Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
Национальный исследовательский университет
«Высшая Школа Экономики»

Факультет экономических наук

Образовательная программа «Экономика»

КУРСОВАЯ РАБОТА

Стохастические методы оптимизации

Выполнил:
студент группы БЭК1812
Хайкин Глеб Алексеевич

Научный руководитель:
старший преподаватель
Борзых Дмитрий Александрович

Москва — 2020

Оглавление

1. Введение

jjj.

2. Методы и их применение

2.1. Имитация отжига

Имитация отжига (simulated annealing) представляет собой алгоритм решения задачи по поиску глобального оптимума некоторой функции $F : \mathbb{X} \to \mathbb{R}$ через упорядоченный стохастический поиск, базирующийся на моделировании физического процесса кристализации вещества из жидкого состояние в твердое.

Для описания метода рассмотрим задачу нахождения глобального минимума:

$$F(x) \to \min_{x \in \mathbb{X}},$$

где $x = (x_1, \ldots, x_m)$ — вектор всех состояний, $\mathbb{X}$ — множество всех состояний.

2.1.1. Алгоритм

Положим, что $k = 0$ и изначально температура зафиксированна на определенном уровне $T(k) = const$.

1. Из множества всех состояний выберем случайный элемент $\widehat{x}(k) \equiv x_i$,

   $i \in (1, ..., m)$.
2. Понизим температуру одним из следующих способов:

   а) Больцмановский отжиг

   $$T(k) = \frac{T(0)}{\ln(1+k)},\ k > 0 \tag{2.1}$$

   б) Отжиг Коши

   $$T(k) = \frac{T(0)}{k} \tag{2.2}$$

   в) Метод тушения

   $$T(k+1) = \alpha T(k),\ \alpha \in (0, 1) \tag{2.3}$$
3. Пусть следующий элемент зависит от функции из семейства симметричных вероятностных распределений $G : \mathbb{X} \to \mathbb{X}$, порождающей новое состояние:

   $$\tilde{x}(k) \sim G(\widehat{x}(k), T(k)).$$

   а) Часто G выбирается из семейства нормальных распределений:

   $$G(\tilde{x}; \widehat{x}, T) = \frac{1}{\sqrt{(2\pi)^D T}} \exp\left\{\frac{-|\tilde{x} - \widehat{x}|^2}{2T}\right\}, \tag{2.4}$$

где $\widehat{x}$ — математическое ожидание, T — дисперсия, D — размерность пространства всех состояний.

б) Также для D = 1 используется распределение Коши с плотностью:

$$G(\tilde{x}; \widehat{x}, T) = \frac{1}{\pi} \frac{T}{|\tilde{x} - \widehat{x}|^2 + T^2}, \tag{2.5}$$

где $\widehat{x}$ — параметр сдвига, T — параметр масштаба.

4. Рассчитываем разницу двух функций:

$$\Delta F = F(\tilde{x}(k)) - F(\widehat{x}(k)).$$

5. Принимаем $\tilde{x}(k)$ за новый элемент, то есть $\widehat{x}(k+1) \equiv \tilde{x}(k)$, с вероятностью

$$\mathbb{P}(\{\widehat{x}(k+1) = \tilde{x}(k)\}) = \begin{cases} 1, & \Delta F < 0 \\ \exp\left\{-\dfrac{\Delta F}{T(k)}\right\}, & \Delta F \geqslant 0 \end{cases} \tag{2.6}$$

и отвергаем его, то есть $\widehat{x}(k+1) \equiv \widehat{x}(k)$, с вероятностью

$$q = 1 - \mathbb{P}(\{\widehat{x}(k+1) = \tilde{x}(k)\}).$$

Заметим, чем выше температура, тем больше вероятность принять состояние хуже текущего ($\Delta F \geqslant 0$).

6. Возвращаемся к пункту 2, пока не достигнем глобального минимума.

2.1.2. N ферзей

Рассмотрим задачу, в которой необходимо расставить N ферзей на шахматной доске размера $N \times N$ так, чтобы ни один из них не «бил» другого.

В таком случае, множество всех состояний $\mathbb{X}$ будет содержать всевозможные расстановки ферзей на шахмотной доске. Общее число возможных расположений n ферзей на $N \times N$-клеточной доске равно:

$$\binom{N \times N}{n} = \frac{N \times N!}{n!(N \times N - n)!}$$

Тогда функция $F : \mathbb{X} \to \mathbb{R}$ будет выдавать количество атак ферзей, и решением данной задачи будет нахождение такого распложения x^*, что $F(x^*) \equiv 0$.

Зафиксируем изначальное расположение ферзей на шахматной доске. Очевидно, что несколько ферзей не могут находиться на одной вертикали или горизонтали, ибо тогда они будут находиться под ударом друг-друга. Следовательно, наша задача сужается к поиску расположения:

$$x* = (q_1, ..., q_n) = \{(1, h_1), ..., (n, h_n)\}, h_1 \neq ... \neq h_n, \tag{2.7}$$

где (i, h_i) — расположение ферзя q_i на i-ой вертикали по горизонтали h_i. Отметим, что такая задача имеет $N!$ решений.

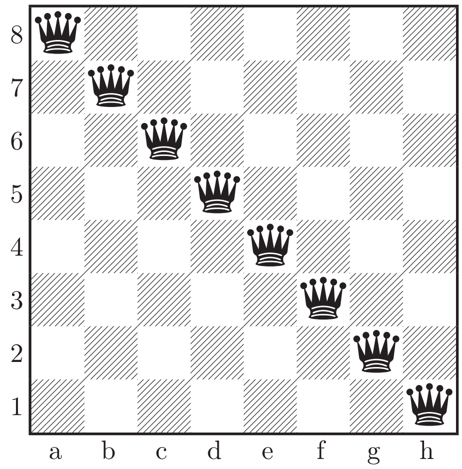

Рисунок 2.1 — Изначальное расположение.

Определим функцию, которая будет создавать изначальное неоптимальное расположение, в общем виде. Учитем, что несколько ферзей не могут находиться на одной вертикали или горизонтали.

In:
```
def queens(N):
  ver = np.arange(1, N + 1)
  hor = np.arange(1, N + 1)
  np.random.shuffle(hor)
  return np.column_stack((ver, hor)) # получаем массив
  # размерности (N, 2), отождествляющий расположение ферзей
```

Выведем первоначальное расположение ферзей для стандартной доски 8×8, где первый столбец массива — расположение по вертикали, второй столбец массива — расположение по горизонтали. Для наглядности — презентации оптимизационного процесса — выстроим изначальную расстановку на главной диагонали (рис. 2.1).

In:
```
matrix = queens(8)
matrix
```

Out:

```
array([[1, 1],
       [2, 2],
       [3, 3],
       [4, 4],
       [5, 5],
       [6, 6],
       [7, 7],
       [8, 8]])
```

Функция F, которая выявляет количество атак ферзей, выглядит следующим образом:

In:

```
def F(Q, N):
  cnt = 0
  for i in range(N):
     for j in range(i + 1, N):
         if abs(Q[i, 0] - Q[j, 0]) == abs(Q[i, 1] - Q[j, 1]):
             cnt += 1
  return cnt * 2 # учитываем взаимные атаки
```

Посмотрим, сколько у атак у исходной расстановки.

In:

```
F(matrix, 8)
```

Out:

```
56
```

В нашей задачи функция G будет случайной незначительной перестановкой номеров горизонтали в исходном наборе.

In:

```
def G(Q, N):
  pos = Q.copy()
  while True:
     i = np.random.randint(0, N - 1)
     j = np.random.randint(0, N - 1)
     if i != j:
        break
  pos[i, 1], pos[j, 1] = pos[j, 1], pos[i, 1]
  return pos # получаем новое расположение
```

Теперь выведем и сам метод имитации отжига.

In:

```
def SA(Q, T, schedule):
  N = np.shape(Q)[0]
  x_hat = Q.copy()
  while F(x_hat, N) != 0:
     x_tilda = G(x_hat, N)
     delta = F(x_tilda, N) - F(x_hat, N)
     prob = np.exp(- delta / T)
```

```python
        if (delta < 0) or (prob >= np.random.random()):
            x_hat = x_tilda
        T *= schedule # используем метод тушения для понижения температуры
    return x_hat
```

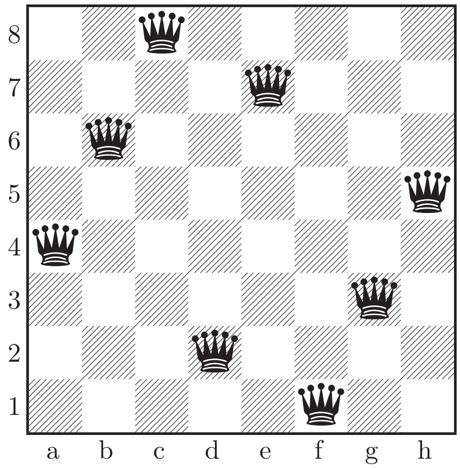

Рисунок 2.2 — Оптимальное расположение

Так для нашего примера с гиперпараметрами $T(0) = 100, \alpha = 0.9$ мы получаем следующее оптимальное решение (рис. 2.2):

In:
```python
SA(matrix, 100, 0.9)
```

Out:
```
array([[1, 4],
       [2, 6],
       [3, 8],
       [4, 2],
       [5, 7],
       [6, 1],
       [7, 3],
       [8, 5]])
```

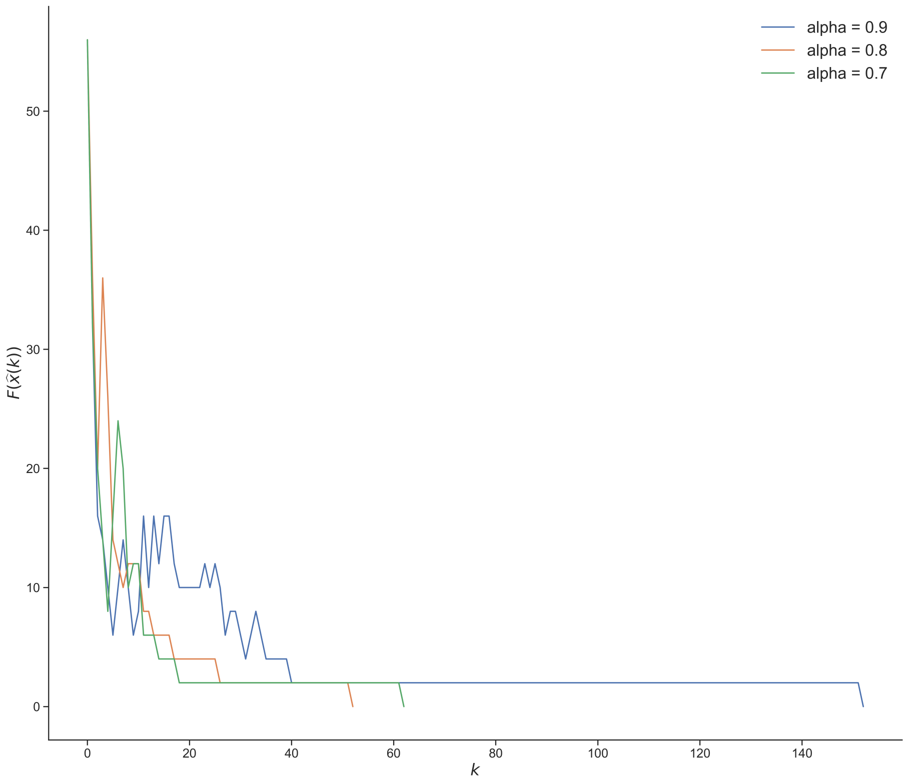

Рисунок 2.3 — Оптимизация расстановки 8 ферзей в зависимости от гиперпараметра α.

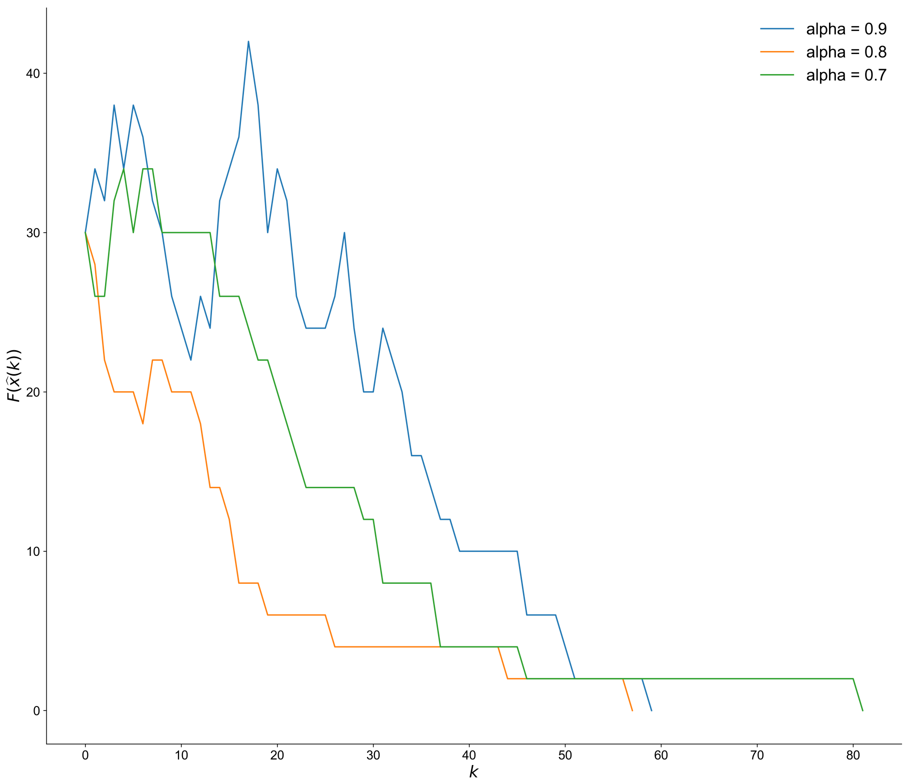

Рисунок 2.4 — Оптимизация расстановки 25 ферзей в зависимости от гиперпараметра α.

2.1.3. Минимизация негладкой функции

Воспользуемся алгоритмом имитации отжига для нахождения глобального минимума следующей функции:

$$f(x) = x^2(1 + |\sin 80x|).$$

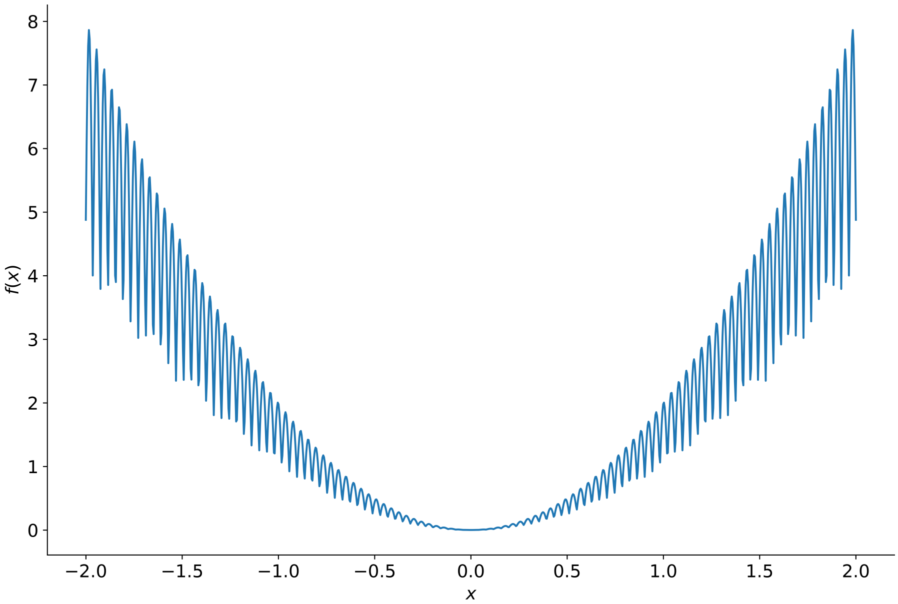

Рисунок 2.5

Стандартные методы оптимизации — к примеру, метод градиентного спуска — в данном случае не применимы. Вследствие наличия модуля этп функция не дифференцируема. Также она имеет очень большое количество локальных минимумов, что затрудняет, к примеру, мультистарт — запуск градиентного спуска из разных начальных направлений.

Применим наш алгоритм к данной задачи. Пусть $T(0) = 0.6$. Для понижения температуры будем использовать Больцмановский отжиг (2.1), а в качестве функции вероятностных распределений G — семейство нормальных распределений (2.4).

In:

```
def SA(space, T, epsilon): # за space берется np.linspace(-2, 2, 1000)
  x_hat = np.random.choice(space)
  T_0 = T
  k = 1
  while True:
     x_tilda = np.random.normal(x_hat, T)
     delta = F(x_tilda) - F(x_hat)
     prob = np.exp(- delta / T)
     if (delta < 0) or (prob >= np.random.random()):
        x_hat = x_tilda
```

```
        if (x_hat < epsilon) and (x_hat > 0):
            return x_hat
        T = T_0 / np.log(1 + k)
        k += 1
```

Остановка итерационного процесса и скорость метода зависят от того, насколько близко мы хотим приблизиться к глобальному минимуму. Так, при точности 10^{-1}, что довольно много, для 1000 повторений алгоритма метод отжига находит глобальный минимум в среднем за 1.97e-3 секунды со стандартным отклонением в 3.47e-5 секунды. Однако, увеличив точность до 10^{-6}, среднюю скорость занимает уже 1.27 секунды со стандартным отклонением в 2.1e-5 секунды. Это наглядно представлено на рисунке 2.6.

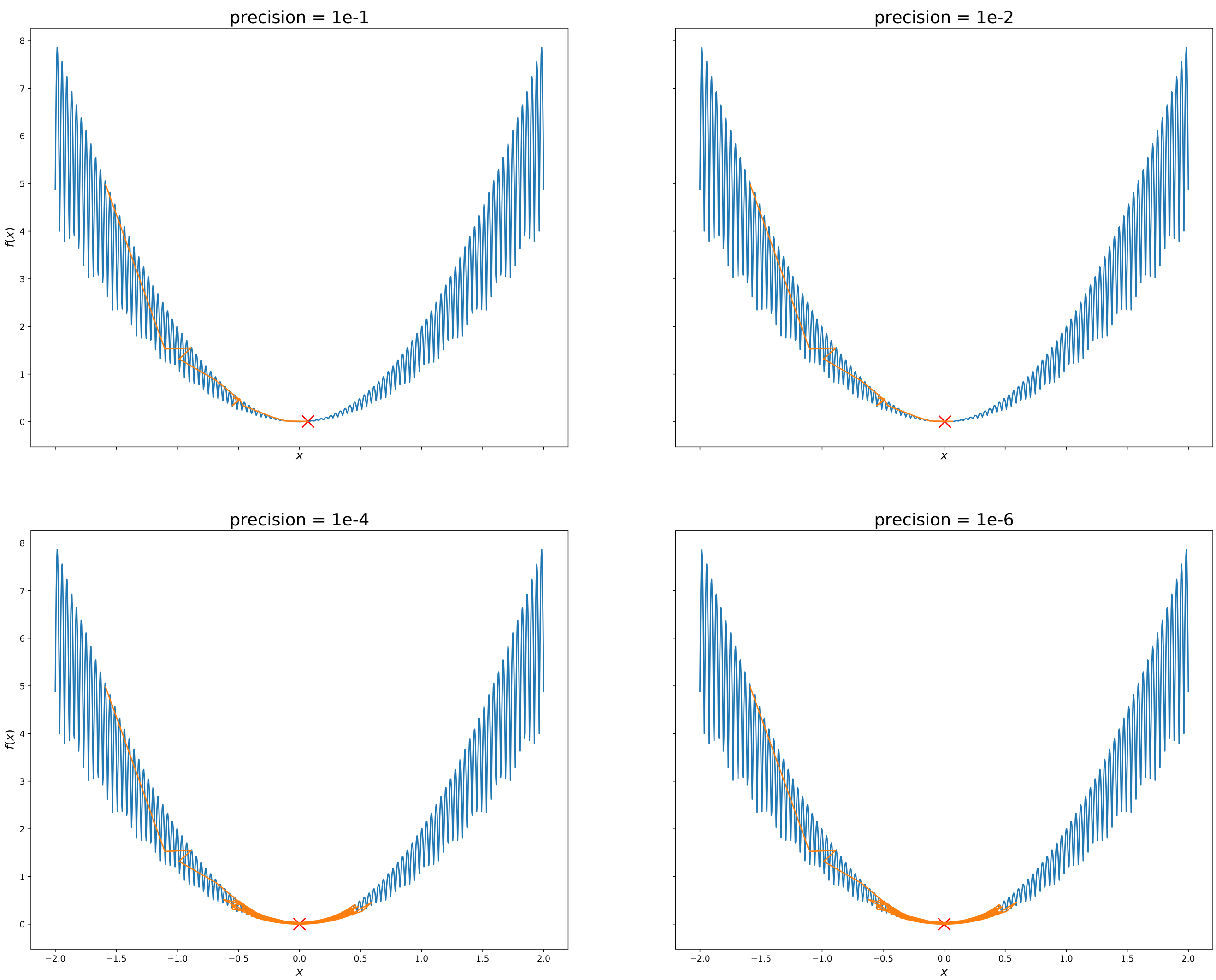

Рисунок 2.6 — Оптимизационный процесс в зависимости от точности.

2.1.4. Задача коммивояжера

Задача коммивояжера или TSP (Traveling Salesman Problem) является образцовым методом проверки многих оптимизационных алгоритмов и заключается в поиске кратчайшего маршрута между городами. Путь должен быть проложен так, чтобы маршрут единственно проходил через все города и его конечная точка совпадала с изначальной.

TSP имеет множество приложений в планировании и логистике, а также выступает в качестве подзадачи во многих других областях. В таком случае города могут представлять, к примеру, клиентов, а расстояние между городами — время или стоимость путешествия.

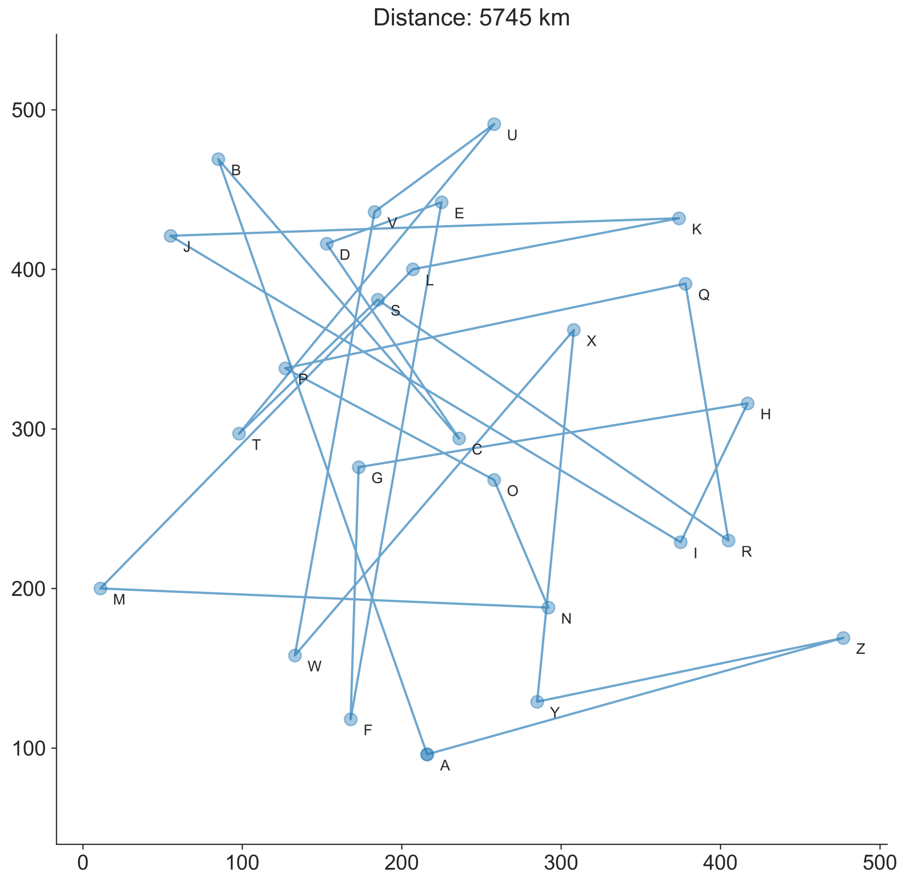

Рисунок 2.7 — Изначальный маршрут для 26-ти городов.

Для нашего примера создадим карту. Функция map_city будет принимать желаемое количество городов в качестве входных данных и выдавать два списка, из которых далее создается общий список кортежей. Первым элементом кортежа является наименование города, а вторым — его расположение в декартовой системе координат.

In:
```python
def map_city(cities_num):
  letters = [chr(i) for i in range(65, 65 + cities_num)]
  coord = np.random.randint(1, 500, size=(cities_num, 2))
  return letters, coord
```

Наш маршрут будет состоять из 26-ти городов.

In:
```python
names, cities = map_city(26)
store_val = list(zip(names, cities))
```

Определим расстояние от города i до всех остальных посредством функции distance_dict. Для измерения расстояния между городами будем использовать евклидову метрику. Напомним, что евклидово расстояние между точками $x = (x_1, ..., x_d)$ и $u = (u_1, ..., u_d)$ задается как:

$$d(x, u) = \sqrt{\sum_{i=1}^{d}(x_i - u_i)^2}$$

In:
```python
def distance_dict(cities, n):
  d = dict()
  for i in range(n):
     city = dict()
     for j in range(n):
        if i == j:
           continue
        c_a = cities[i][1]
        c_b = cities[j][1]
        dist = np.sqrt((c_a[0] - c_b[0]) ** 2 + (c_a[1] - c_b[1]) ** 2)
        city[cities[j][0]] = dist
     d[cities[i][0]] = city
  return d
```

In:
```python
cities_d = distance_dict(store_val, len(store_val))
```

Функция F для подсчета общего расстояния путешествия:

In:
```python
def F(path, cities):
  dist = 0
  for i in range(len(path) - 1):
     dist += cities[path[i]][path[i + 1]]
  dist += cities[path[i + 1]][path[0]]
  return dist
```

За функцию G будет выступать простая перестановка как и в задаче о N ферзях.

In:
```
def G(path, n):
  pos = path.copy()
  while True:
     i = np.random.randint(0, n - 1)
     j = np.random.randint(0, n - 1)
     if i != j:
        break
     pos[i], pos[j] = pos[j], pos[i]
  return pos
```

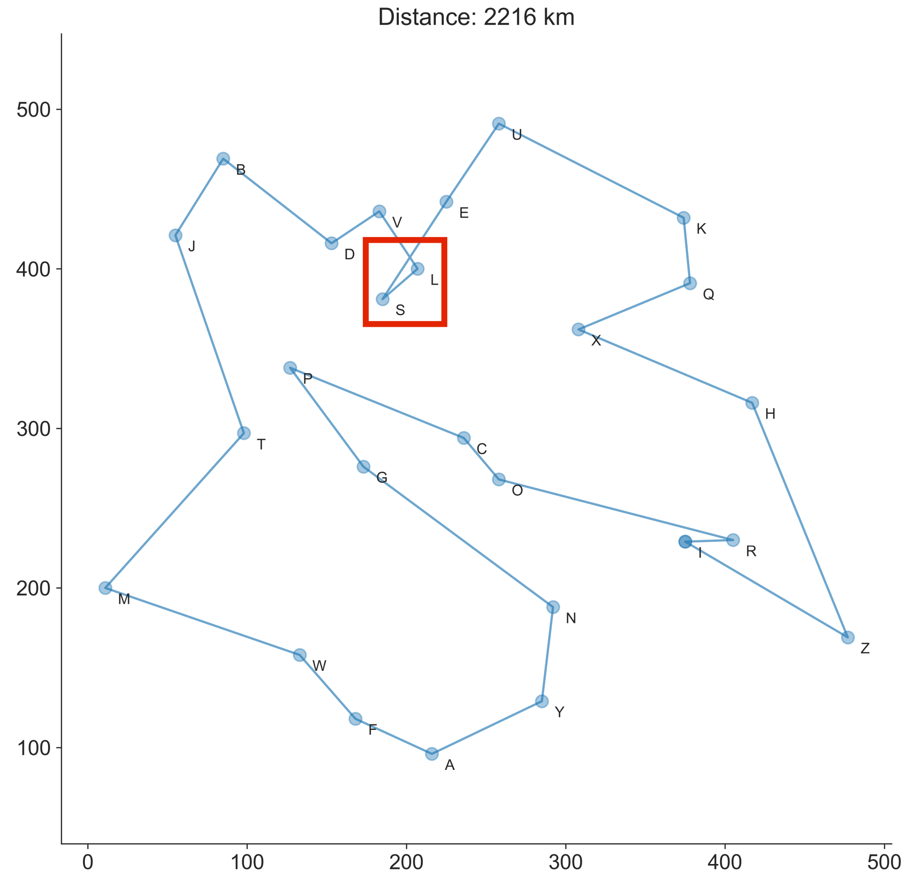

Рисунок 2.8 — Применение метода отжига для построения оптимального маршрута для 26-ти городов.

Для понижения температуры используем Больцмановский отжиг (2.1).

In:
```
def SA(path, T):
  path_hat = path
  n = len(path_hat)
  np.random.shuffle(path_hat)
  T_0 = T
  k = 1
  for i in range(100000):
     path_tilda = G(path_hat, n)
     delta = F(path_tilda, cities_d) - F(path_hat, cities_d)
     prob = np.exp(- delta / T)
```

Теперь построим оптимальный маршрут (рис. 2.8).

In:
```
path_opt = SA(names, 100)
```

Несмотря на то, что метод отжига сократил преодалеваемую дистанцию, TSP была решена неидеально. К примеру, можно выделить соединение вершин E-S-L-V: очевидно, оно не оптимально, поскольку путь E-L-S-V имеет меньшее расстояние.

2.1.5. Вывод

Заключение

Основные результаты работы заключаются в следующем.

1. На основе анализа ...
2. Численные исследования показали, что ...
3. Математическое моделирование показало ...
4. Для выполнения поставленных задач был создан ...

И какая-нибудь заключающая фраза.

Последний параграф может включать благодарности. В заключение автор выражает благодарность и большую признательность научному руководителю Иванову И. И. за поддержку, помощь, обсуждение результатов и научное руководство. Также автор благодарит Сидорова А. А. и Петрова Б. Б. за помощь в работе с образцами, Рабиновича В. В. за предоставленные образцы и обсуждение результатов, Занудятину Г. Г. и авторов шаблона *Russian-Phd-LaTeX-Dissertation-Template* за помощь в оформлении диссертации. Автор также благодарит много разных людей и всех, кто сделал настоящую работу автора возможной.

Список литературы

[1] *Шамин Р.В.* (2019) Практическое руководство по машинному обучению. — М.: Научный канал. — 93 с.