

巡回セールスマン問題 (TSP:Traveling Salesman Problem)

smat1957@gmail.com^{*1}

2025 年 4 月 23 日

^{*1} <https://altema.is.tohoku.ac.jp/QA4U3/>

第 1 章

量子ニーリングの基礎

[1] 西森秀稔、大関真之 著「量子アニーリングの基礎」共立出版より 2.6.1

1.1 問題

予め決められた地点を全て 1 度ずつ訪れて元の地点に戻ってくるための最短経路を探す。

例えば、A,B,C,D,E の 5 つの地点があって、仮に A からスタートすると、次の地点は B,C,D,E の 4 箇所の中から選ばなければならない。B を選んだら次は C,D,E の 3 通り。その次は 2 通り。最終的には $4 \times 3 \times 2 \times 1 = 24$ 通りの選び方がある事になる。訪れる地点の数が N 地点だと、 $(N-1) \times (N-2) \times \dots \times 2 \times 1 = (N-1)!$ 通りになる。 N が大きいと総当たりで経路を探すことは困難であり、巡回セールスマン問題は「NP 困難問題」に分類されている。

1.2 巡回セールスマン問題をイジング模型で表す

まず、 $N \times N$ の表を考え、横方向に地点の名前 (A,B,C,D,E)、縦方向には何番目に訪れるかを割り当てる。

表 1.1 表のタイトル

	A	B	C	D	E
1 番目	1	0	0	0	0
2 番目	0	0	1	0	0
3 番目	0	1	0	0	0
4 番目	0	0	0	1	0
5 番目	0	0	0	0	1
1 番目	1	0	0	0	0

セールスマンが訪れる箇所には 1 を、そうでない所には 0 を置く。上の表の例では、 $A \rightarrow C \rightarrow B \rightarrow D \rightarrow E \rightarrow A$ という経路に対応している。

式で表現するために、表の各箇所に対応した 2 値変数 $q_{\alpha i}$ を割り当ててセールスマンの辿る経路を表現する。 α は地点名 (A, B, C, ...) を、 i は巡る順番 (1, 2, 3, ...) を表している。

地点 α と β の間の距離 $d_{\alpha\beta}$ が予め与えられているとすると、セールスマンが巡る全経路長 L は、

$$L = \sum_{\alpha, \beta} \sum_{i=1}^N d_{\alpha\beta} q_{\alpha,i} q_{\beta,i+1}$$

$q_{\alpha,i}$ は2値変数 (0 か 1) なので、 $q_{\alpha,i}$ と $q_{\beta,i+1}$ の両方が1の場合に限り、 $d_{\alpha\beta}$ が L に加算される事になる。この経路長 L を最小にする $\{q_{\alpha,i}\}$ (0 か 1 か)、を選ぶことになる。

ただし、

- 各地点には1度しか訪れない (表では、各列に1は1つだけ)
→ 各 α において、 $(\sum_i q_{\alpha,i} - 1)^2 = 0$
- 各時点で訪れる地点は1箇所だけ (表では、各行に1は1つだけ)
→ 各 i において、 $(\sum_{\alpha} q_{\alpha,i} - 1)^2 = 0$

という2つの制約が課された上で、 L が最小になる様に $\{q_{\alpha,i}\}$ を選ばなければならない。

以上の考察より、目的関数全体の H は次の様になる

$$H = \sum_{\alpha, \beta} \sum_i d_{\alpha\beta} q_{\alpha,i} q_{\beta,i+1} + \lambda \sum_{\alpha} \left(\sum_i q_{\alpha,i} - 1 \right)^2 + \lambda \sum_i \left(\sum_{\alpha} q_{\alpha,i} - 1 \right)^2$$

λ は正の定数。

1.3 式の展開

$$\begin{aligned} H &= \lambda \sum_{\alpha} \left(\left(\sum_i q_{\alpha,i} \right)^2 - 2 \sum_i q_{\alpha,i} \right) + \lambda \sum_i \left(\left(\sum_{\alpha} q_{\alpha,i} \right)^2 - 2 \sum_{\alpha} q_{\alpha,i} \right) + \sum_{\alpha, \beta} \sum_i d_{\alpha\beta} q_{\alpha,i} q_{\beta,i+1} \\ &= \lambda \sum_{\alpha} \left(\sum_i q_{\alpha,i}^2 + 2 \sum_{i,j} q_{\alpha,i} q_{\alpha,j} - 2 \sum_i q_{\alpha,i} \right) + \lambda \sum_i \left(\sum_{\alpha} q_{\alpha,i}^2 + 2 \sum_{\alpha, \beta} q_{\alpha,i} q_{\beta,i} - 2 \sum_{\alpha} q_{\alpha,i} \right) \\ &\quad + \sum_{\alpha, \beta} \sum_i d_{\alpha\beta} q_{\alpha,i} q_{\beta,i+1} \\ &= -\lambda \sum_{\alpha} \left(\sum_i q_{\alpha,i} - 2 \sum_i \sum_j q_{\alpha,i} q_{\alpha,j} \right) - \lambda \sum_i \left(\sum_{\alpha} q_{\alpha,i} - 2 \sum_{\alpha} \sum_{\beta} q_{\alpha,i} q_{\beta,i} \right) + \sum_{\alpha, \beta} \sum_i d_{\alpha\beta} q_{\alpha,i} q_{\beta,i+1} \end{aligned}$$

ここで q は2値変数なので、 $q^2 = q$ が成り立つ。また、定数は最小化では無視できる。

1.4 実装

```
from openjij import SASampler, SQASampler
from collections import defaultdict, Counter
import numpy as np

class TSP:
    def __init__(self, city, cost_matrix):
        samplers = [SASampler(), SQASampler]
        self.sampler = samplers[0]
        self.cities = len(city)
```

```

        self.cost_matrix = np.array(cost_matrix)

    def gen_qubo1(self, cities):
        qubo_size = cities * cities
        Q1 = np.zeros((qubo_size, qubo_size))
        # u, v は訪れる都市。i, j は巡回の順番
        indices = [(u, v, i, j) for u in range(cities) for v in range(cities) for i
in range(cities) for j in range(cities)]
        for u,v,i,j in indices:
            ui = u * cities + i
            vj = v * cities + j
            #print(u, v, i, j, ui, vj)
            if ui>vj: # 上三角だけ
                continue
            if ui==vj: # 対角要素 \sum\_alpha\sum\_i
                Q1[(ui, vj)] -= 2
            if u==v and i!=j: # 都市が同じ(u==v)でタイミングが異なる(i!=j)
                Q1[(ui, vj)] += 2
            if u<v and i==j: # 同一タイミング(i==j)で都市が異なる(u<v)
                Q1[(ui, vj)] += 2
        return Q1

    def gen_qubo2(self, cities, cost_matrix):
        qubo_size = cities * cities
        Q2 = np.zeros((qubo_size, qubo_size))
        # u, v は訪れる都市。i, j は巡回の順番
        indices = [(u, v, i, j) for u in range(cities) for v in range(cities) for i
in range(cities) for j in range(cities)]
        for u,v,i,j in indices:
            ui = u * cities + i
            vj = v * cities + j
            k = abs(i - j)
            if ui>vj: # 上三角だけ
                continue
            if (k == 1 or k == (cities - 1)) and u < v: # 隣り合う都市順なら
                for r in range(len(cost_matrix)):
                    if cost_matrix[r][0] == u and cost_matrix[r][1] == v:
                        Q2[ui][vj] += cost_matrix[r][2] # 都市の u と v の間のコスト
        return Q2

    def gen_qubo(self, lagrange1=1.0, lagrange2=1.0):
        Q1 = self.gen_qubo1(self.cities)
        Q2 = self.gen_qubo2(self.cities, self.cost_matrix)
        Q = lagrange1 * Q1 + lagrange2 * Q2
        return Q

    def solv(self, Q, num_reads=1):
        response = self.sampler.sample_qubo(Q, num_reads=num_reads)
        #sample = response.first.sample
        return response

    def result(self, sample_frequency):
        solved = []
        for solution in sample_frequency:

```

```

        if not tsp.check(np.array(solution).reshape(cities, cities)):
            continue
        else:
            solved.append(solution)
ans = []
min_cost = cost_matrix[0][2] * 100
for item in solved:
    jyun = []
    w = np.array(item).reshape(cities, cities)
    for row in range(cities):
        for clmn in range(cities):
            if w[row][clmn] == 1:
                jyun.append(city[clmn])
    cost = u = v = 0
    for i, c in enumerate(jyun):
        u = city.index(jyun[i])
        if i == len(jyun) - 1:
            v = city.index(jyun[0])
        else:
            v = city.index(jyun[i + 1])
    for r in range(len(cost_matrix)):
        if cost_matrix[r][0] == u and cost_matrix[r][1] == v:
            cost += cost_matrix[r][2] # 都市の u と v の間のコスト
            break
    sol = (jyun, cost)
    if cost < min_cost:
        min_cost = cost
    ans.append(sol)
return ans, min_cost

def evaluate(self, sampleset, prn=True):
    # Extract sample solutions, energies, and sort them by frequency
    samples = sampleset.record['sample']
    energies = sampleset.record['energy']
    # Combine solutions and corresponding energies
    sample_data = [(tuple(sample), energy) for sample, energy in zip(samples,
energies)]
    # Sort the results by appearance frequency and then energy
    sample_frequency = Counter(sample for sample, _ in sample_data)
    # Print sorted results by frequency and include energy
    if prn:
        print("\nSorted samples by frequency and energy:")
        for solution, freq in sample_frequency.most_common():
            energy = next(energy for sample, energy in sample_data if sample ==
solution)
            print(f"Sample: {solution}, Frequency: {freq}, Energy: {energy:+.2f
}")
    return sample_data, sample_frequency

def check(self, w):
    for row in w:
        sum=0
        for s in row:
            sum += s

```

```

        if sum!=1:
            return False
    wt = w.T
    for row in wt:
        sum = 0
        for s in row:
            sum += s
        if sum!=1:
            return False
    return True

if __name__ == '__main__':
    city=["A", "B", "C", "D", "E"]
    cities = len(city)
    cost = np.array([
        [0.0, 3.0, 4.0, 2.0, 7.0],
        [3.0, 0.0, 4.0, 6.0, 3.0],
        [4.0, 4.0, 0.0, 5.0, 8.0],
        [2.0, 6.0, 5.0, 0.0, 6.0],
        [7.0, 3.0, 8.0, 6.0, 0.0]])
    cost_matrix = []
    for i, row in enumerate(cost):
        for j, column in enumerate(cost.T):
            if i!=j:
                row1 = [i, j, cost[i][j]]
                cost_matrix.append(row1)
    tsp = TSP(city, cost_matrix)
    lagrange2=1/15
    Q = tsp.gen_qubo(lagrange2=lagrange2)
    sampleset = tsp.solve(Q, num_reads=100)
    sample_data, sample_frequency = tsp.evaluate(sampleset, prn=False)
    answer, min_cost = tsp.result(sample_frequency)
    for ans, cost in answer:
        if cost==min_cost:
            print(ans, cost)

```

プログラム 1.1 巡回セールスマン問題

[実行結果] A,C,B,E,D が循環している（どこからスタートしたかによる）

```

['A', 'C', 'B', 'E', 'D'] 19.0
['E', 'B', 'C', 'A', 'D'] 19.0
['D', 'E', 'B', 'C', 'A'] 19.0
['B', 'E', 'D', 'A', 'C'] 19.0
['D', 'A', 'C', 'B', 'E'] 19.0
['C', 'B', 'E', 'D', 'A'] 19.0
['C', 'A', 'D', 'E', 'B'] 19.0
['A', 'D', 'E', 'B', 'C'] 19.0

```

プロセスは終了コード 0 で終了しました

参考文献

- [1] 西森秀稔、大関真之, 量子アニーリングの基礎, 共立出版, 2.6.1
- [2] https://qiita.com/suzuki_sh/items/32468fbb3f400edce35
- [3] <https://qiita.com/yabish/items/9f42e3752174aef8b79f>
- [4] <https://qiita.com/yufuji25/items/0425567b800443a679f7>
- [5] <https://motojapan.hateblo.jp/entry/2017/11/15/082738>