第 1 章

ナップザック問題を量子アニーリングで解く

参考にしたネット上の記事：
https://qiita.com/farcpan/items/5c2053ce1fd79e644eef
https://zenn.dev/ayu_dev/articles/caca48408b7754

1.1 問題

それぞれに価値 v とコスト（例えば体積）c が設定されている N 個の荷物がある。あるナップザックに、それらから荷物を選んで詰めて運ぼうとしている。運ぶ荷物の価値の総和ができるだけ大きくになる様に荷物を選びたいが、ナップザックには許容される容量に限度があり、その値は C である。この決められたコストを上回らない様に、選んだ荷物の価値合計を最大化する様な荷物の選び方を決める問題である。

N 個の荷物にインデックス $\alpha = 1, 2, \cdots, N$ をつける。ナップザックに入れる荷物に対応する変数には 1 を、入れない荷物には 0 を割り当てる事にすると、この問題は以下の様に書ける。

$$max \quad \sum_{\alpha=1}^{N} v_\alpha q_\alpha \qquad s.t. \quad \sum_{\alpha=1}^{N} c_\alpha q_\alpha \leq C \tag{1.1}$$

1.2 定式化（その 1）

イジングモデルのハミルトニアン H は、選択した荷物の価値合計を最大化する項 H_{value} と、コストを C 以下になる様にする制約事項の項 H_{cost} の 2 つからなっている。

$$H = H_{value} + H_{cost} \tag{1.2}$$

H_{value} は次の通り

$$H_{value} = -\sum_{\alpha}^{N} v_\alpha q_\alpha \ , \qquad q_\alpha \in \{0, 1\} \tag{1.3}$$

価値の合計が最大の時に、H_{value} は最小値をとる。

一方、制約条件の項だが、選んだ荷物のコスト（各体積）の合計 $\sum_\alpha c_\alpha q_\alpha$ が、ナップザックの最大許容体積である C と等しいとする制約条件ならば、次の様になる

$$H_{cost} = \lambda \left(C - \sum_{\alpha=1}^{N} c_\alpha q_\alpha \right)^2 \tag{1.4}$$

ここで、パラメータ λ は正の定数である。

この制約条件を展開する

$$\begin{aligned} H_{cost} = \lambda\Big(C - \sum_\alpha c_\alpha q_\alpha \Big)^2 &= \lambda\Big(C^2 - 2C\sum_\alpha c_\alpha q_\alpha + \Big(\sum_\alpha c_\alpha q_\alpha \Big)^2 \Big) \\ &= \lambda\Big(-2C\sum_{\alpha=1}^{N} c_\alpha q_\alpha + \sum_{\alpha=1}^{N} c_\alpha^2 q_\alpha^2 + 2\sum_{\alpha_1<\alpha_2} c_{\alpha_1} c_{\alpha_2} q_{\alpha_1} q_{\alpha_2} \Big) \\ &= \lambda\Big(\sum_{\alpha=1}^{N} c_\alpha (c_\alpha - 2C) q_\alpha + 2\sum_{\alpha_1<\alpha_2} c_{\alpha_1} c_{\alpha_2} q_{\alpha_1} q_{\alpha_2} \Big) \\ &= \lambda\Big(\sum_{\alpha=1}^{N} c_\alpha (c_\alpha - 2C) q_\alpha + 2\sum_{\alpha=1}^{N-1}\sum_{\beta=\alpha+1}^{N} c_\alpha c_\beta q_\alpha q_\beta \Big) \end{aligned}$$

式の変形にあたっては、以下の性質を利用している.

$$\Big(\sum_i a_i x_i \Big)^2 = \sum_i a_i^2 x_i^2 + 2\sum_{i<j} a_i a_j x_i x_j$$

また, バイナリ変数に対する $s_k^2 = s_k,\ \ s_k \in \{0,1\}$ の性質も使っている。

こうして、この問題のハミルトニアン H は、λ_1 と λ_2 を正の定数として、次の様に書ける。

$$H = H_{cost} + H_{value} = \lambda_1 \Big(\sum_{\alpha=1}^{N} c_\alpha (c_\alpha - 2C) q_\alpha + 2\sum_{\alpha=1}^{N-1}\sum_{\beta=\alpha+1}^{N} c_\alpha c_\beta q_\alpha q_\beta \Big) - \lambda_2 \sum_{\alpha}^{N} v_\alpha q_\alpha$$

制約条件の項にかかるパラメータ λ を決める必要がある.
これを適当に決めた場合には, 得られる最適解は期待したものにはならない.
例えば,$1 \gg \lambda$ となるようなパラメータを選んだ場合には, 価値合計を最大化する項が強くなり, 制約条件の項が相対的に弱くなる. これはすなわち, 最適解として得た QUBO 変数が, 制約条件を満たさない可能性が高いことを意味する. 一方で,$1 \ll \lambda$ を満たすパラメータを選んだ場合, 制約条件の項が強くなり制約条件が優先的に満たされるようになるが, 一方で価値合計が最大とはならない可能性が高くなる.
最適解を与える QUBO 変数列 $\{q_\alpha\}$ を用意し、適当に選んだ QUBO 変数 $q_k \in \{q_\alpha\}$ を 1 つだけ変更することを考える. このとき価値合計は $|\Delta H_{value}| = c_k$ だけ小さくなる. 制約条件の項が全体のハミルトニアンに影響するためには, 以下を満たすようにパラメータを選ぶのがよい.

$$0 \leq max(c_\alpha) \leq \lambda$$

1.2.1　実装

具体的な問題を解いてみる。下表の通り、A, B, C, D, E の $N = 5$ 個の荷物がある。それぞれの荷物の価値と体積は表の通りである。また、ナップザックの最大許容体積 C は 12 である。

荷物	A	B	C	D	E
インデックス (α)	1	2	3	4	5
体積 (c_α)	3	4	6	1	5
価値 (v_α)	6	7	8	1	4

プログラムのコードに落としてみると次の通り

```
from openjij import SASampler, SQASampler
from collections import defaultdict, Counter

class MODEL1:
    def __init__(self):
        self.N = 5
        self.C = 12
        self.c = [3.0, 4.0, 6.0, 1.0, 5.0]
        self.v = [6.0, 7.0, 8.0, 1.0, 4.0]
        self.L1 = 1.0
        self.L2 = self.L1 / 9.0
        self.Samplers = [SASampler, SQASampler]
        self.sampler = self.Samplers[1]()

    def cost(self, Q, L1):
        for a in range(self.N):
            Q[(a, a)] += L1 * self.c[a] * (self.c[a] - 2.0 * self.C)
        for a in range(self.N - 1):
            for b in range(a + 1, self.N):
                Q[(a, b)] += 2.0 * L1 * self.c[a] * self.c[b]

    def value(self, Q, L2):
        for a in range(self.N):
            Q[(a, a)] -= L2 * self.v[a]

    def hamiltonian(self, Q):
        self.cost(Q, self.L1)
        self.value(Q, self.L2)

    def print_QUBO(self, Q):
        # Print QUBO matrix (for debugging)
        for i in range(self.N):
            for j in range(self.N):
                print("{:7.1f}".format(Q[(i, j)]), end=" ")
            print()

    def sample(self, Q, num_reads=100):
        # Perform sampling
        sampleset = self.sampler.sample_qubo(Q, num_reads=num_reads)
        return sampleset
```

```
    def evaluate(self, sampleset):
        # 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
        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}")
        # Evaluate each solution
        print("\nEvaluation of solutions:")
        for nth, (solution, freq) in enumerate(sample_frequency.most_common(), 1):
            value = volume = 0.0
            for i, bit in enumerate(solution[:self.N]):  # Consider only the first
    N bits for w and c
                if bit == 1:
                    volume += self.c[i]
                    value += self.v[i]
            if volume <= self.C:
                energy = next(energy for sample, energy in sample_data if sample ==
     solution)
                print(f"[{nth:2d}] Frequency={freq}, Energy={energy:+.2f},\n
    Solution={solution}", end="\t")
                print(f": value={value:4.1f}, cost={volume:4.1f}")

if __name__ == "__main__":
    model = MODEL1()
    Q = defaultdict(lambda: 0.0)
    model.hamiltonian(Q)
    model.print_QUBO(Q)
    sampleset = model.sample(Q, num_reads=100)
    model.evaluate(sampleset)
```

プログラム 1.1　MODEL1

プログラムの実行

```
-63.7    24.0    36.0     6.0    30.0
  0.0   -80.8    48.0     8.0    40.0
  0.0     0.0  -108.9    12.0    60.0
  0.0     0.0     0.0   -23.1    10.0
  0.0     0.0     0.0     0.0   -95.4

Sorted samples by frequency and energy:
```

```
Sample: (1, 1, 0, 0, 1), Frequency: 37, Energy: -145.89
Sample: (0, 0, 1, 1, 1), Frequency: 30, Energy: -145.44
Sample: (1, 1, 1, 0, 0), Frequency: 30, Energy: -145.33
Sample: (0, 1, 1, 1, 0), Frequency: 2, Energy: -144.78
Sample: (0, 0, 1, 0, 1), Frequency: 1, Energy: -144.33

Evaluation of solutions:
[ 1] Frequency=37, Energy=-145.89,
 Solution=(1, 1, 0, 0, 1) : value=17.0, cost=12.0
[ 2] Frequency=30, Energy=-145.44,
 Solution=(0, 0, 1, 1, 1) : value=13.0, cost=12.0
[ 4] Frequency=2, Energy=-144.78,
 Solution=(0, 1, 1, 1, 0) : value=16.0, cost=11.0
[ 5] Frequency=1, Energy=-144.33,
 Solution=(0, 0, 1, 0, 1) : value=12.0, cost=11.0
```

プロセスは終了コード 0 で終了しました

1.3　定式化（その 2）

しかし今の問題では、総体積は必ずしも C に等しい必要はなく、C より小さい総体積であっても、価値合計が大きいのであれば、そちらの方を最適解だと選ぶ事になる。従ってここでは「コストの合計値が C 以下」になるとする制約条件が必要になる。そこで新たに補助変数（slack 返送鵜と呼ばれる）$s_k \in \{0,1\}, k = 1, 2, \ldots, C$ を用意して、次の通り定式化する。

$$H_{cost} = \lambda \left[\left(1 - \sum_k s_k \right)^2 + \left(\sum_k k s_k - \sum_\alpha c_\alpha q_\alpha \right)^2 \right] = \lambda \left(H_{cost}^{(1)} + H_{cost}^{(2)} \right) \tag{1.5}$$

まずは $H_{cost}^{(1)}$ を展開する.

$$\begin{aligned} H_{cost}^{(1)} &= \left(1 - \sum_k s_k \right)^2 \\ &= 1 - 2\sum_k s_k + \left(\sum_k s_k \right)^2 \\ &= 1 - 2\sum_k s_k + \left(\sum_k s_k^2 + 2\sum_{k<l} s_k s_l \right) \\ &= 1 - \sum_k s_k + 2\sum_{k<l} s_k s_l \end{aligned}$$

次に、$H_{cost}^{(2)}$ を展開する.

$$
\begin{aligned}
H_{cost}^{(2)} &= \left(\sum_k k s_k - \sum_\alpha c_\alpha q_\alpha \right)^2 \\
&= \left(\sum_k k s_k \right)^2 + \left(\sum_\alpha c_\alpha q_\alpha \right)^2 - 2 \sum_k \sum_\alpha k c_\alpha s_k q_\alpha \\
&= \left(\sum_k k^2 s_k + 2 \sum_{k<l} k l s_k s_l \right) + \left(\sum_\alpha c_\alpha^2 q_\alpha + 2 \sum_{\alpha<\beta} c_\alpha c_\beta q_\alpha q_\beta \right) - 2 \sum_k \sum_\alpha k c_\alpha s_k q_\alpha
\end{aligned}
$$

こうして、$H_{cost} = H_{cost}^{(1)} + H_{cost}^{(2)}$ は、次の様になる。

$$
\frac{H_{cost}}{\lambda} = 1 + \sum_k (k^2 - 1) s_k + 2 \sum_{k<l} (kl + 1) s_k s_l + \sum_\alpha c_\alpha^2 q_\alpha + 2 \sum_{\alpha<\beta} c_\alpha c_\beta q_\alpha q_\beta - 2 \sum_k \sum_\alpha k c_\alpha s_k q_\alpha
$$

ここで,s_k も QUBO 変数の一種と見なす. すなわち,

$$
s_k \longrightarrow q_\alpha, \quad \alpha = k + N
$$

以上の様にすれば, ハミルトニアンは以下のようになる.

$$
\begin{aligned}
\frac{H_{cost}}{\lambda} = 1 &+ \sum_{k=1}^{C} (k^2 - 1) q_{k+N} + \sum_{\alpha=1}^{N} c_\alpha^2 q_\alpha \\
&+ 2 \sum_{k=1}^{C-1} \sum_{l=k+1}^{C} (kl + 1) q_{k+N} q_{l+N} + 2 \sum_{\alpha=1}^{N-1} \sum_{\beta=\alpha+1}^{N} c_\alpha c_\beta q_\alpha q_\beta - 2 \sum_{k=1}^{C} \sum_{\alpha=1}^{N} k c_\alpha q_{k+N} q_\alpha
\end{aligned}
$$

1.3.1　実装

C 個の slack 変数 $s_1, \cdots, s_{C-1}, s_C$ を導入して、その中の 1 つだけが 1 になることを許す（複数の 1 は禁止）を課している。

ナップザックの体積（c）	1	2	3	...	10	11	12
総体積 (c) になるかどうか（s_n）	0	0	0	...	0	0	1
$n \times s_n$	0 1×0	0 2×0	0 3×0	...	0 10×0	0 11×0	12 12×1

一方、ナップサックの荷物の体積の総和 (荷物の総体積) は以下で求められる。

$$
c = \sum_{\alpha=1}^{N} c_\alpha q_\alpha = c_1 q_1 + c_2 q_2 + c_3 q_3 + \cdots + c_N q_N
$$

q_n は $0, 1$ を取る決定変数で, n 番目の荷物をナップサックに詰めるかどうかを決める変数なので, 例えば下表のように荷物 C, D, E を選択すれば, 荷物の総体積（$3 \times 0 + 4 \times 0 + 6 \times 1 + 1 \times 1 + 5 \times 1 = 12$）を 12 にすることができる.

```
from knapsack import MODEL1
from collections import defaultdict, Counter

class MODEL2(MODEL1):
    def __init__(self):
        super().__init__()
        self.L1 = 1.0
        self.L2 = self.L1 / 9.0
        self.L3 = self.L1 / 0.2

    def slack_unique(self, Q, L1):
        for k in range(self.C):
            Q[(k+self.N, k+self.N)] -= L1
        for k in range(self.C - 1):
            for l in range(k + 1, self.C):
                Q[(k+self.N, l+self.N)] += 2.0 * L1

    def slack_sum(self, Q, L1):
        for k in range(self.C):
            Q[(k+self.N, k+self.N)] += L1 * k**2
        for k in range(self.C - 1):
            for l in range(k + 1, self.C):
                Q[(k+self.N, l+self.N)] += 2.0 * L1 * k * l
        for a in range(self.N):
            Q[(a, a)] += L1 * self.c[a] * self.c[a]
        for a in range(self.N - 1):
            for b in range(a + 1, self.N):
                Q[(a, b)] += 2.0 * L1 * self.c[a] * self.c[b]
        for a in range(self.N - 1):
            for b in range(a + 1, self.N):
                Q[(a, b)] += 2.0 * L1 * self.c[a] * self.c[b]
        for k in range(self.C):
            for a in range(self.N):
                Q[(k+self.N, a)] -= 2.0 * L1 * k * self.c[a]

    def cost2(self, Q, L1):
        self.slack_unique(Q, L1)
        self.slack_sum(Q, L1)

    def hamiltonian(self, Q):
        self.cost(Q, self.L3)
        self.cost2(Q, self.L1)
        self.value(Q, self.L2)

if __name__ == "__main__":
    model = MODEL2()
    Q = defaultdict(lambda: 0.0)
    model.hamiltonian(Q)
    model.print_QUBO(Q)
    sampleset = model.sample(Q, num_reads=10)
    model.evaluate(sampleset)
```

プログラム 1.2　MODEL2

ここで作成した cost2() 関数は、結局のところ「12 以下」のいずれの場合も許容されることになり、そのいずれの場合でも等しく低いエネルギーレヴェルを実現できることになる。そこで、MODEL1 クラスで作成した cos() 関数を追加してみた。ただし、この cost() 関数の寄与を大きくすると結局 MODEL1 と同じことをしている事になるので要注意。

以下の結果の出力では、選ばれた荷物の容積の総和が 12 を超えるものは表示していない

```
-306.7   168.0   252.0    42.0   210.0
0.0  -384.8   336.0    56.0   280.0
0.0     0.0  -504.9    84.0   420.0
0.0     0.0     0.0  -114.1    70.0
0.0     0.0     0.0     0.0  -450.4

Sorted samples by frequency and energy:
Sample: (0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0), Frequency: 2, Energy: -653.67
Sample: (0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0), Frequency: 1, Energy: -643.33
Sample: (0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0), Frequency: 1, Energy: -656.33
Sample: (0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0), Frequency: 1, Energy: -655.33
Sample: (0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0), Frequency: 1, Energy: -635.44
Sample: (0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0), Frequency: 1, Energy: -525.11
Sample: (0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0), Frequency: 1, Energy: -652.67
Sample: (0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0), Frequency: 1, Energy: -656.33
Sample: (0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0), Frequency: 1, Energy: -642.33

Evaluation of solutions:
[ 1] Frequency=2, Energy=-653.67,
Solution=(0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0) : value=15.0, cost=10.0
[ 2] Frequency=1, Energy=-643.33,
Solution=(0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0) : value=12.0, cost=10.0
[ 3] Frequency=1, Energy=-656.33,
Solution=(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0) : value=12.0, cost=11.0
[ 4] Frequency=1, Energy=-655.33,
Solution=(0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0) : value=12.0, cost=11.0
[ 5] Frequency=1, Energy=-635.44,
Solution=(0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0) : value=13.0, cost=12.0
[ 7] Frequency=1, Energy=-652.67,
Solution=(0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0) : value=15.0, cost=10.0
[ 8] Frequency=1, Energy=-656.33,
Solution=(0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0) : value=12.0, cost=11.0
[ 9] Frequency=1, Energy=-642.33,
Solution=(0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0) : value=12.0, cost=10.0
```

プロセスは終了コード 0 で終了しました

1.4　定式化（その 3）

slack 変数は、決定変数 q_α の後ろに繋げて解く事になるので、不等式の制約条件によっては、slack 変数の数が多くなるという不都合がある。そこで、slack 変数のバイナリエンコーディングを行うことがある。今のナップザックぬ問題の具体例では、ナップザックの最大容量 C が 12 だとしているので、前の例の様な slack 変数の使い方だと、$s_1, s_2, \cdots, s_{C-1}, s_C$ と、全部で $C = 12$ 個の slack 変数を追加していることになる。一方バイナリーエンコーディングでは、例えば 12 なら、次の表の様な 2 進数の 4 桁で表現できるので、4 個の slack 変数を用意すれば良い事になる。

$$12 = (2^0 \times 0) + (2^1 \times 0) + (2^2 \times 1) + (2^3 \times 1) \rightarrow \sum_{n=0}^{3} (2^n \times s_n)$$

s3	s2	s1	s0	値
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

前の例で実施した、バイナリーエンコーディングなしの、通常の slack 変数の場合のハミルトニアンは、

$$H \ = \ H_A \ + \ H_B = \ A_1 \left(1 - \sum_{n=1}^{12} s_n \right)^2 + A_2 \left(\sum_{n=1}^{12} n s_n - \sum_{\alpha=1}^{5} c_\alpha q_\alpha \right)^2 - B \sum_{\alpha=1}^{5} v_\alpha q_\alpha$$

この様にしていたが、今回はまず A_1 のかかる項はなくなる。（slack 変数の選択は 1 つだけではなくなるから）そして、A_2 のかかる項では、2 進符号化 $(2^0 s_0 + 2^1 s_1 + 2^2 s_2 + 2^3 s_n = \sum_{n=0}^{3} 2^n s_n)$ をしなければいけないから、

$$H \ = \ H_A \ + \ H_B = A_2 \left(\sum_{n=0}^{3} 2^n s_n - \sum_{\alpha=1}^{5} c_\alpha q_\alpha \right)^2 - B \sum_{\alpha=1}^{5} v_\alpha q_\alpha$$

4 ビットで 2 進符号化している（0、1、2、3=M）ことから $M = 3$ とし, 荷物の数を $N = 5$ として式を変形していく。

$$
\begin{aligned}
H &= H_A + H_B \\
&= A_2\Big(\sum_{n=0}^{3} 2^n s_n - \sum_{\alpha=1}^{5} c_\alpha q_\alpha\Big)^2 - B\sum_{\alpha=1}^{5} v_\alpha q_\alpha \\
&= A_2\Big(\sum_{n=0}^{M} 2^n s_n - \sum_{\alpha=1}^{N} c_\alpha q_\alpha\Big)^2 - B\sum_{\alpha=1}^{N} v_\alpha q_\alpha \\
&= A_2\Big(\sum_{n=0}^{M} 2^n s_n\Big)^2 + A_2\Big(\sum_{\alpha=1}^{N} c_\alpha q_\alpha\Big)^2 - 2A_2\Big(\sum_{n=0}^{M}\sum_{\alpha=1}^{N} 2^n s_n c_\alpha q_\alpha\Big) - B\sum_{\alpha=1}^{N} v_\alpha q_\alpha \\
&= A_2\Big(\sum_{n=0}^{M} 2^{2n} s_n^2 + 2\sum_{n_1<n_2} 2^{n_1}2^{n_2} s_{n_1} s_{n_2}\Big) + A_2\Big(\sum_{\alpha=1}^{N} c_\alpha^2 q_\alpha^2 + 2\sum_{\alpha_1<\alpha_2} c_{\alpha_1} c_{\alpha_2} q_{\alpha_1} q_{\alpha_2}\Big) \\
&\quad - 2A_2\Big(\sum_{n=0}^{M}\sum_{\alpha=1}^{N} 2^n s_n c_\alpha q_\alpha\Big) - B\sum_{\alpha=1}^{N} v_\alpha q_\alpha \\
&= 2A_2\Big(\sum_{n_1<n_2} 2^{n_1}2^{n_2} s_{n_1} s_{n_2} + \sum_{\alpha_1<\alpha_2} c_{\alpha_1} c_{\alpha_2} q_{\alpha_1} q_{\alpha_2} - \sum_{n=0}^{M}\sum_{\alpha=1}^{N} 2^n s_n c_\alpha q_\alpha\Big) \\
&\quad + A_2\Big(\sum_{\alpha=1}^{N} c_\alpha^2 q_\alpha + \sum_{n=0}^{M} 2^{2n} s_n\Big) - B\sum_{\alpha=1}^{N} v_\alpha q_\alpha \\
&= 2A_2\Big(\sum_{n_1=0}^{M-1}\sum_{n_2=n_1+1}^{M} 2^{n_1}2^{n_2} s_{n_1+N} s_{n_2+N} + \sum_{\alpha=1}^{N-1}\sum_{\beta=\alpha+1}^{N} c_\alpha c_\beta q_\alpha q_\beta - \sum_{n=0}^{M}\sum_{\alpha=1}^{N} 2^n s_{n+N} c_\alpha q_\alpha\Big) \\
&\quad + A_2\Big(\sum_{\alpha=1}^{N} c_\alpha^2 q_\alpha + \sum_{n=0}^{M} 2^{2n} s_{n+1+N}\Big) - B\sum_{\alpha=1}^{N} v_\alpha q_\alpha \\
&= 2A\Big(\sum_{n_1=0}^{M-1}\sum_{n_2=n_1+1}^{M} 2^{n_1}2^{n_2} q_{n_1+N} q_{n_2+N} + \sum_{\alpha=1}^{N-1}\sum_{\beta=\alpha+1}^{N} c_\alpha c_\beta q_\alpha q_\beta - \sum_{n=0}^{M}\sum_{\alpha=1}^{N} 2^n q_{n+N} c_\alpha q_\alpha\Big) \\
&\quad + A\Big(\sum_{\alpha=1}^{N} c_\alpha^2 q_\alpha + \sum_{n=0}^{M} 2^{2n} q_{n+N}\Big) - B\sum_{\alpha=1}^{N} v_\alpha q_\alpha
\end{aligned}
$$

終わりの式の変形では、slack 変数 s_0, s_1, s_2, s_3 のそれぞれが、N 個の決定変数 q の並びの後ろに、$q_{0+N}, q_{1+N}, q_{2+N}, q_{3+N}$ の 4 個が繋がって来ることを意識して変形させているので要注意！（A_2 は A とした）

$$
\begin{aligned}
&s_0 \to s_{0+N} \to q_{0+N}, \\
&s_1 \to s_{1+N} \to q_{1+N}, \\
&s_2 \to s_{2+N} \to q_{2+N}, \\
&s_3 \to s_{3+N} \to q_{3+N}
\end{aligned}
$$

また, この並びによって, LSB の S_0 は左側に,MSB が右に並ぶことも確認しておく必要がある.

1.4.1　実装

```
from knapsack import MODEL1
from collections import defaultdict, Counter

class MODEL3(MODEL1):
    def __init__(self):
        super().__init__()
        self.M = 3
        self.L1 = 1.0
        self.L2 = self.L1 / 9.0
        self.L3 = self.L1 / 100.0

    def cost3(self, Q, L1):
        for n1 in range(self.M):
            for n2 in range(n1+1, self.M+1):
                Q[(n1+self.N, n2+self.N)] += 2.0 * L1 * 2**n1 * 2**n2
        for a in range(self.N - 1):
            for b in range(a + 1, self.N):
                Q[(a, b)] += 2.0 * L1 * self.c[a] * self.c[b]
        for n in range(self.M+1):
            for a in range(self.N):
                Q[(n + self.N, a)] -= 2.0 * L1 * 2**n * self.c[a]
        for a in range(self.N):
            Q[(a, a)] += L1 * self.c[a] * self.c[a]
        for n in range(self.M + 1):
            Q[(n + self.N, n + self.N)] += L1 * 2**(2*n)

    def hamiltonian(self, Q):
        self.cost(Q, self.L3)
        self.cost3(Q, self.L1)
        self.value(Q, self.L2)

if __name__ == '__main__':
    model = MODEL3()
    Q = defaultdict(lambda: 0.0)
    model.hamiltonian(Q)
    model.print_QUBO(Q)
    sampleset = model.sample(Q, num_reads=10)
    model.evaluate(sampleset)
```

プログラム 1.3　MODEL3

この 4 ビットの 2 進符号化による cost3() 関数は、結局のところ「15 以下」のいずれの場合も許容されることになり、「15 以下」のどの値でも等しくエネルギーレヴェルの低い状態に落ち着けるだろう。つまり「12 に近い方がよい」という制約が必要になる。そこで、MODEL1 クラスで作成した cos() 関数を追加した。ただし、この cost() 関数の寄与を大きくすると結局 MODEL1 と同じことをしている事になるので要注意。

以下の結果の出力では、選ばれた荷物の容積の総和が 12 を超えるものは表示していない

```
7.7    24.2    36.4     6.1    30.3
0.0    14.4    48.5     8.1    40.4
0.0     0.0    34.0    12.1    60.6
0.0     0.0     0.0     0.7    10.1
```

```
  0.0     0.0     0.0     0.0    23.6

Sorted samples by frequency and energy:
Sample: (1, 1, 1, 1, 0, 1, 1, 1, 1), Frequency: 2, Energy: -2.84
Sample: (1, 1, 0, 1, 0, 1, 1, 1, 0), Frequency: 2, Energy: -1.84
Sample: (1, 1, 0, 0, 1, 0, 0, 1, 1), Frequency: 2, Energy: -3.33
Sample: (1, 1, 1, 0, 0, 1, 0, 1, 1), Frequency: 1, Energy: -3.76
Sample: (1, 0, 1, 1, 1, 1, 1, 1, 1), Frequency: 1, Energy: -3.46
Sample: (1, 0, 1, 1, 0, 1, 1, 0, 1), Frequency: 1, Energy: -2.07
Sample: (1, 1, 0, 1, 1, 1, 0, 1, 1), Frequency: 1, Energy: -3.43

Evaluation of solutions:
[ 2] Frequency=2, Energy=-1.84,
Solution=(1, 1, 0, 1, 0, 1, 1, 1, 0) : value=14.0, cost= 8.0
[ 3] Frequency=2, Energy=-3.33,
Solution=(1, 1, 0, 0, 1, 0, 0, 1, 1) : value=17.0, cost=12.0
[ 6] Frequency=1, Energy=-2.07,
Solution=(1, 0, 1, 1, 0, 1, 1, 0, 1) : value=15.0, cost=10.0
```

プロセスは終了コード 0 で終了しました