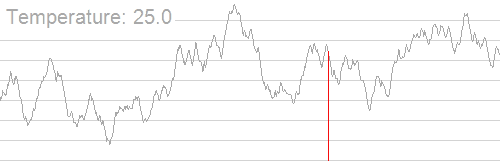
**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 4**

Алгори́тм имита́ции о́тжига (англ. Simulated annealing) — общий алгоритмический метод решения задачи глобальной оптимизации, особенно дискретной и комбинаторной оптимизации. Один из примеров методов Монте-Карло.

  
За основу взят процесс кристаллизации вещества.

Совокупность позиций всех атомов будем называть состоянием системы,

каждому состоянию соответствует определенный уровень энергии.

Цель отжига – привести систему в состояние с наименьшей энергией. Чем

ниже уровень энергии, тем «лучше» кристаллическая решетка, т.е. тем меньше у

нее дефектов и прочнее металл.

В ходе «отжига» металл сначала нагревают до

некоторой температуры, что заставляет атомы

кристаллической решетки покинуть свои позиции. Затем

начинается медленное и контролируемое охлаждение.

Атомы стремятся попасть в состояние с меньшей

энергией, однако, с определенной вероятностью они

могут перейти и в состояние с большей. Эта вероятность

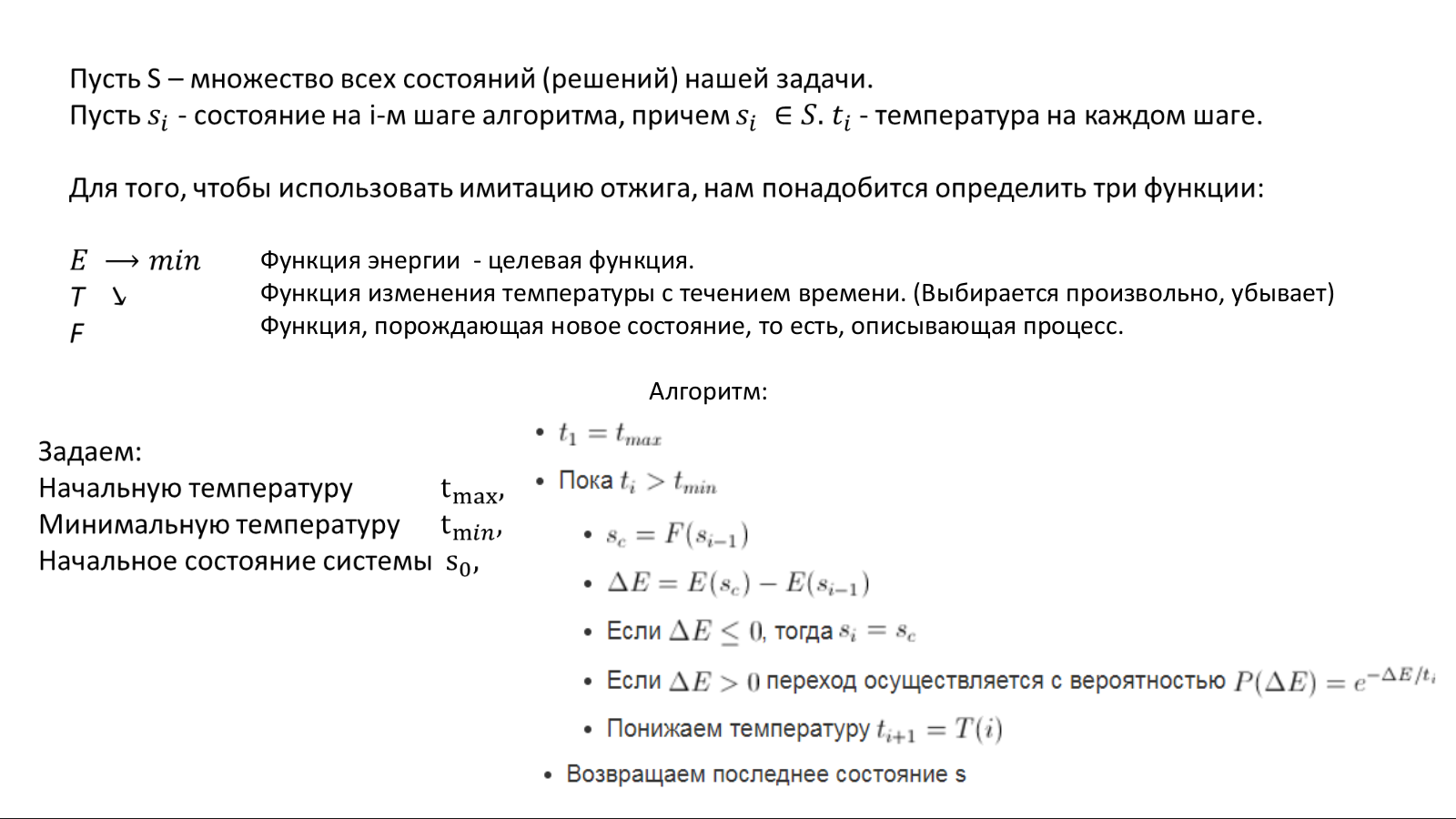
уменьшается вместе с температурой. Переход в худшее

состояние помогает в итоге отыскать состояние с

энергией меньшей, чем начальная. Процесс завершается,

когда температура падает до заранее заданного

значения.

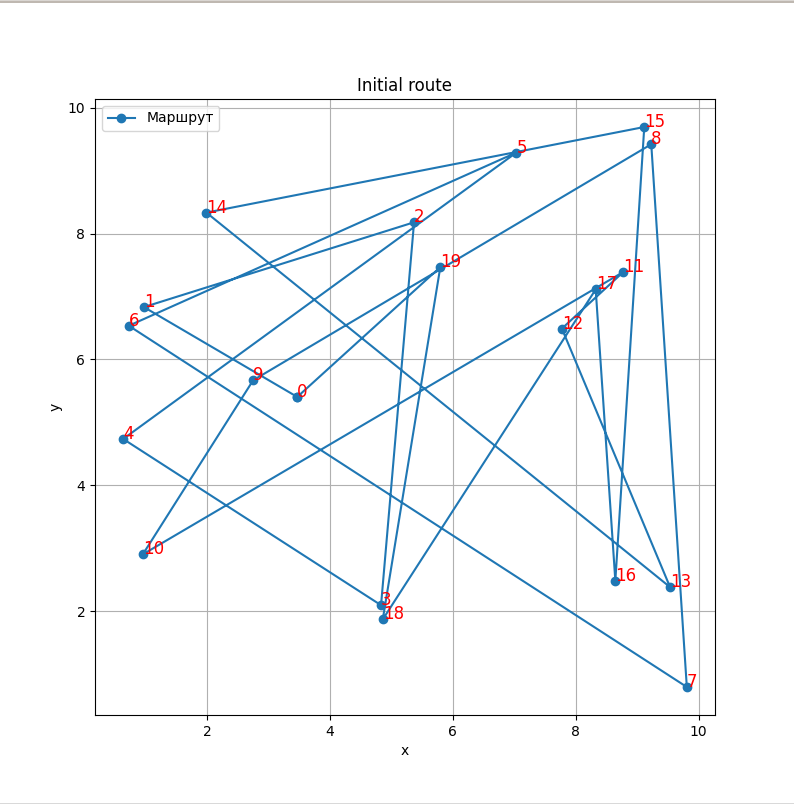
  
Задача коммивояжёра (или TSP от англ. travelling salesman problem) — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

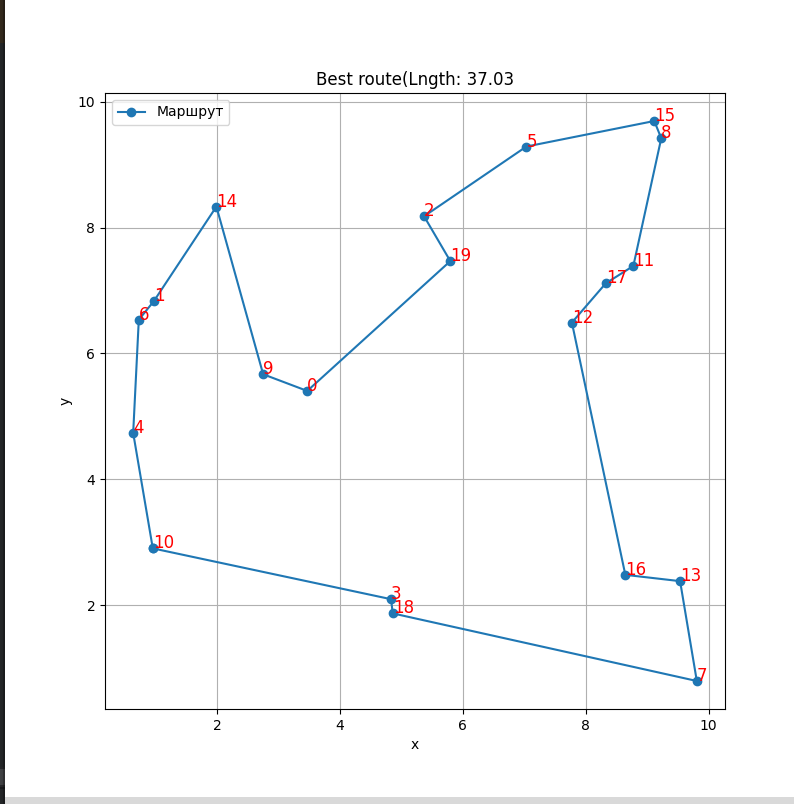


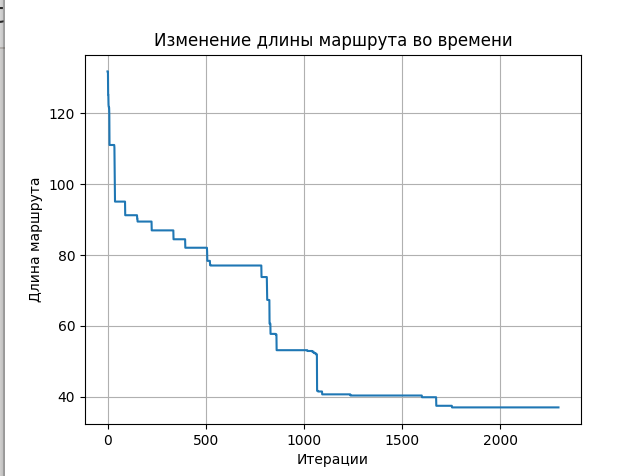
**Код:**

import numpy as np  
import matplotlib.pyplot as plt  
  
# Функция для вычисления евклидова расстояния между городами  
def distance(city1, city2):  
 return np.linalg.norm(city1 - city2)  
  
# Функция для вычисления общей длины маршрута  
def total\_distance(route, cities):  
 return sum(distance(cities[route[i]], cities[route[i+1]])  
 for i in range(N-1)) + distance(cities[route[-1]], cities[route[0]])  
  
# Функция для генерации нового состояния (переворот участка маршрута)  
def generate\_new\_state(route):  
 i, j = sorted(np.random.choice(N, 2, replace=False))  
 new\_route = route.copy()  
 new\_route[i:j+1] = np.flip(route[i:j+1])  
 return new\_route  
  
  
# Алгоритм имитации отжига  
def simulated\_annealing(cities, tmax=100, tmin=1e-3, alpha=0.995, max\_iter=100000):  
 # Инициализация  
 route = np.arange(N)  
 np.random.shuffle(route)  
 best\_route = route.copy()  
 best\_distance = total\_distance(best\_route, cities)  
  
 current\_route = route.copy()  
 current\_distance = best\_distance  
  
 T = tmax  
 distances = []  
  
 for k in range(max\_iter):  
 if T < tmin:  
 break  
  
 # Генерация нового состояния  
 new\_route = generate\_new\_state(current\_route)  
 new\_distance = total\_distance(new\_route, cities)  
  
 # Разница в расстояниях  
 delta\_E = new\_distance - current\_distance  
  
 # Принимаем новое состояние с вероятностью  
 if delta\_E < 0 or np.random.rand() < np.exp(-delta\_E / T):  
 current\_route = new\_route  
 current\_distance = new\_distance  
  
 # Обновление лучшего маршрута  
 if current\_distance < best\_distance:  
 best\_route = current\_route  
 best\_distance = current\_distance  
  
 # Снижение температуры  
 T \*= alpha  
 distances.append(best\_distance)  
  
 return best\_route, best\_distance, distances  
  
  
# график  
def plot\_route(cities, route, title):  
 plt.figure(figsize=(8, 8))  
 route\_cities = cities[route]  
 route\_cities = np.vstack((route\_cities, route\_cities[0])) # Замыкаем маршрут  
 plt.plot(route\_cities[:, 0], route\_cities[:, 1], 'o-', label='Маршрут')  
 for i, (x, y) in enumerate(cities):  
 plt.text(x, y, str(i), fontsize=12, color='red')  
 plt.title(title)  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.grid(True)  
 plt.legend()  
 plt.show()  
  
  
N = 20 #города  
cities = np.random.rand(N, 2) \* 10 #Координаты городов в квадрате 10х10  
#параметры  
tmax = 100  
tmin = 1e-3  
alpha = 0.995  
max\_iterations = 100000  
  
best\_route, best\_distance, distances = simulated\_annealing(cities, tmax, tmin, alpha)  
plot\_route(cities, np.arange(N), 'Initial route')  
plot\_route(cities, best\_route, f'Best route(Lngth: {best\_distance:.2f}')  
  
# График изменения длины маршрута в процессе оптимизации  
plt.plot(distances)  
plt.title('Изменение длины маршрута во времени')  
plt.xlabel('Итерации')  
plt.ylabel('Длина маршрута')  
plt.grid(True)  
plt.show()

**Графики:**

****

****

****