

**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
Код алгоритма.....	6
Интерфейс приложения	10
Результаты исследования влияния параметров алгоритма.....	12
Вывод.....	15
Приложения	17

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 коммивояжера с использованием муравьиного алгоритма

Блок-схемы алгоритма представлены на рисунках 1-3.

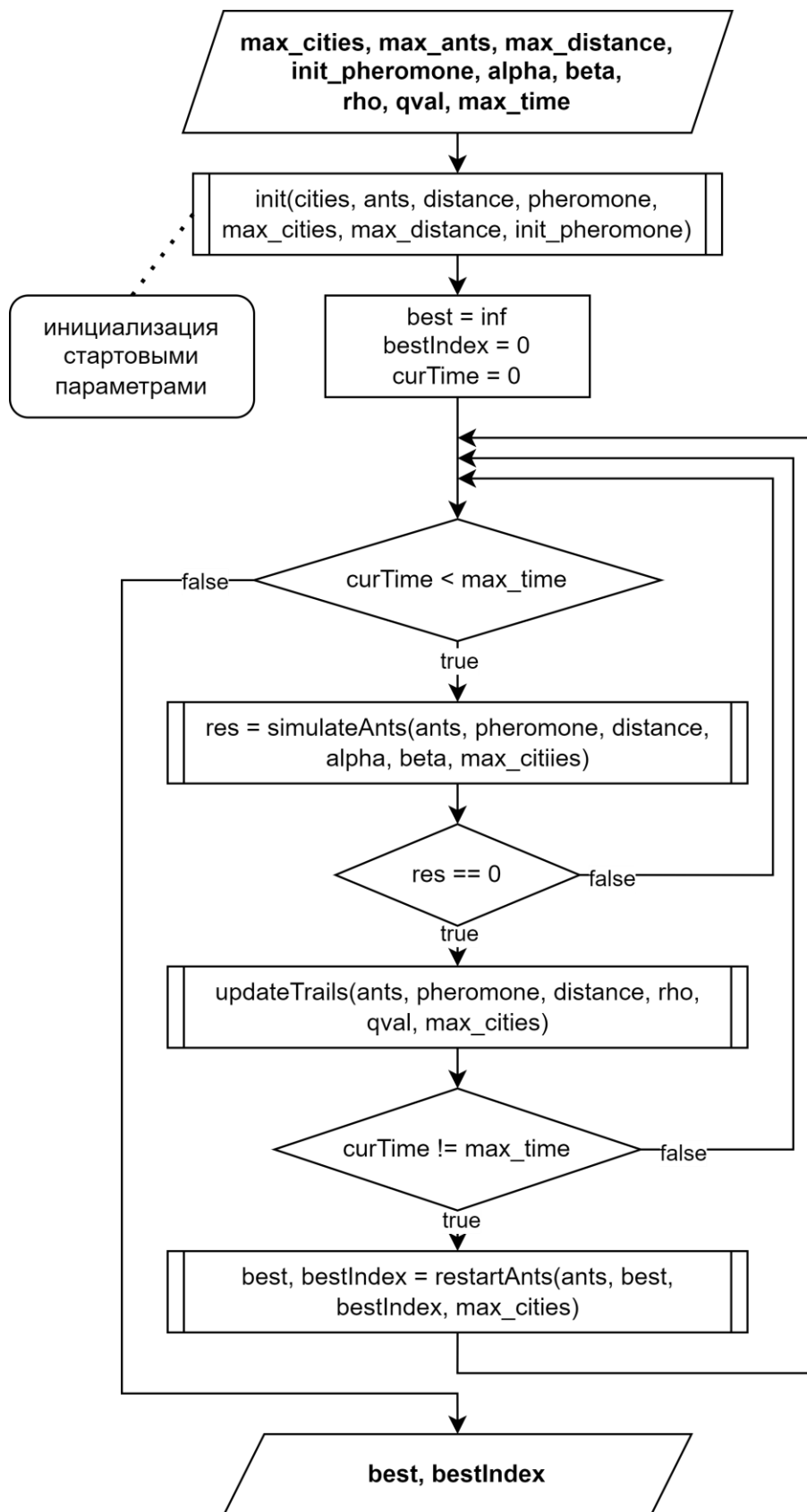


Рисунок 1. Блок-схема решения задачи коммивояжера с использованием муравьиного алгоритма.

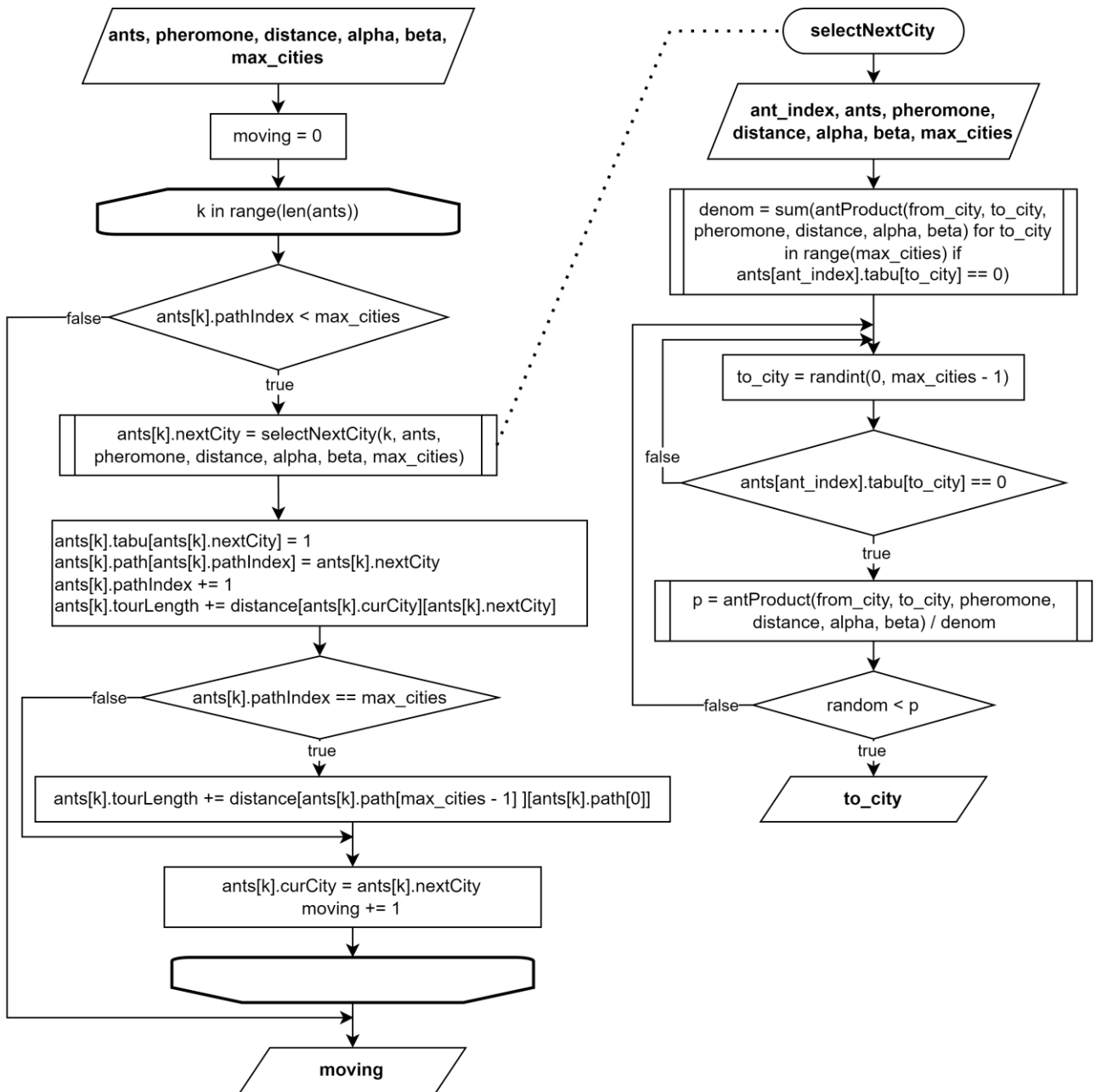


Рисунок 2. Блок-схема процедуры simulateAnts().

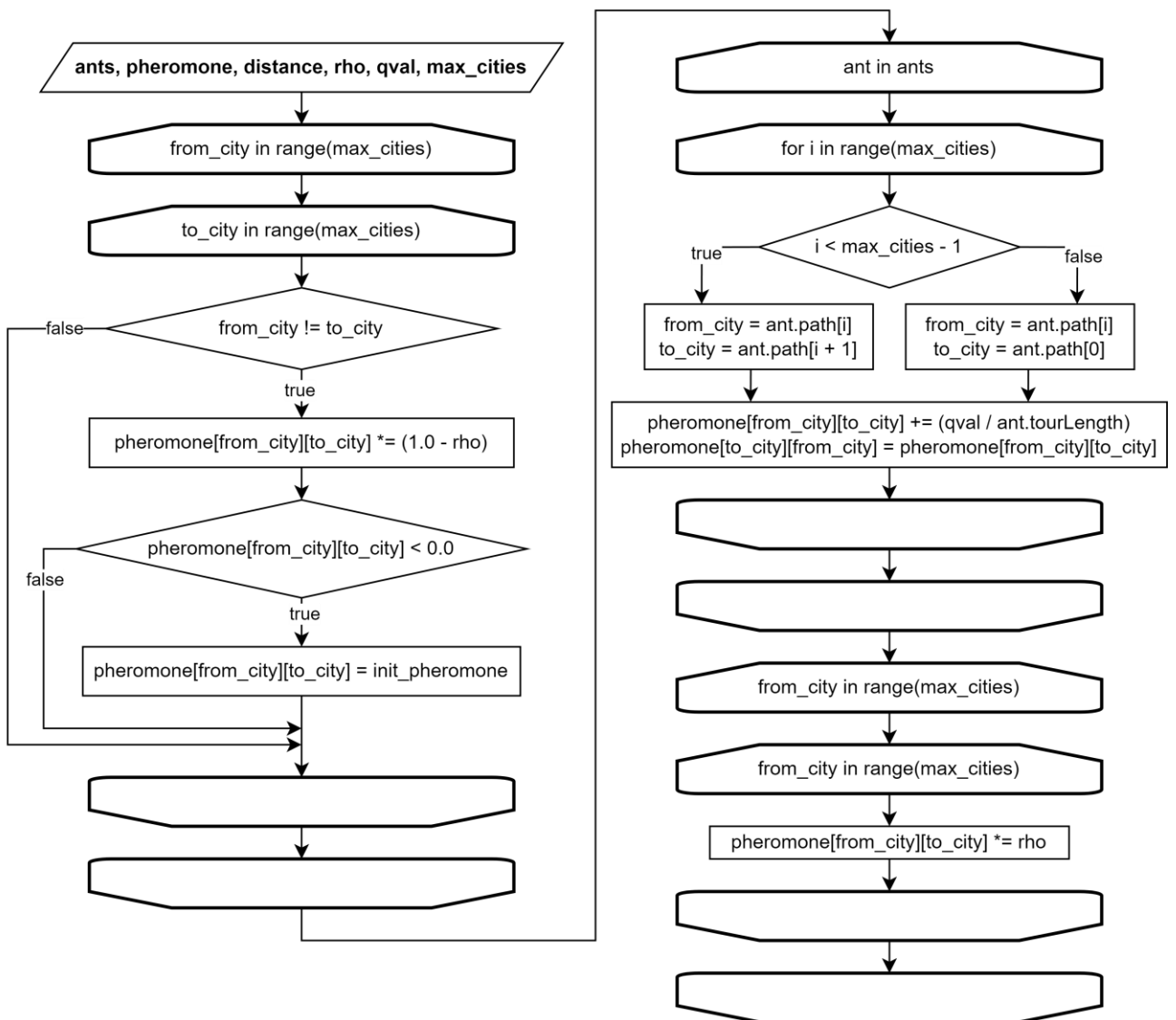


Рисунок 3. Блок-схема процедуры updateTrails().

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

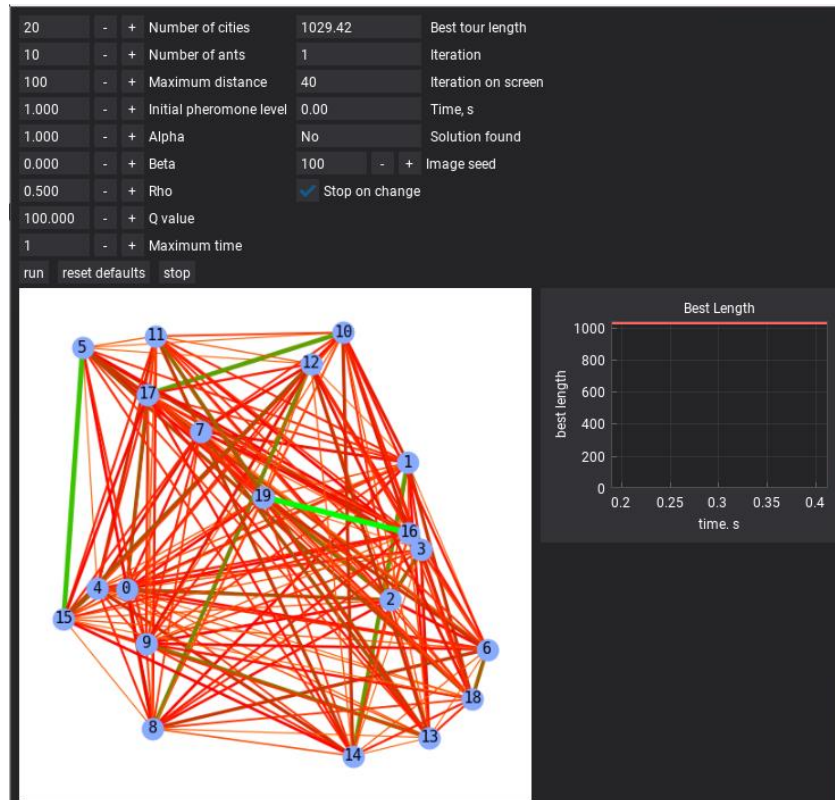


Рисунок 4. Интерфейс программы. Первые секунды работы алгоритма.

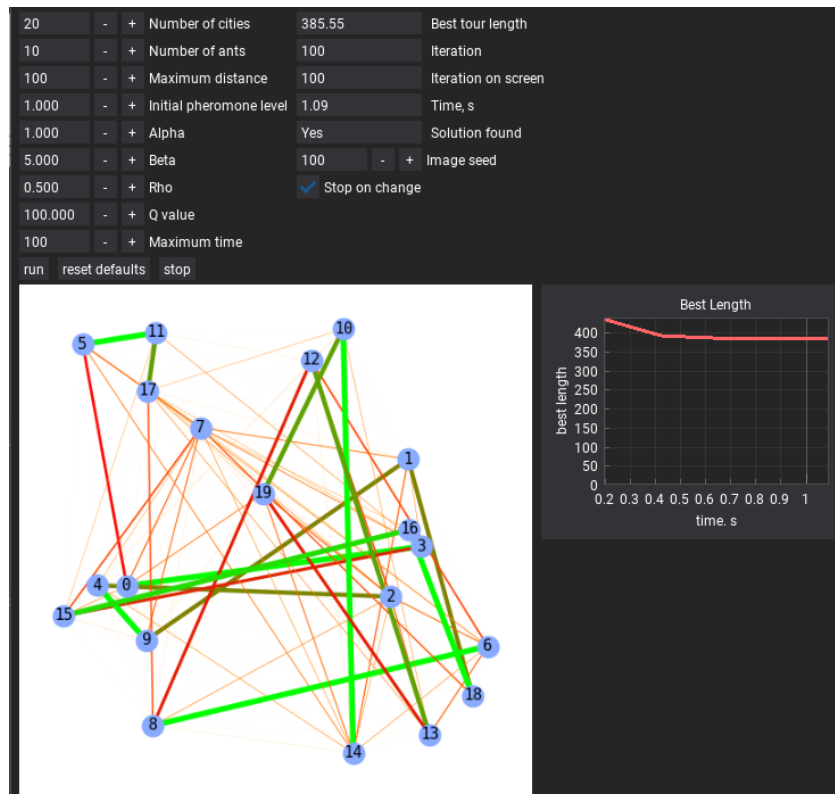


Рисунок 5. Результат при начальных указанных параметрах.

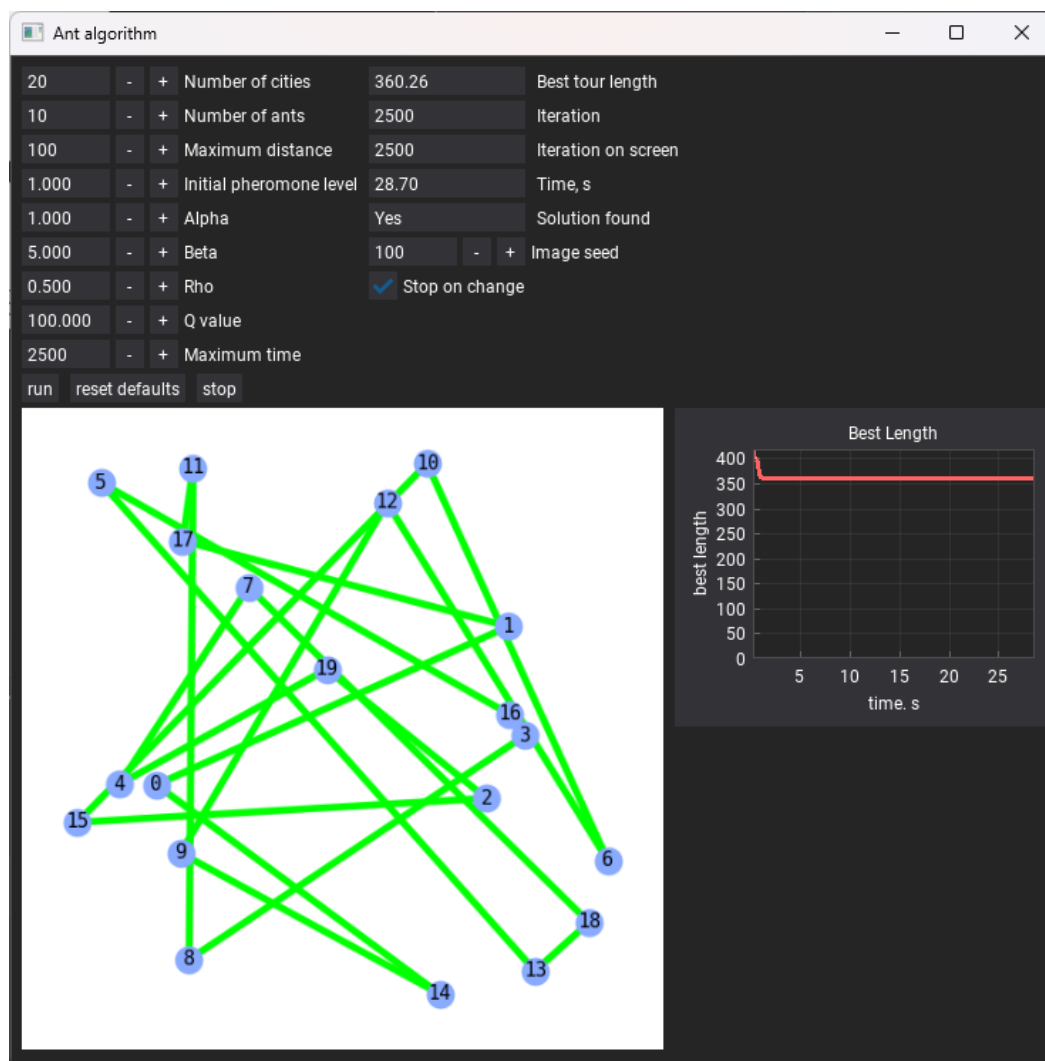


Рисунок 6. Результат при увеличенном времени работы алгоритма.

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.

Изменяемый параметр	Кол-во городов	Кол-во муравьев	Максимальное расстояние	Начальный уровень	Альфа	Бета	P ₀	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 расстояние	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	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	

	20	10	100	0.5	1	5	0.5	100	1000	11.51	419.96	1	Лучшая длина на старте около 450. Других изменений не замечено
	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 увеличивается 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 феромона 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	

Р ₀	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	Стартовое решение остается финальным. Граф не отрисовывается
	20	10	100	1	1	5	0	100	100	1.14	492.77	1	Нет особенностей
	20	10	100	1	1	5	0.5	100	100	1.21	345.53	1	На графе много остаточного феромона
	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	Для значений феромона каждую итерацию используются случайные числа
	20	10	100	1	1	5	10	100	100	1.17	466.37	1	
	20	10	100	1	1	5	0.5	-100	100	1.32	372.38	1	Уровень феромона не снижается
Q	20	10	100	1	1	5	0.5	-10	100	1.17	394.11	1	Замедление снижения феромона по сравн. с нормальной работой
	20	10	100	1	1	5	0.5	0	100	1.13	445.28	1	Нормальная работа алгоритма
	20	10	100	1	1	5	0.5	1	100	1.26	421.9	1	
	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	Алгоритм не выполняется
	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	
	20	10	100	1	1	5	0.5	100	-1	-	-	0	Нормальная работа алгоритма
Максимальное время	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	
	20	10	100	1	1	5	0.5	100	1000	11.4	419.67	1	
	20	10	100	1	1	5	0.5	100	5000	58.41	391.38	1	

												уменьшается кол-во остаточного феромона
--	--	--	--	--	--	--	--	--	--	--	--	--

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 / NETWORKX_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='#88AFFF',
                  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}")
# /PLOTING

# 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 dpq.does_item_exist("iteration_text"):
    dpq.configure_item("iteration_text", width=outputs_width)
if dpq.does_item_exist("shown_iteration_text"):
    dpq.configure_item("shown_iteration_text", width=outputs_width)
if dpq.does_item_exist("time_text"):
    dpq.configure_item("time_text", width=outputs_width)
if dpq.does_item_exist("networkx_seed"):
    dpq.configure_item("networkx_seed", width=outputs_width)
if dpq.does_item_exist("is_solution_found"):
    dpq.configure_item("is_solution_found", width=outputs_width)

if dpq.does_item_exist("plot_best_length"):
    dpq.configure_item("plot_best_length",
                      width=plots_width, height=plots_height)

def read_data_blob_from_ui():
    global data_blob
    data_blob['max_cities'] = dpq.get_value("num_cities")
    data_blob['max_ants'] = dpq.get_value("num_ants")
    data_blob['max_distance'] = dpq.get_value("max_distance")
    data_blob['init_pheromone'] = dpq.get_value("init_pheromone")
    data_blob['alpha'] = dpq.get_value("alpha")
    data_blob['beta'] = dpq.get_value("beta")
    data_blob['rho'] = dpq.get_value("rho")
    data_blob['qval'] = dpq.get_value("qval")
    data_blob['max_time'] = dpq.get_value("max_time")
    data_blob['networkx_seed'] = dpq.get_value("networkx_seed")
    data_blob['stop_on_change'] = dpq.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
    dpq.set_value("num_cities", data_blob['max_cities'])
    dpq.set_value("num_ants", data_blob['max_ants'])
    dpq.set_value("max_distance", data_blob['max_distance'])
    dpq.set_value("init_pheromone", data_blob['init_pheromone'])
    dpq.set_value("alpha", data_blob['alpha'])
    dpq.set_value("beta", data_blob['beta'])
    dpq.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 dpq.theme(tag="plot_theme_best_length"):
    with dpq.theme_component(dpq.mvLineSeries):
        dpq.add_theme_color(
            dpq.mvPlotCol_Line, (255, 99, 99),
            category=dpq.mvThemeCat_Plots)
        dpq.add_theme_style(
            dpq.mvPlotStyleVar_Marker, dpq.mvPlotMarker_None,
            category=dpq.mvThemeCat_Plots)
        dpq.add_theme_style(
            dpq.mvPlotStyleVar_MarkerSize, 0,
            category=dpq.mvThemeCat_Plots)
        dpq.add_theme_style(
            dpq.mvPlotStyleVar_LineWeight, 3.0,
            category=dpq.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 dpkg.group(horizontal=True):
    with dpkg.group():
        # Max city count
        dpkg.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
        dpkg.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
        dpkg.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
        dpkg.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
        dpkg.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
        dpkg.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()

```