Import tkinter as tk

From tkinter import ttk

Import networkx as nx

Import matplotlib.pyplot as plt

Import numpy as np

Import random

Import json

# Параметры алгоритма

NUM\_ANTS = 10

NUM\_ITERATIONS = 100

ALPHA = 1.0

BETA = 2.0

EVAPORATION\_RATE = 0.5

Q = 100

# Глобальные переменные для графа и феромонов

G = nx.Graph()

Pheromones = {}

# Функции работы с графом и алгоритмом

Def create\_graph\_with\_cargo\_type(edges, cargo\_type):

G.clear()

For edge in edges:

Node1, node2, base\_weight = edge

Weight = base\_weight

If cargo\_type == “Неоднородный”:

Weight \*= 1.5

G.add\_edge(node1, node2, weight=weight)

Return G

Def initialize\_pheromones(G):

Pheromones.clear()

For edge in G.edges():

Pheromones[edge] = 1.0

Def select\_next\_node(G, pheromones, current\_node, visited):

Neighbors = list(G.neighbors(current\_node))

Probabilities = []

For neighbor in neighbors:

If neighbor in visited:

Probabilities.append(0)

Else:

Edge = (current\_node, neighbor) if (current\_node, neighbor) in pheromones else (neighbor, current\_node)

Pheromone\_level = pheromones[edge] \*\* ALPHA

Distance = G[current\_node][neighbor][‘weight’] \*\* BETA

Probabilities.append(pheromone\_level / distance)

Total = sum(probabilities)

If total == 0:

Return None

Probabilities = [p / total for p in probabilities]

Return random.choices(neighbors, weights=probabilities, k=1)[0]

Def ant\_travel(G, pheromones, start\_node):

Path = [start\_node]

Current\_node = start\_node

Visited = {start\_node}

While len(visited) < len(G.nodes):

Next\_node = select\_next\_node(G, pheromones, current\_node, visited)

If next\_node is None:

Break

Path.append(next\_node)

Visited.add(next\_node)

Current\_node = next\_node

Return path

Def update\_pheromones(G, pheromones, ant\_paths):

For edge in pheromones:

Pheromones[edge] \*= (1 – EVAPORATION\_RATE)

For path in ant\_paths:

Path\_length = sum(G[path[i]][path[i+1]][‘weight’] for I in range(len(path) – 1))

Pheromone\_to\_add = Q / path\_length if path\_length > 0 else 0

For I in range(len(path) – 1):

Edge = (path[i], path[i+1]) if (path[i], path[i+1]) in pheromones else (path[i+1], path[i])

Pheromones[edge] += pheromone\_to\_add

Def find\_shortest\_path(G, pheromones, start\_node):

Current\_node = start\_node

Path = [current\_node]

Visited = {current\_node}

While len(visited) < len(G.nodes):

Neighbors = [(neighbor, pheromones[(current\_node, neighbor)] if (current\_node, neighbor) in pheromones else pheromones[(neighbor, current\_node)]) for neighbor in G.neighbors(current\_node) if neighbor not in visited]

If not neighbors:

Break

Next\_node = max(neighbors, key=lambda x: x[1])[0]

Path.append(next\_node)

Visited.add(next\_node)

Current\_node = next\_node

Return path

Def display\_graph(G, pheromones=None):

Pos = nx.spring\_layout(G)

Nx.draw(G, pos, with\_labels=True, node\_size=700, node\_color=”lightblue”, font\_size=10, font\_weight=”bold”)

Labels = nx.get\_edge\_attributes(G, ‘weight’)

Nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

If pheromones:

Pheromone\_labels = {edge: f”{pheromones[edge]:.2f}” for edge in G.edges()}

Nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=pheromone\_labels, font\_color=’red’)

Plt.show()

Def display\_shortest\_path(G, shortest\_path):

Pos = nx.spring\_layout(G)

Nx.draw(G, pos, with\_labels=True, node\_size=700, node\_color=”lightblue”, font\_size=10, font\_weight=”bold”)

Path\_edges = [(shortest\_path[i], shortest\_path[I + 1]) for I in range(len(shortest\_path) – 1)]

Nx.draw\_networkx\_edges(G, pos, edgelist=path\_edges, edge\_color=”red”, width=2)

Labels = nx.get\_edge\_attributes(G, ‘weight’)

Nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

Plt.show()

Def run\_ant\_colony\_optimization():

Global NUM\_ANTS, NUM\_ITERATIONS, ALPHA, BETA, EVAPORATION\_RATE, G

NUM\_ANTS = int(num\_ants\_var.get())

NUM\_ITERATIONS = int(num\_iterations\_var.get())

ALPHA = float(alpha\_var.get())

BETA = float(beta\_var.get())

EVAPORATION\_RATE = float(evaporation\_rate\_var.get())

Cargo\_type = cargo\_type\_var.get()

Edges = [

(‘A’, ‘B’, 1.0),

(‘A’, ‘C’, 2.0),

(‘B’, ‘D’, 1.5),

(‘C’, ‘D’, 2.5),

(‘A’, ‘D’, 3.0)

]

G = create\_graph\_with\_cargo\_type(edges, cargo\_type)

Initialize\_pheromones(G)

For iteration in range(NUM\_ITERATIONS):

Ant\_paths = []

For \_ in range(NUM\_ANTS):

Start\_node = random.choice(list(G.nodes))

Path = ant\_travel(G, pheromones, start\_node)

Ant\_paths.append(path)

Update\_pheromones(G, pheromones, ant\_paths)

Display\_graph(G, pheromones)

Start\_node = ‘A’

Shortest\_path = find\_shortest\_path(G, pheromones, start\_node)

Log(“Кратчайший путь: “ + “ -> “.join(shortest\_path))

Display\_shortest\_path(G, shortest\_path)

# Интерфейс пользователя

Root = tk.Tk()

Root.title(“Ant Colony Optimization Settings”)

# Параметры алгоритма

Num\_ants\_var = tk.StringVar(value=”10”)

Num\_iterations\_var = tk.StringVar(value=”100”)

Alpha\_var = tk.StringVar(value=”1.0”)

Beta\_var = tk.StringVar(value=”2.0”)

Evaporation\_rate\_var = tk.StringVar(value=”0.5”)

Cargo\_type\_var = tk.StringVar(value=”Однородный”)

# Элементы интерфейса

Ttk.Label(root, text=”Количество муравьев:”).grid(column=0, row=0, sticky=”W”)

Ttk.Entry(root, textvariable=num\_ants\_var).grid(column=1, row=0)

Ttk.Label(root, text=”Число итераций:”).grid(column=0, row=1, sticky=”W”)

Ttk.Entry(root, textvariable=num\_iterations\_var).grid(column=1, row=1)

Ttk.Label(root, text=”Alpha (влияние феромонов):”).grid(column=0, row=2, sticky=”W”)

Ttk.Entry(root, textvariable=alpha\_var).grid(column=1, row=2)

Ttk.Label(root, text=”Beta (влияние расстояния):”).grid(column=0, row=3, sticky=”W”)

Ttk.Entry(root, textvariable=beta\_var).grid(column=1, row=3)

Ttk.Label(root, text=”Коэффициент испарения:”).grid(column=0, row=4, sticky=”W”)

Ttk.Entry(root, textvariable=evaporation\_rate\_var).grid(column=1, row=4)

Ttk.Label(root, text=”Тип груза:”).grid(column=0, row=5, sticky=”W”)

Cargo\_type\_dropdown = ttk.Combobox(root, textvariable=cargo\_type\_var, values=[“Однородный”, “Неоднородный”])

Cargo\_type\_dropdown.grid(column=1, row=5)

# Функции управления графом

Node\_var = tk.StringVar()

Ttk.Label(root, text=”Узел:”).grid(column=0, row=7, sticky=”W”)

Ttk.Entry(root, textvariable=node\_var).grid(column=1, row=7)

Ttk.Button(root, text=”Добавить узел”, command=lambda: G.add\_node(node\_var.get()) if node\_var.get() not in G else log(“Узел уже существует”)).grid(column=0, row=8)

Ttk.Button(root, text=”Удалить узел”, command=lambda: G.remove\_node(node\_var.get()) if node\_var.get() in G else log(“Узел не найден”)).grid(column=1, row=8)

Edge\_node1\_var = tk.StringVar()

Edge\_node2\_var = tk.StringVar()

Edge\_weight\_var = tk.StringVar()

# Элементы интерфейса для добавления и удаления рёбер

Ttk.Label(root, text=”Ребро (узел1 – узел2):”).grid(column=0, row=9, sticky=”W”)

Ttk.Entry(root, textvariable=edge\_node1\_var, width=5).grid(column=1, row=9, sticky=”W”)

Ttk.Entry(root, textvariable=edge\_node2\_var, width=5).grid(column=2, row=9, sticky=”W”)

Ttk.Label(root, text=”Вес:”).grid(column=3, row=9, sticky=”W”)

Ttk.Entry(root, textvariable=edge\_weight\_var, width=5).grid(column=4, row=9, sticky=”W”)

Ttk.Button(root, text=”Добавить ребро”, command=lambda: add\_edge()).grid(column=0, row=10)

Ttk.Button(root, text=”Удалить ребро”, command=lambda: remove\_edge()).grid(column=1, row=10)

# Функции для добавления и удаления рёбер

Def add\_edge():

Node1 = edge\_node1\_var.get()

Node2 = edge\_node2\_var.get()

Try:

Weight = float(edge\_weight\_var.get())

If node1 in G and node2 in G and node1 != node2:

G.add\_edge(node1, node2, weight=weight)

Log(f”Ребро {node1}-{node2} с весом {weight} добавлено”)

Else:

Log(“Узлы не найдены или совпадают”)

Except ValueError:

Log(“Вес должен быть числом”)

Display\_graph(G, pheromones)

Def remove\_edge():

Node1 = edge\_node1\_var.get()

Node2 = edge\_node2\_var.get()

If G.has\_edge(node1, node2):

G.remove\_edge(node1, node2)

Log(f”Ребро {node1}-{node2} удалено”)

Else:

Log(“Ребро не найдено”)

Display\_graph(G, pheromones)

# Кнопки для сохранения и загрузки графа

Ttk.Button(root, text=”Сохранить граф”, command=save\_graph).grid(column=0, row=11)

Ttk.Button(root, text=”Загрузить граф”, command=load\_graph).grid(column=1, row=11)

# Функции сохранения и загрузки графа

Def save\_graph():

Graph\_data = {

“nodes”: list(G.nodes),

“edges”: [(u, v, G[u][v][‘weight’]) for u, v in G.edges]

}

With open(“graph\_data.json”, “w”) as file:

Json.dump(graph\_data, file)

Log(“Граф сохранен в файл ‘graph\_data.json’”)

Def load\_graph():

Global G

Try:

With open(“graph\_data.json”, “r”) as file:

Graph\_data = json.load(file)

G.clear()

G.add\_nodes\_from(graph\_data[“nodes”])

G.add\_weighted\_edges\_from(graph\_data[“edges”])

Log(“Граф загружен из файла ‘graph\_data.json’”)

Display\_graph(G, pheromones)

Except FileNotFoundError:

Log(“Файл ‘graph\_data.json’ не найден”)

# Текстовая область для логов

Log\_text = tk.Text(root, height=10, width=50, state=’normal’)

Log\_text.grid(column=0, row=12, columnspan=5)

# Функция для отображения логов

Def log(message):

Log\_text.insert(tk.END, message + “\n”)

Log\_text.see(tk.END)

# Кнопка для запуска алгоритма

Ttk.Button(root, text=”Запустить”, command=run\_ant\_colony\_optimization).grid(column=0, row=6, columnspan=2)

# Запуск интерфейса

Root.mainloop()