length"):

        dpg.configure\_item("plot\_best\_length",

                           width=plots\_width, height=plots\_height)

def read\_data\_blob\_from\_ui():

    global data\_blob

    data\_blob['max\_cities'] = dpg.get\_value("num\_cities")

    data\_blob['max\_ants'] = dpg.get\_value("num\_ants")

    data\_blob['max\_distance'] = dpg.get\_value("max\_distance")

    data\_blob['init\_pheromone'] = dpg.get\_value("init\_pheromone")

    data\_blob['alpha'] = dpg.get\_value("alpha")

    data\_blob['beta'] = dpg.get\_value("beta")

    data\_blob['rho'] = dpg.get\_value("rho")

    data\_blob['qval'] = dpg.get\_value("qval")

    data\_blob['max\_time'] = dpg.get\_value("max\_time")

    data\_blob['networkx\_seed'] = dpg.get\_value("networkx\_seed")

    data\_blob['stop\_on\_change'] = dpg.get\_value("stop\_on\_change")

def run():

    global is\_running

    global process

    stop()

    if process is None:

        is\_running = True

        read\_data\_blob\_from\_ui()

        process = threading.Thread(target=main)

        process.start()

def stop():

    global is\_running

    global process

    is\_running = False

    if process is not None:

        try:

            process.join()

        except:

            pass

    process = None

def reset():

    global data\_blob, default\_data\_blob

    data\_blob = default\_data\_blob

    dpg.set\_value("num\_cities", data\_blob['max\_cities'])

    dpg.set\_value("num\_ants", data\_blob['max\_ants'])

    dpg.set\_value("max\_distance", data\_blob['max\_distance'])

    dpg.set\_value("init\_pheromone", data\_blob['init\_pheromone'])

    dpg.set\_value("alpha", data\_blob['alpha'])

    dpg.set\_value("beta", data\_blob['beta'])

    dpg.set\_value("rho", data\_blob['rho'])

    dpg.set\_value("qval", data\_blob['qval'])

    dpg.set\_value("max\_time", data\_blob['max\_time'])

    dpg.set\_value("networkx\_seed", data\_blob['networkx\_seed'])

    dpg.set\_value("stop\_on\_change", data\_blob['stop\_on\_change'])

def gen\_texture\_empty(width, height, scale=1):

    texture = []

    for y in range(height):

        for x in range(width):

            if ((x // scale) + (y // scale)) % 2 == 0:

                texture.extend([0.5, 0.0, 0.5, 1.0])  # Purple

            else:

                texture.extend([0.0, 0.0, 0.0, 1.0])  # Black

    return texture

def gen\_texture\_solid(width: int, height: int, color: list) -> list:

    """Generate a solid color texture.

    Args:

        width (int): Width of the texture.

        height (int): Height of the texture.

        color (list): List of RGBA values.

        For example, [1.0, 0.0, 0.0, 1.0] is red.

    Returns:

        list: List of RGBA values.

    """

    texture = []

    for y in range(height):

        for x in range(width):

            texture.extend(color)

    return texture

def app():

    global TEXTURE\_WIDTH, TEXTURE\_HEIGHT, data\_blob, best\_length\_x, \

        best\_length\_y, start\_time

    dpg.create\_context()

    with dpg.font\_registry():

        with dpg.font(r"public/fonts/Roboto-Regular.ttf",

            14,

            default\_font=True

        ) as default\_font:

            dpg.add\_font\_range\_hint(dpg.mvFontRangeHint\_Default)

            dpg.add\_font\_range\_hint(dpg.mvFontRangeHint\_Cyrillic)

    dpg.create\_viewport(title='Ant algorithm',

                        width=DEFAULT\_RES\_WIDTH, height=DEFAULT\_RES\_HEIGHT)

    dpg.set\_viewport\_resize\_callback(update\_layout)

    with dpg.texture\_registry(show=False):

        dpg.add\_dynamic\_texture(

            width=TEXTURE\_WIDTH, height=TEXTURE\_HEIGHT,

            default\_value=gen\_texture\_solid(TEXTURE\_WIDTH, TEXTURE\_HEIGHT,

                                            [1, 1, 1, 1]),

            tag="texture\_tag")

    with dpg.window(label="Example Window", tag="fullscreen"):

        with dpg.theme(tag="plot\_theme\_best\_length"):

            with dpg.theme\_component(dpg.mvLineSeries):

                dpg.add\_theme\_color(

                    dpg.mvPlotCol\_Line, (255, 99, 99),

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_Marker, dpg.mvPlotMarker\_None,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_MarkerSize, 0,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_LineWeight, 3.0,

                    category=dpg.mvThemeCat\_Plots)

        def on\_value\_change\_num\_cities(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['num\_cities'] = int(app\_data)

        def on\_value\_change\_num\_ants(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['num\_ants'] = int(app\_data)

        def on\_value\_change\_max\_distance(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['max\_distance'] = int(app\_data)

        def on\_value\_change\_init\_pheromone(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['init\_pheromone'] = float(app\_data)

        def on\_value\_change\_alpha(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['alpha'] = float(app\_data)

        def on\_value\_change\_beta(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['beta'] = float(app\_data)

        def on\_value\_change\_rho(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['rho'] = float(app\_data)

        def on\_value\_change\_qval(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['qval'] = float(app\_data)

        def on\_value\_change\_max\_time(sender, app\_data, user\_data):

            if user\_data['stop\_on\_change']:

                stop()

            user\_data['max\_time'] = int(app\_data)

        def on\_value\_change\_networkx\_seed(sender, app\_data, user\_data):

            user\_data['networkx\_seed'] = int(app\_data)

        def on\_value\_change\_stop\_on\_change(sender, app\_data, user\_data):

            user\_data['stop\_on\_change'] = app\_data

        with dpg.group(horizontal=True):

            with dpg.group():

                # Max city count

                dpg.add\_input\_int(label="Number of cities",

                                  default\_value=20,

                                  width=200,

                                  user\_data=data\_blob,

                                  callback=on\_value\_change\_num\_cities,

                                  min\_value=1, max\_value=999,

                                  tag="num\_cities")

                # Max ant count

                dpg.add\_input\_int(label="Number of ants",

                                  default\_value=10,

                                  width=200,

                                  user\_data=data\_blob,

                                  callback=on\_value\_change\_num\_ants,

                                  min\_value=1, max\_value=999,

                                  tag="num\_ants")

                # Max distance

                dpg.add\_input\_int(label="Maximum distance",

                                  default\_value=100,

                                  width=200,

                                  user\_data=data\_blob,

                                  callback=on\_value\_change\_max\_distance,

                                  min\_value=1,

                                  tag="max\_distance")

                # Initial pheromone

                dpg.add\_input\_float(label="Initial pheromone level",

                                    default\_value=1.0,

                                    width=200,

                                    user\_data=data\_blob,

                                    callback=on\_value\_change\_init\_pheromone,

                                    min\_value=0.0,

                                    step=0.01,

                                    tag="init\_pheromone")

                # Alpha

                dpg.add\_input\_float(label="Alpha",

                                    default\_value=1.0,

                                    width=200,

                                    user\_data=data\_blob,

                                    callback=on\_value\_change\_alpha,

                                    min\_value=0.0,

                                    step=0.01,

                                    tag="alpha")

                # Beta

                dpg.add\_input\_float(label="Beta",

                                    default\_value=5.0,

                                    width=200,

                                    user\_data=data\_blob,

                                    callback=on\_value\_change\_beta,

                                    min\_value=0.0,

                                    step=0.1,

                                    tag="beta")

                # Rho

                dpg.add\_input\_float(label="Rho",

                                    default\_value=0.5,

                                    width=200,

                                    user\_data=data\_blob,

                                    callback=on\_value\_change\_rho,

                                    min\_value=0.0, max\_value=0.999999,

                                    step=0.01,

                                    tag="rho")

                # Q value

                dpg.add\_input\_float(label="Q value",

                                    default\_value=100,

                                    width=200,

                                    user\_data=data\_blob,

                                    callback=on\_value\_change\_qval,

                                    min\_value=0.0,

                                    step=0.01,

                                    tag="qval")

                # Max time

                dpg.add\_input\_int(label="Maximum time",

                                  default\_value=100,

                                  width=200,

                                  user\_data=data\_blob,

                                  callback=on\_value\_change\_max\_time,

                                  min\_value=1,

                                  tag="max\_time")

            with dpg.group():

                with dpg.group(horizontal=True):

                    with dpg.group():

                        # text output

                        dpg.add\_input\_text(default\_value="...",

                                           readonly=True,

                                           tag="temp\_text",

                                           width=50)

                        dpg.add\_input\_text(default\_value="...",

                                           readonly=True,

                                           tag="iteration\_text",

                                           width=50)

                        dpg.add\_input\_text(default\_value="...",

                                           readonly=True,

                                           tag="shown\_iteration\_text",

                                           width=50)

                        dpg.add\_input\_text(default\_value="...",

                                           readonly=True,

                                           tag="time\_text",

                                           width=50)

                        dpg.add\_input\_text(default\_value="...",

                                             readonly=True,

                                             tag="is\_solution\_found",

                                             width=50)

                    with dpg.group():

                        dpg.add\_text("Best tour length")

                        dpg.add\_text("Iteration")

                        dpg.add\_text("Iteration on screen")

                        dpg.add\_text("Time, s")

                        dpg.add\_text("Solution found")

                with dpg.group():

                    dpg.add\_input\_int(label="Image seed",

                                      default\_value=100,

                                      width=50,

                                      user\_data=data\_blob,

                                      callback=on\_value\_change\_networkx\_seed,

                                      min\_value=1,

                                      tag="networkx\_seed")

                    dpg.add\_checkbox(label="Stop on change",

                                     default\_value=True,

                                     tag="stop\_on\_change")

        with dpg.group(horizontal=True):

            dpg.add\_button(label="run", callback=run)

            dpg.add\_button(label="reset defaults", callback=reset)

            dpg.add\_button(label="stop", callback=stop)

        # global texture\_width, texture\_height

        text\_width = dpg.get\_viewport\_client\_width() \* 5 / 10

        text\_height = dpg.get\_viewport\_client\_width() \* 5 / 10

        with dpg.group(horizontal=True):

            dpg.add\_image("texture\_tag",

                          width=text\_width,

                          height=text\_height, tag="image")

            plots\_width = text\_width / 2

            plots\_height = text\_height / 3.031  # 4.548

            with dpg.group():

                with dpg.plot(label="Best Length",

                              width=plots\_width,

                              height=plots\_height,

                              tag="plot\_best\_length"):

                    dpg.add\_plot\_legend()

                    dpg.add\_plot\_axis(

                        dpg.mvXAxis, label="time. s", tag="x\_axis\_best\_length")

                    dpg.add\_plot\_axis(

                        dpg.mvYAxis, label="best length",

                        tag="y\_axis\_best\_length")

                    dpg.add\_line\_series(

                        best\_length\_x, best\_length\_y,

                        parent="y\_axis\_best\_length",

                        tag="series\_best\_length")

                dpg.bind\_item\_theme("series\_best\_length",

                                    "plot\_theme\_best\_length")

    update\_layout()

    dpg.bind\_font(default\_font)

    dpg.setup\_dearpygui()

    dpg.set\_primary\_window("fullscreen", True)

    dpg.show\_viewport()

    dpg.start\_dearpygui()

    dpg.destroy\_context()

if \_\_name\_\_ == "\_\_main\_\_":

    app()