iTHEMS-RCNP Workshop
Third Workshop on Density Functional Theory: Fundamentals, Developments, and Applications (DFT2025)

25–27 March 2025
Auditorium, Integrated Innovation Building (IIB), RIKEN Kobe Campus

Quantum Annealing for Optimizing Isotopic Substitutions in Fullerene: A DFT-Assisted Spectral Analysis

Jubin Park

Soongsil University and OMEG institute

In collaboration with
Mr. Minkyu Lee, Prof. Myung-Ki Cheoun (Soongsil University & OMEG institute),
Dr. NAITO Tomoya (RIKEN)

Contents

1. Introduction to optimization using Quantum Annealer (QA)

2. Example I: Graphene with vacancies

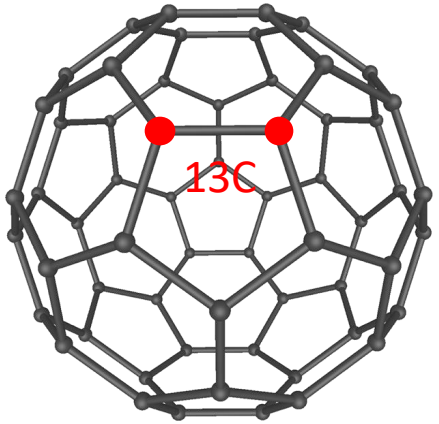
- Optimization of Graphene with vacancies
  Using QA

3. Example II: Fullerene C60 Isotopologues

- Optimization of 13C-Substituted Fullerene Configurations
  Using QA

4. Conclusion

- Summary and future outlook.



1. Introduction to optimization using Quantum Annealer(QA)

- Introduction to Quantum Computer

D-wave Quantum Annealer

→ D-wave Advantage 2 (2023)

→ 7000++ qubit

→ High applicability and accuracy of quantum measurement

→ **Programming difficulty**

→ Require more qubits

IBM-Q, ION-Q

→ IBM-Q, Ion-Q

→ 400++ qubit

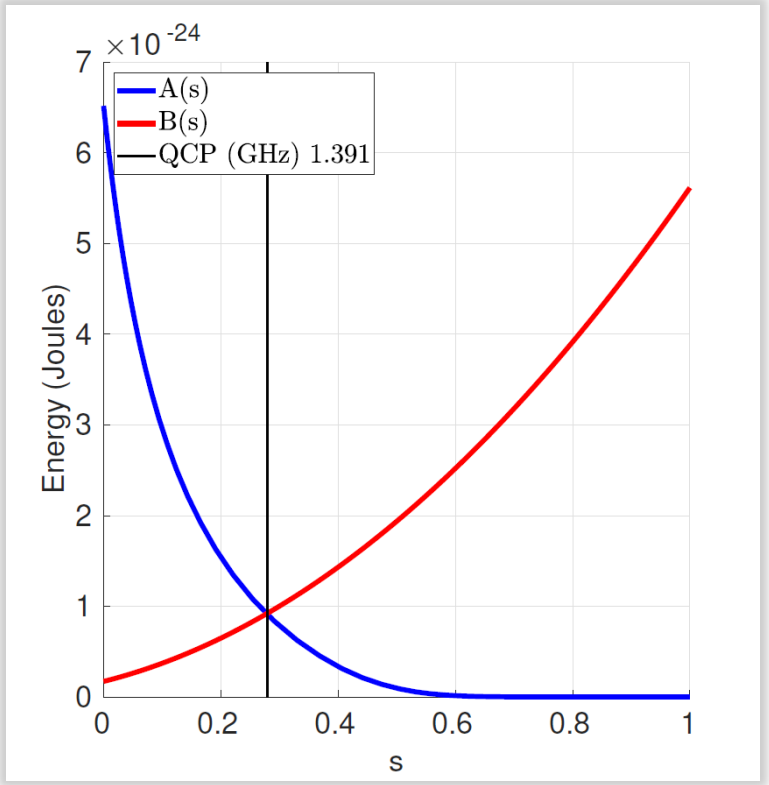
→ Large quantum error

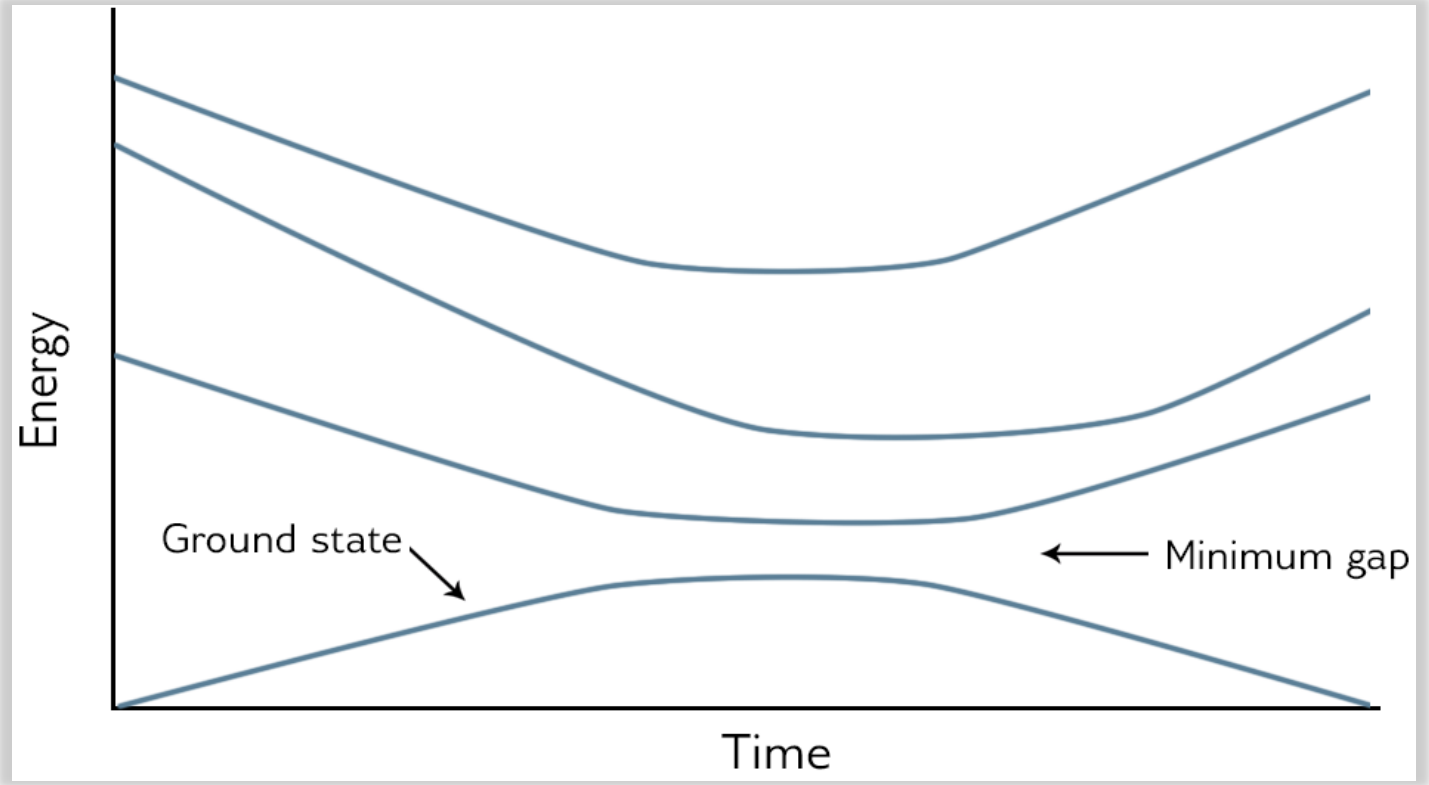
→ **Programmable**

→ Require more qubits

- Annealing

$$H_{Ising} = \underbrace{-\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x^{(i)}\right)}_{\text{Initial state}} + \underbrace{\frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i,j} J_{i,j} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)}\right)}_{\text{Final state}}$$





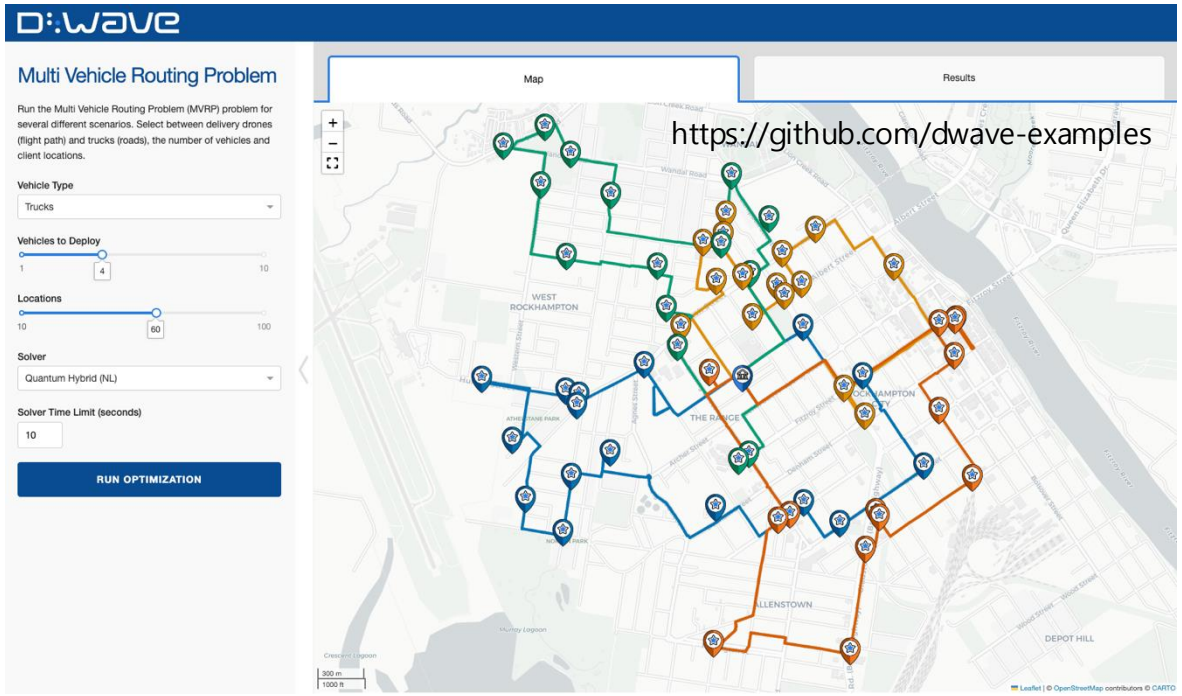
What is Quantum Annealing? — D-Wave System Documentation documentation (dwavesys.com)

<Traveling Salesperson>

The goal of renowned [traveling salesperson](#) optimization problem is, for a given a list of cities and distances between each pair of cities, to find the shortest possible route that visits each city exactly once and returns to the city of origin.

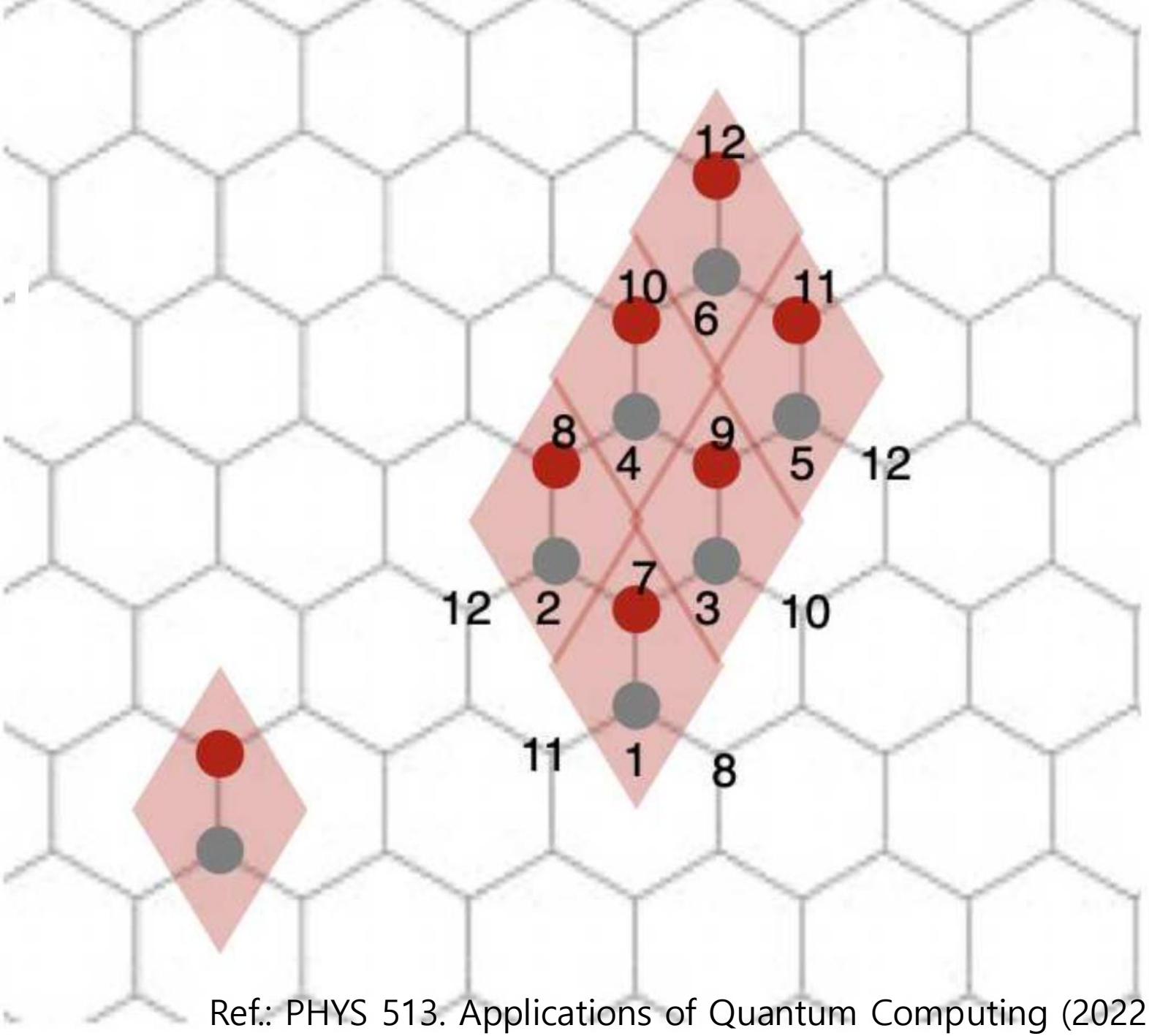
https://docs.ocean.dwavesys.com/en/stable/examples/nl_tsp.html

<Multi Vehicle Routing Problem>



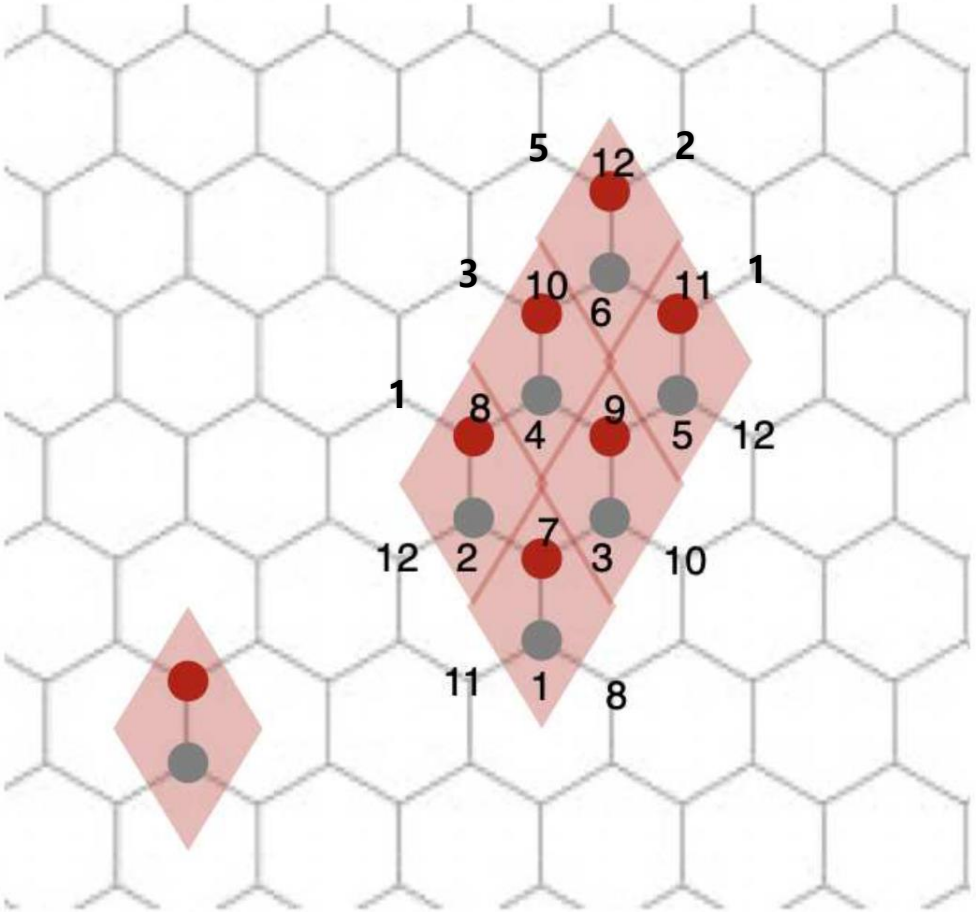
The multi-vehicle routing problem is to deliver a set of resources to a set of predetermined locations using a limited number of vehicles, all of which start and finish at a single depot location.

2. Example I: Graphene with vacancies

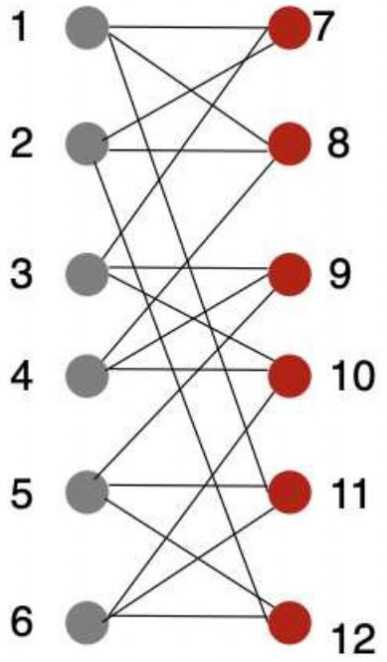


Ref.: PHYS 513. Applications of Quantum Computing (2022), , 1–6

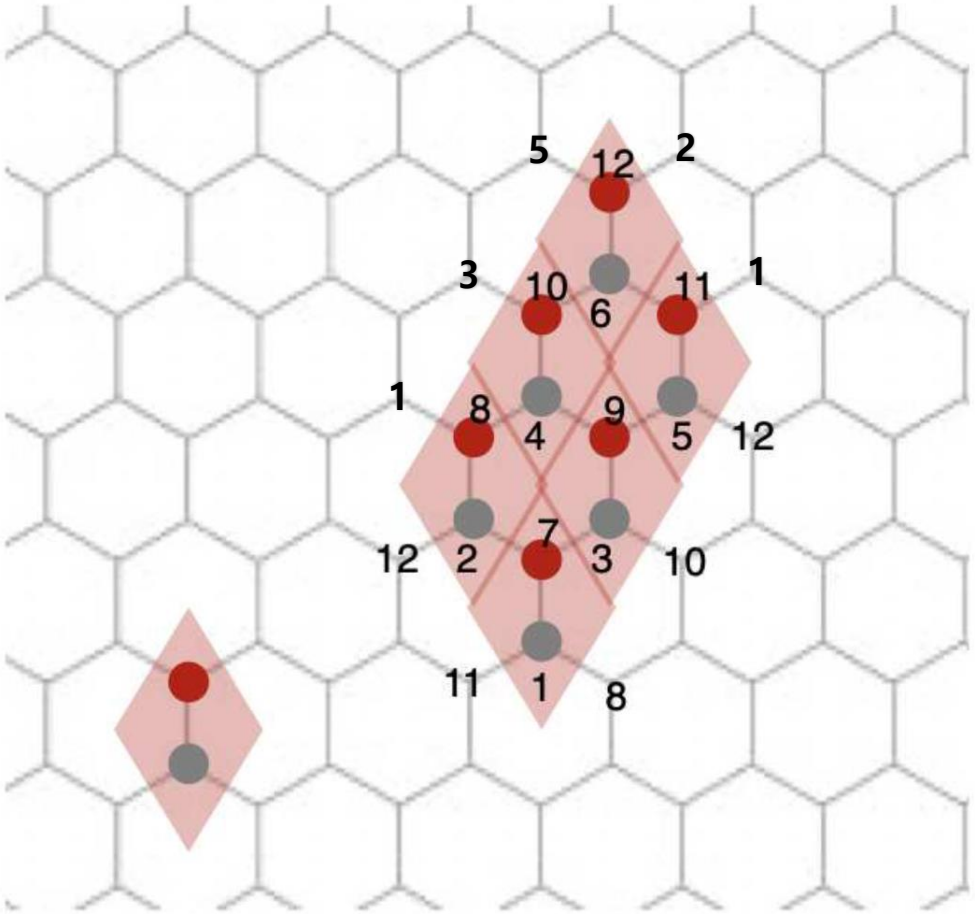
Model using QUBO (**Q**uadratic **U**nconstrained **B**inary **O**ptimization)



Order-3 bipartite graph $G(12, 18)$



- 12 sites → 12 qubits
- 18 edges
→ connectivity btw. qubits



- For carbon atoms on sites → 12 qubits: $x_{i\alpha_c}$
- For vacancies → additional 12 qubits: $x_{i\alpha_v}$

A node(site) $i \in E$ can be occupied by either a carbon α_c or a vacancy α_v. Each node(site) i is associated with a couple of binary variables $\{x_{i\alpha_c}, x_{i\alpha_v}\}$, such that $x_{i\alpha_{c(v)}}=1$ when node(site) i is occupied by a carbon (vacancy) atom, $x_{i\alpha_{c(v)}}=0$ otherwise.

A constraint and the occupation numbers N_{α_c} and N_{α_v} give us the following conditions

- $x_{i\alpha_c} + x_{i\alpha_v} = 1$ for all $i \in E$

→ Each site i must be filled with a carbon or a vacancy.

- $N_{\alpha_c} = \sum_{i\in E} x_{i\alpha_c}$ $\quad N_{\alpha_v} = \sum_{i\in E} x_{i\alpha_v}$

→ The filled carbon atoms and cavities must have a fixed occupancy number.

Therefore, the Qubo function is

$$Q(\mathbf{x}) = p\sum_{i=0}^{d-1}(1 - x_{i\alpha_c} - x_{i\alpha_v})^2 + (N_{\alpha_c} - \sum_{i=0}^{d-1} x_{i\alpha_c})^2 + (N_{\alpha_v} - \sum_{i=0}^{d-1} x_{i\alpha_v})^2 + \sum_{i,j=0}^{d-1} A_{ij}x_{i\alpha_c}x_{j\alpha_v}$$

※ The value of QUBO can be reduced when all three constraints are fully satisfied (note that the constraints are in the form of square equations) !!!

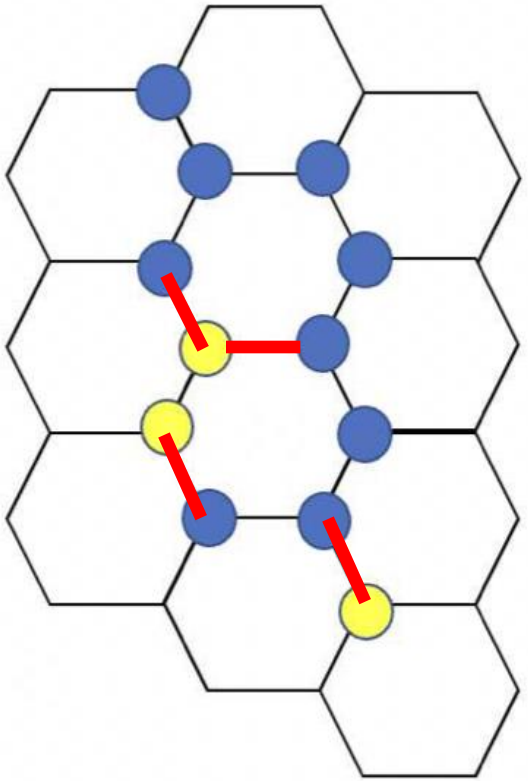
→ Remember that after the annealing process, we can find the state with the lowest energy (i.e. satisfying the constraints) of the system we am considering.

※ However, the above constraints alone do not provide an energy difference between the absence and presence of vacancies. Therefore, in the case where vacancies exist, we must provide an energy difference according to the (physically based) location of the vacancies.

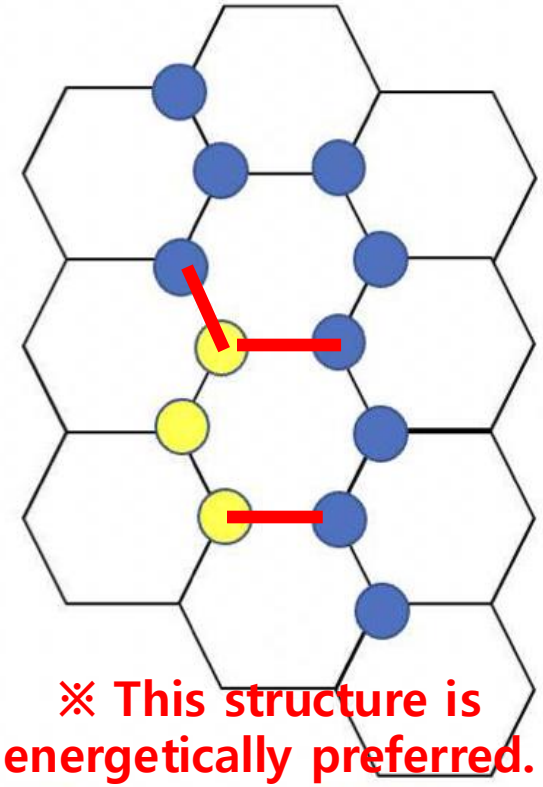
$$Q(\mathbf{x}) = p\sum_{i=0}^{d-1}(1 - x_{i\alpha_c} - x_{i\alpha_v})^2 + (N_{\alpha_c} - \sum_{i=0}^{d-1} x_{i\alpha_c})^2 + (N_{\alpha_v} - \sum_{i=0}^{d-1} x_{i\alpha_v})^2 + \sum_{i,j=0}^{d-1} A_{ij}x_{i\alpha_c}x_{j\alpha_v}$$

→ Increases energy depending on **the number of dangling bonds** in the connection.

# of dangling bonds = 4

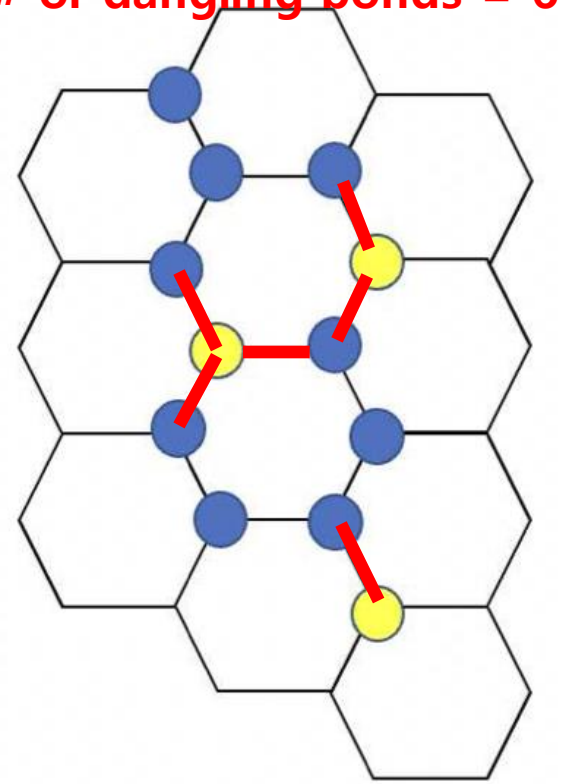


# of dangling bonds = 3

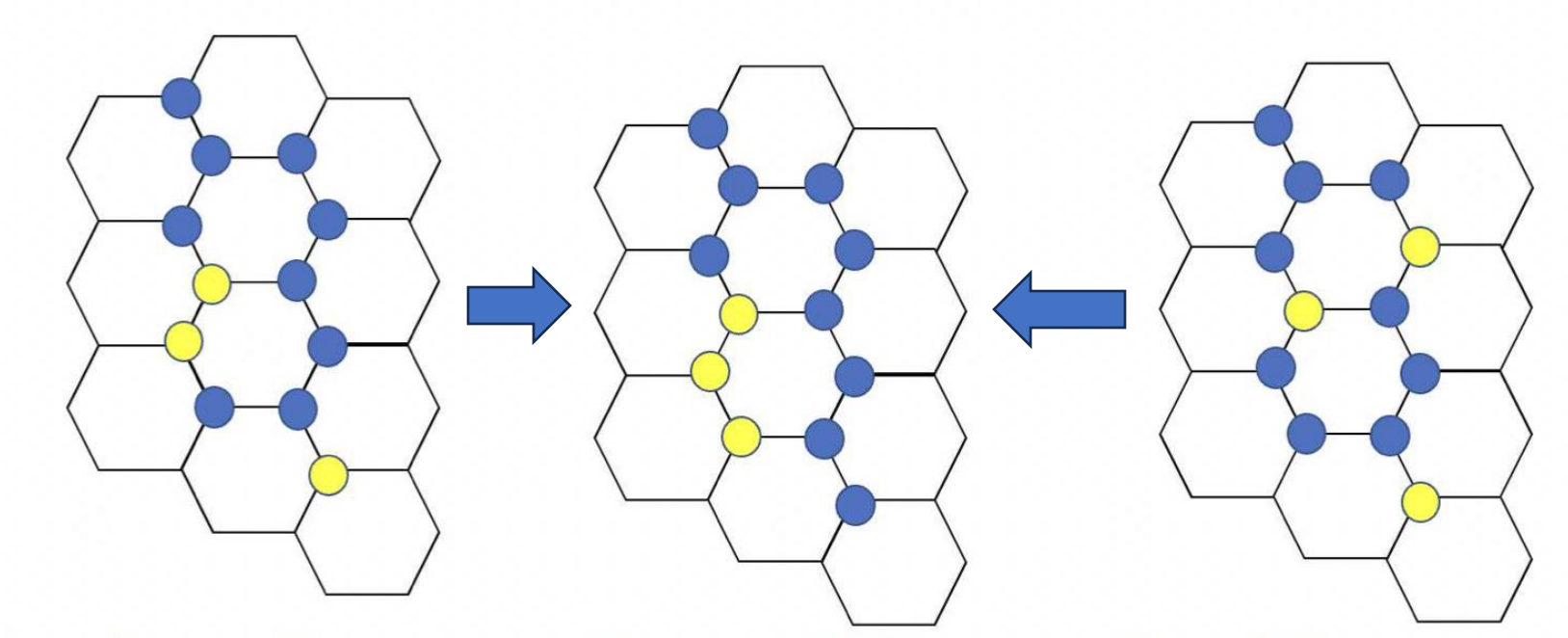


※ This structure is energetically preferred.

# of dangling bonds = 6



※ The dangling bonds form instabilities in the graphene layer, which will cause a reorganization on its structure like vacancy clustering.

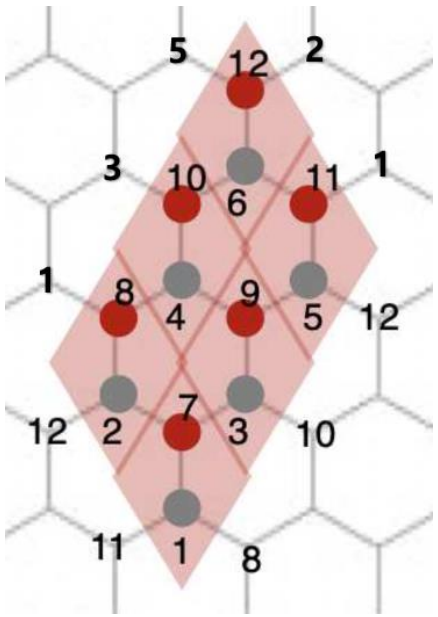


※ This structure is energetically preferred.

$$Q(\mathbf{x}) = p\sum_{i=0}^{d-1}(1 - x_{i\alpha_c} - x_{i\alpha_v})^2 + (N_{\alpha_c} - \sum_{i=0}^{d-1} x_{i\alpha_c})^2 + (N_{\alpha_v} - \sum_{i=0}^{d-1} x_{i\alpha_v})^2 + \sum_{i,j=0}^{d-1} A_{ij} x_{i\alpha_c} x_{j\alpha_v}$$

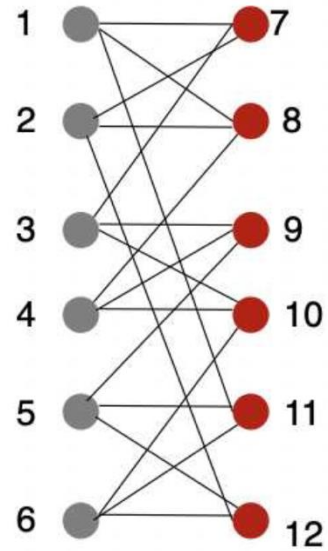
- For carbon atoms on sites → 12 qubits: $x_{i\alpha_c}$
- For vacancies → additional 12 qubits: $x_{i\alpha_v}$

```
oc = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
ov = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```



atrixForm=

0	0	0	0	0	0	1	1	0	0	1	0
0	0	0	0	0	0	1	1	0	0	0	1
0	0	0	0	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0



All possible cases (or all possible states) for each site ($2^{12} = 4096$)

```
ConfigurationTuples = Tuples[{0, 1}, 12]
% // Length
```

```
{{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
 {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1}, ··· 4077 ···, {1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1},
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1},
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}
```

Full expression not available (original memory size: 393.4 kB)

```
4096
```

```
4096 * 4096
```
(For both states of carbon atoms and vacancies)

```
16 777 216
```

※ Even in this simple case, a whopping 16777216 loops are required!!

$$(16777216 \approx 1.6 \times 10^7)$$

Ex. Case with four vacancies (Nv=4)

※ We can see that there are 3 exact solutions!!!

```
AbsoluteTiming[
 Qvalue = 100 000;
 LowestStateOc = {};
 LowestStateOv = {};
 Nc = 8;
 Nv = 4;
 For[i = 1, i ≤ Length[ConfigurationTuples], i++,
  oc = ConfigurationTuples[[i]];
  For[j = 1, j ≤ Length[ConfigurationTuples], j++,
   ov = ConfigurationTuples[[j]];
   If[Qvalue ≥ ObjQ[Nc, Nv],
    Qvalue = ObjQ[Nc, Nv];
    LowestStateOc = oc;
    LowestStateOv = ov;
    If[Qvalue < 10,
     Print["LowestStateOc=", LowestStateOc]; Print["LowestStateOv=", Lowest
     Print["Qvalue=", Qvalue];
    ];
   ];
  ];
 ];
       ObjQ[Nc_, Nv_] := 10 * Sum[(1 - oc[[i]] - ov[[i]]) ^2, {i, 1, 12}] + (Nc - Sum[oc[[i]], {i, 1, 12}]) ^2 +
         100 * (Nv - Sum[ov[[i]], {i, 1, 12}]) ^2 + oc.SMatA.ov
]
```

```
LowestStateOc={0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1}
LowestStateOv={1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0}
Qvalue=6
LowestStateOc={0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1}
LowestStateOv={1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0}
Qvalue=6
LowestStateOc={0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1}
LowestStateOv={1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0}
Qvalue=4
LowestStateOc={1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1}
LowestStateOv={0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0}
Qvalue=4
LowestStateOc={1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0}
LowestStateOv={0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1}
Qvalue=4
{431.75, Null}
```

Interestingly,
it took a total of 431 seconds.

Using QA

```python
# Define the parameters for your QUBO
p0 = 10 # Penalty parameter (adjust based on your requirements)
p1 = 1 # Penalty parameter (adjust based on your requirements)
d = 12  # Adjust 'd' based on the problem size

# Placeholder for constants like N_alpha and adjacency matrix A
N_c = 8  # Example value for the number of the carbon atoms
N_v = 4  # Example value for the number of the vacancies

# The adjacency matrix A as provided
A = [
    [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
    [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
]
```

```python
# QUBO 딕셔너리 초기화
Q = {}

# 첫 번째 항: p * sum((1 - x_{iαc} - x_{iαv}) ** 2)
for i in range(d):
    Q[(f'xc_{i}', f'xc_{i}')] = -p0
    Q[(f'xv_{i}', f'xv_{i}')] = -p0
    Q[(f'xc_{i}', f'xv_{i}')] = +2*p0

# 두 번째, 세 번째 항: (N_alpha - sum(x_{iαc}))**2 와 (N_alpha - sum(x_{iαv}))**2
for i in range(d):
    for j in range(i, d):
        if i == j:
            Q[(f'xc_{i}', f'xc_{i}')] += 1 -2 * N_c
            Q[(f'xv_{i}', f'xv_{i}')] += 1 -2 * p1 * N_v
        else:
            Q[(f'xc_{i}', f'xc_{j}')] = +2
            Q[(f'xv_{i}', f'xv_{j}')] = +2

# 네 번째 항: 인접 행렬에 따른 x_{iαc}와 x_{jαv}의 상호작용 항 (Aij * x_{iαc} * x_{jαv})
for i in range(d):
    for j in range(i+1, d):
            #print(i, j)
            Q[(f'xc_{i}', f'xv_{j}')] = A[i][j]
            Q[(f'xc_{j}', f'xv_{i}')] = A[i][j]
```

```python
# Initialize the sampler and submit the QUBO problem to D-Wave
sampler = EmbeddingComposite(DWaveSampler())
sampleset = sampler.sample_qubo(Q, num_reads=1000)

# Print the results
print(sampleset)
```

A total of three lowest energy states have been discovered!

	xc_0	xc_1	xc_10	xc_11	xc_2	xc_3	xc_4	xc_5	xc_6	...	xv_9	energy	num_oc.	...
0	1	1	1	1	0	0	1	1	1	...	1	-196.0	3	...
1	0	0	1	1	1	1	1	1	0	...	0	-196.0	4	...
2	1	1	0	0	1	1	0	0	1	...	0	-196.0	6	...
3	0	1	0	1	1	1	1	0	1	...	1	-194.0	1	...
4	1	1	1	1	0	0	1	1	1	...	0	-194.0	3	...
5	0	1	0	0	1	1	1	0	1	...	0	-194.0	2	...
6	1	1	1	1	0	0	1	0	1	...	1	-194.0	4	...
7	1	1	0	1	0	1	1	0	1	...	1	-194.0	4	...
8	1	1	1	1	0	1	0	1	0	...	0	-194.0	1	...
9	0	1	0	1	1	1	1	0	1	...	0	-194.0	1	...
10	0	1	0	1	1	0	1	1	1	...	0	-194.0	3	...
11	1	1	1	1	0	1	1	0	1	...	1	-194.0	6	...
12	1	1	1	0	0	1	1	0	1	...	1	-194.0	5	...
13	1	1	1	1	0	1	1	0	0	...	1	-194.0	1	...
14	1	1	1	1	0	0	1	1	0	...	1	-194.0	1	...
15	0	0	1	0	1	1	1	1	1	...	0	-194.0	3	...
16	1	1	1	1	0	0	1	1	1	...	1	-194.0	3	...

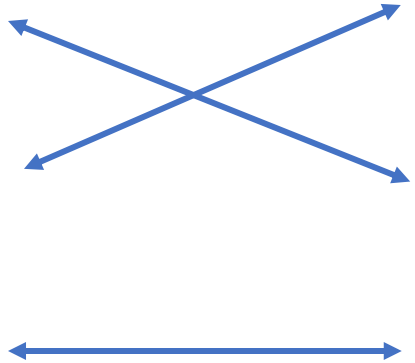
```python
# 전체 결과를 리스트로 변환 후 상위 5개의 결과 출력
samples_list = list(sampleset)
for i, sample in enumerate(samples_list[:5]):
    print(f"Sample {i}: {sample}")
```

```
Sample 0: {'xc_0': 1, 'xc_1': 1, 'xc_10': 1, 'xc_11': 1, 'xc_2': 0, 'xc_3': 0, 'xc_4': 1, 'xc_5': 1, 'xc_6': 1, 'xc_7': 1, 'xc_8': 0, 'xc_9':
0, 'xv_0': 0, 'xv_1': 0, 'xv_10': 0, 'xv_11': 0, 'xv_2': 1, 'xv_3': 1, 'xv_4': 0, 'xv_5': 0, 'xv_6': 0, 'xv_7': 0, 'xv_8': 1, 'xv_9': 1}
Sample 1: {'xc_0': 0, 'xc_1': 0, 'xc_10': 1, 'xc_11': 1, 'xc_2': 1, 'xc_3': 1, 'xc_4': 1, 'xc_5': 1, 'xc_6': 0, 'xc_7': 0, 'xc_8': 1, 'xc_9':
1, 'xv_0': 1, 'xv_1': 1, 'xv_10': 0, 'xv_11': 0, 'xv_2': 0, 'xv_3': 0, 'xv_4': 0, 'xv_5': 0, 'xv_6': 1, 'xv_7': 1, 'xv_8': 0, 'xv_9': 0}
Sample 2: {'xc_0': 1, 'xc_1': 1, 'xc_10': 0, 'xc_11': 0, 'xc_2': 1, 'xc_3': 1, 'xc_4': 0, 'xc_5': 0, 'xc_6': 1, 'xc_7': 1, 'xc_8': 1, 'xc_9':
1, 'xv_0': 0, 'xv_1': 0, 'xv_10': 1, 'xv_11': 1, 'xv_2': 0, 'xv_3': 0, 'xv_4': 1, 'xv_5': 1, 'xv_6': 0, 'xv_7': 0, 'xv_8': 0, 'xv_9': 0}
Sample 3: {'xc_0': 1, 'xc_1': 0, 'xc_10': 1, 'xc_11': 0, 'xc_2': 1, 'xc_3': 1, 'xc_4': 0, 'xc_5': 1, 'xc_6': 1, 'xc_7': 1, 'xc_8': 0, 'xc_9':
1, 'xv_0': 0, 'xv_1': 1, 'xv_10': 0, 'xv_11': 1, 'xv_2': 0, 'xv_3': 0, 'xv_4': 1, 'xv_5': 0, 'xv_6': 0, 'xv_7': 0, 'xv_8': 1, 'xv_9': 0}
Sample 4: {'xc_0': 1, 'xc_1': 1, 'xc_10': 1, 'xc_11': 1, 'xc_2': 1, 'xc_3': 0, 'xc_4': 1, 'xc_5': 0, 'xc_6': 1, 'xc_7': 0, 'xc_8': 1, 'xc_9':
0, 'xv_0': 0, 'xv_1': 0, 'xv_10': 0, 'xv_11': 0, 'xv_2': 0, 'xv_3': 1, 'xv_4': 0, 'xv_5': 1, 'xv_6': 0, 'xv_7': 1, 'xv_8': 0, 'xv_9': 1}
```

oc1=(110011110011)
ov1=(001100001100)

oc2=(001111001111)
ov2=(110000110000)

oc3=(111100111100)
ov4=(000011000011)



3 exact solutions using Mathematic !

```
LowestStateOc={0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1}
LowestStateOv={1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0}
Qvalue=4
LowestStateOc={1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1}
LowestStateOv={0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0}
Qvalue=4
LowestStateOc={1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0}
LowestStateOv={0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1}
Qvalue=4
{431.75, Null}
```

3 lowest energy states using QA !

※ The Mathematica results and QA results are completely consistent!

QA

Problem Parameters | Solution | Timing

QPU_SAMPLING_TIME
165.34 ms

QPU_ANNEAL_TIME_PER_SAMPLE
20.0 µs

QPU_READOUT_TIME_PER_SAMPLE
124.76 µs

QPU_ACCESS_TIME
181.10276 ms

QPU_ACCESS_OVERHEAD_TIME
1.45624 ms

QPU_PROGRAMMING_TIME
15.76276 ms

QPU_DELAY_TIME_PER_SAMPLE
20.58 µs

POST_PROCESSING_OVERHEAD_TIME
48.0 µs

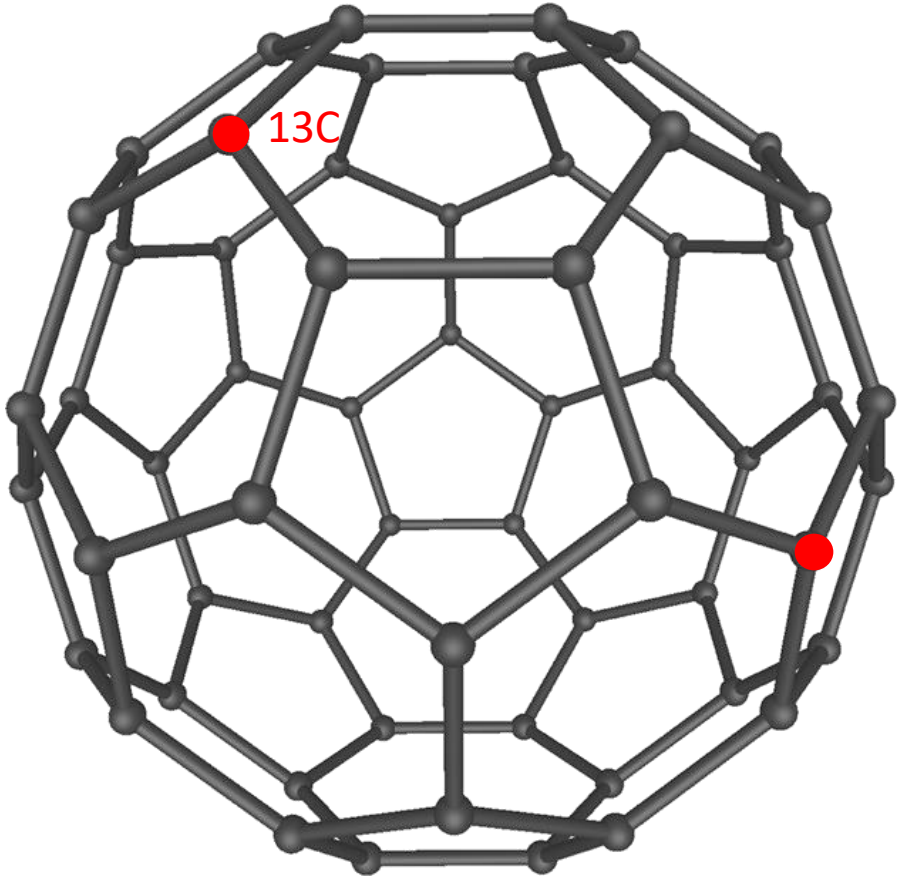
TOTAL_POST_PROCESSING_TIME
48.0 µs

Mathematica

```
LowestStateOc={0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1}
LowestStateOv={1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0}
Qvalue=6
LowestStateOc={0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1}
LowestStateOv={1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0}
Qvalue=6
LowestStateOc={0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1}
LowestStateOv={1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0}
Qvalue=4
LowestStateOc={1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1}
LowestStateOv={0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0}
Qvalue=4
LowestStateOc={1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0}
LowestStateOv={0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1}
Qvalue=4
= {431.75, Null}
```

※ We can see that it took roughly microseconds(µs) to milliseconds(ms).
This is a huge time savings compared to the previous ~ 430 seconds!

3. Example II: Fullerene C60 Isotopologues

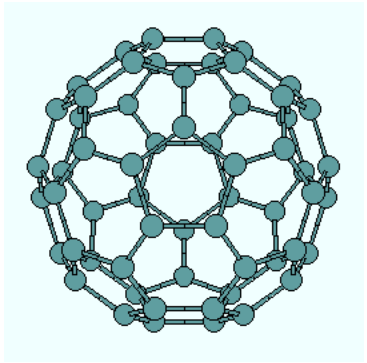


https://www.public.asu.edu/~cosmen/C60_vibrations/mode_assignments.htm

Four infrared active vibrational transitions of the fullerene C60

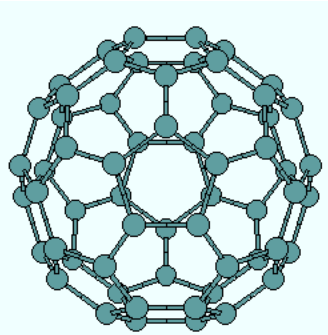
MODE	DEGENERACY	FREQ. (cm-1)
T1u(1)	3	**526**

19.01 μm



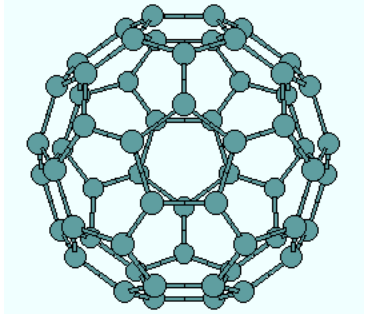
MODE	DEGENERACY	FREQ. (cm-1)
T1u(3)	3	**1182**

8.46 μm



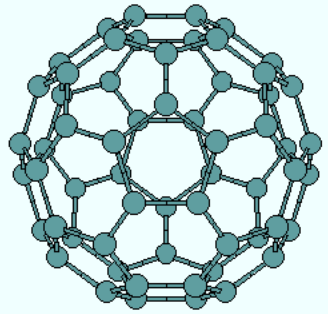
MODE	DEGENERACY	FREQ. (cm-1)
T1u(2)	3	575

17.39 μm



MODE	DEGENERACY	FREQ. (cm-1)
T1u(4)	3	**1429**

7.00 μm



Vibrational Spectroscopy of C60 (*José Menéndez and John B. Page*)

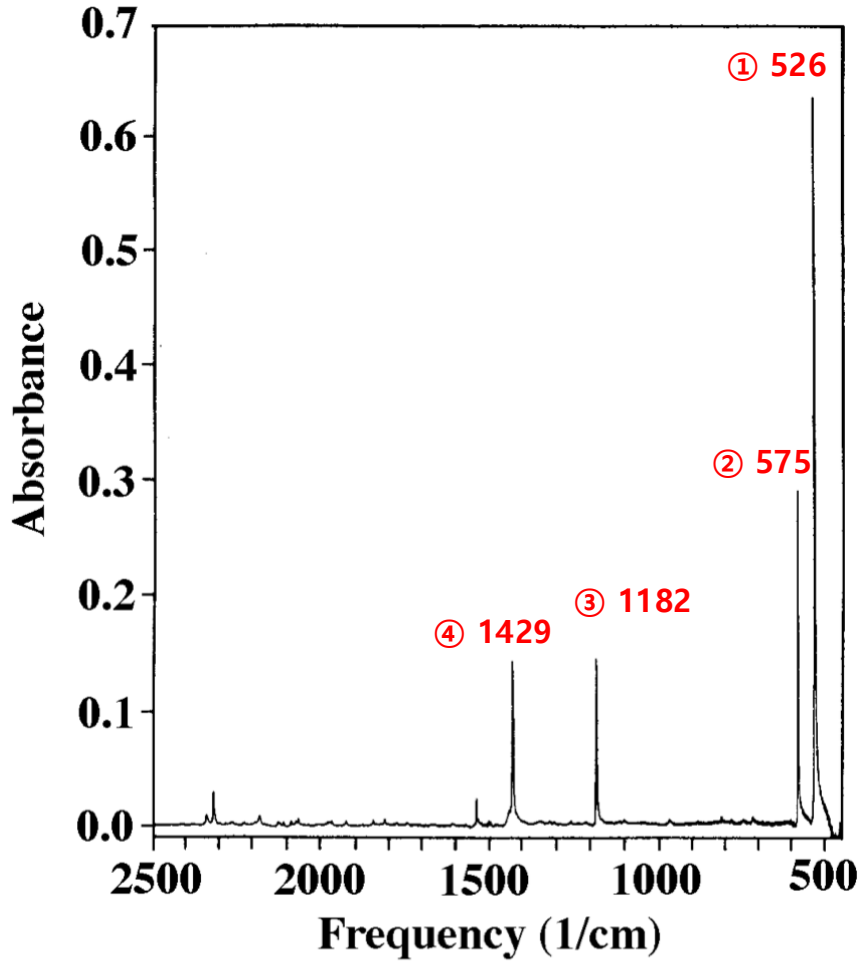


Fig. 1. Fourier transform IR spectra of a 1.4 mm thick film of C_{60}. After *Chase* et al. [84]

Quantitative Assessment of Isotope Effects

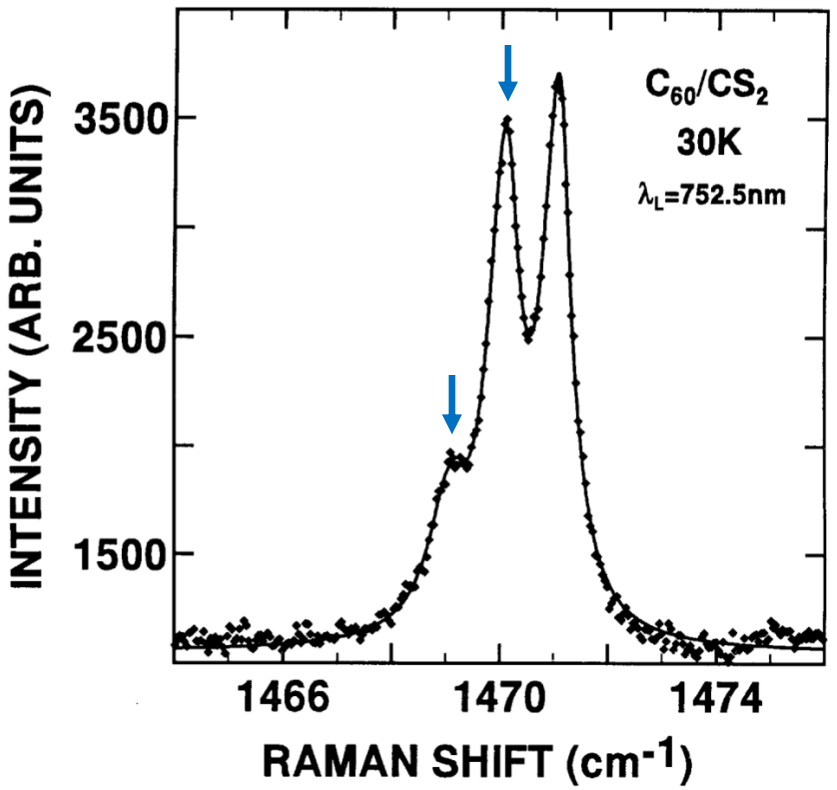
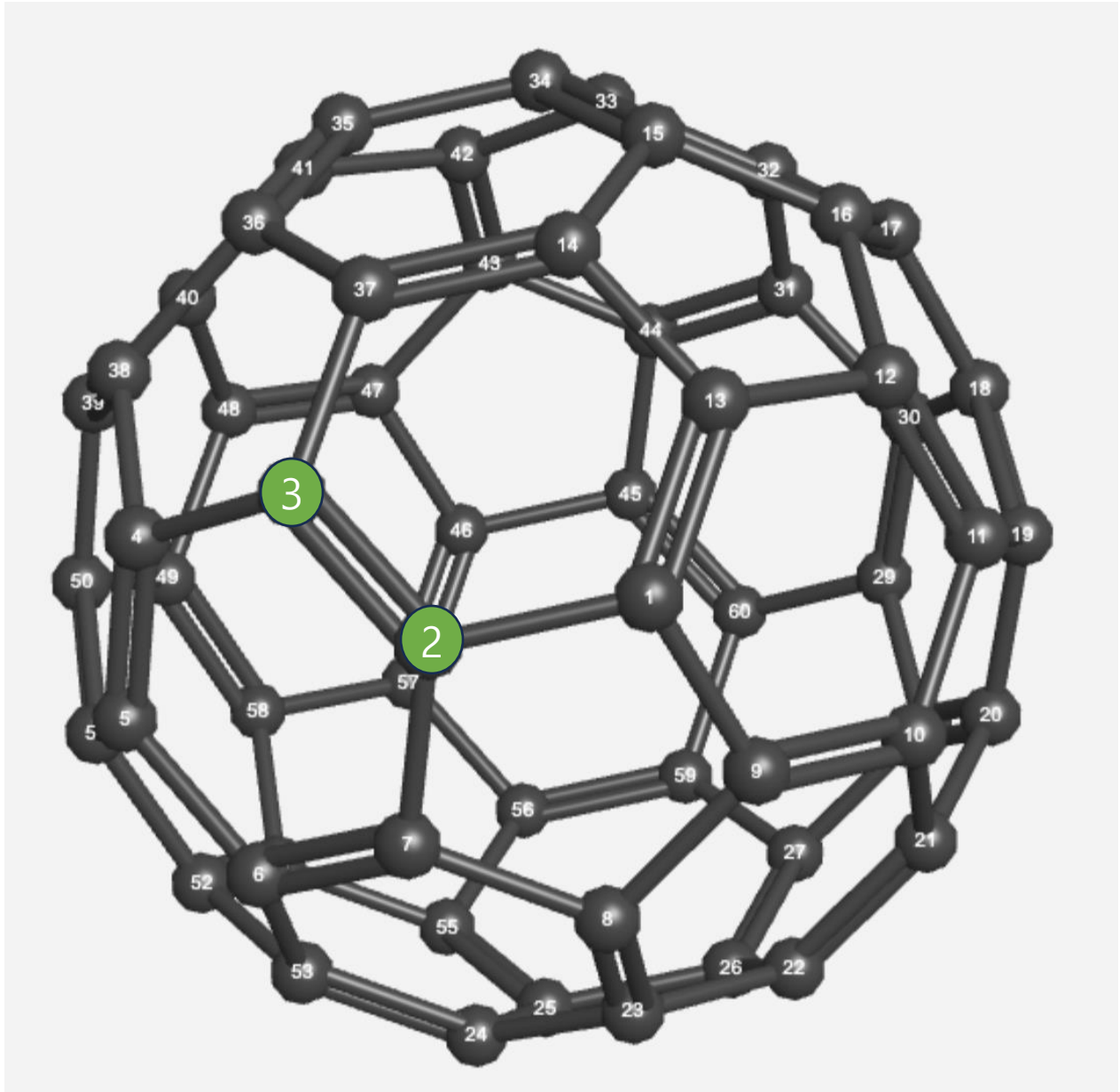


Fig. 3. Unpolarized Raman spectrum in the frequency region of the pentagonal pinch mode, for a frozen sample of non-isotopically enriched C_{60} in CS_2 at 30 K. The points represent the experimental data, while the solid curve is a 3-Lorentzian fit. The highest-frequency peak is assigned to the totally-symmetric pentagonal-pinch $A_g(2)$ mode in isotopically pure $^{12}C_{60}$. The other two peaks are assigned to the perturbed pentagonal-pinch mode in molecules having one and two ^{13}C isotopes, respectively. After *Guha* et al. [22]




```
13C(2_3)12C58-HF

0 1
C
C(Iso=13) 1 B1
C(Iso=13) 2 B2 1 A1
C 3 B3 2 A2 1 D1
C 4 B4 3 A3 2 D2
C 5 B5 4 A4 3 D3
C 2 B6 1 A5 3 D4
C 7 B7 2 A6 1 D5
C 1 B8 2 A7 3 D6
C 9 B9 1 A8 2 D7
C 10 B10 9 A9 1 D8
C 11 B11 10 A10 9 D9
C 1 B12 2 A11 3 D10
C 13 B13 1 A12 2 D11
C 14 B14 13 A13 1 D12
C 12 B15 11 A14 10 D13
C 16 B16 12 A15 11 D14
C 17 B17 16 A16 12 D15
C 11 B18 10 A17 9 D16
```

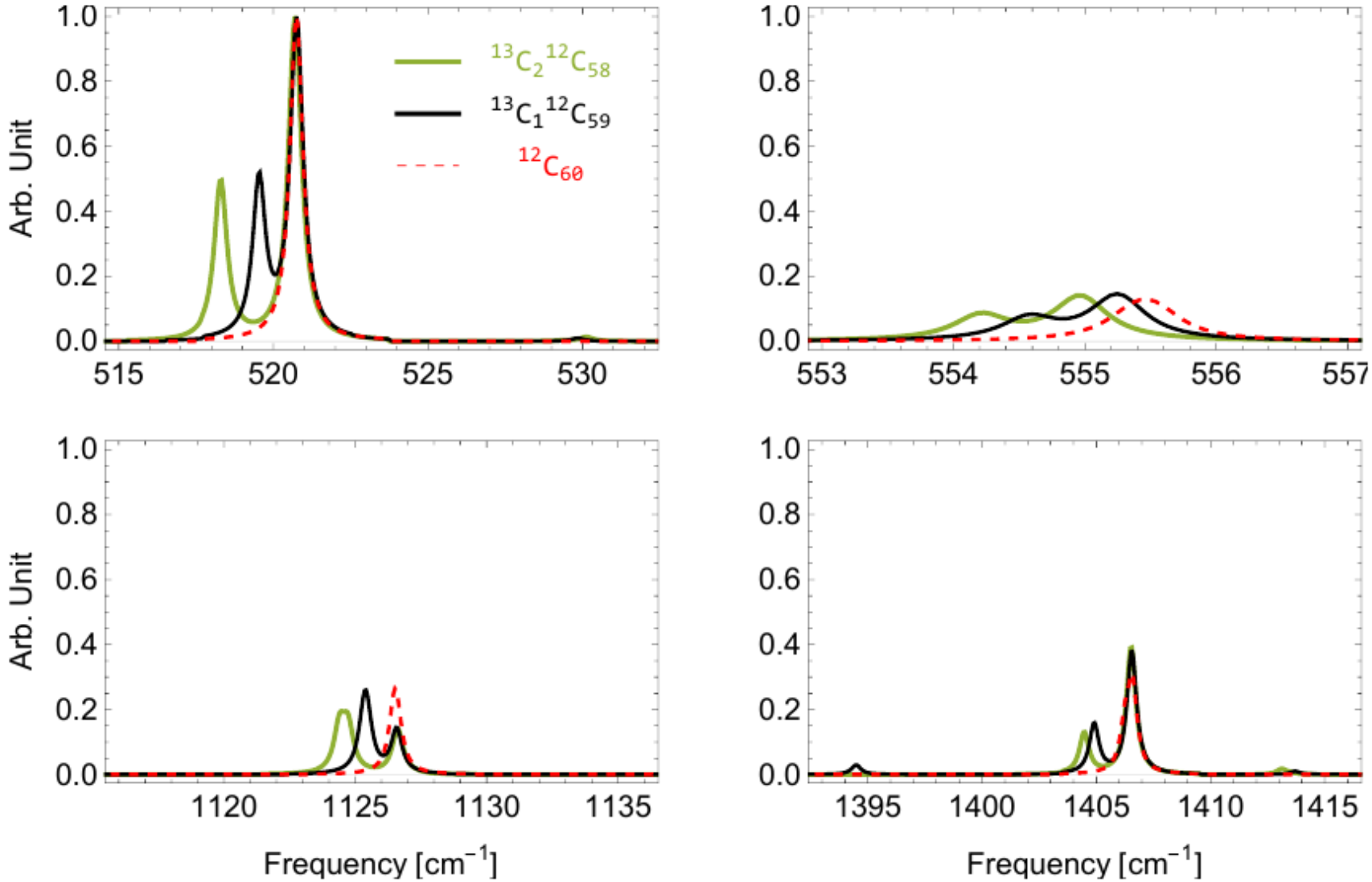


Figure 12: Calculated vibrational frequencies (cm^{-1}) of $^{13}C_2^{12}C_{58}$ (green thick line), $^{13}C_1^{12}C_{59}$ (black thick line) and $^{12}C_{60}$ (red dashed line) using the RHF method with the 3-21G basis set. The frequency shifts of about -1.02 cm^{-1} can be seen based on those of pure $^{12}C_{60}$.

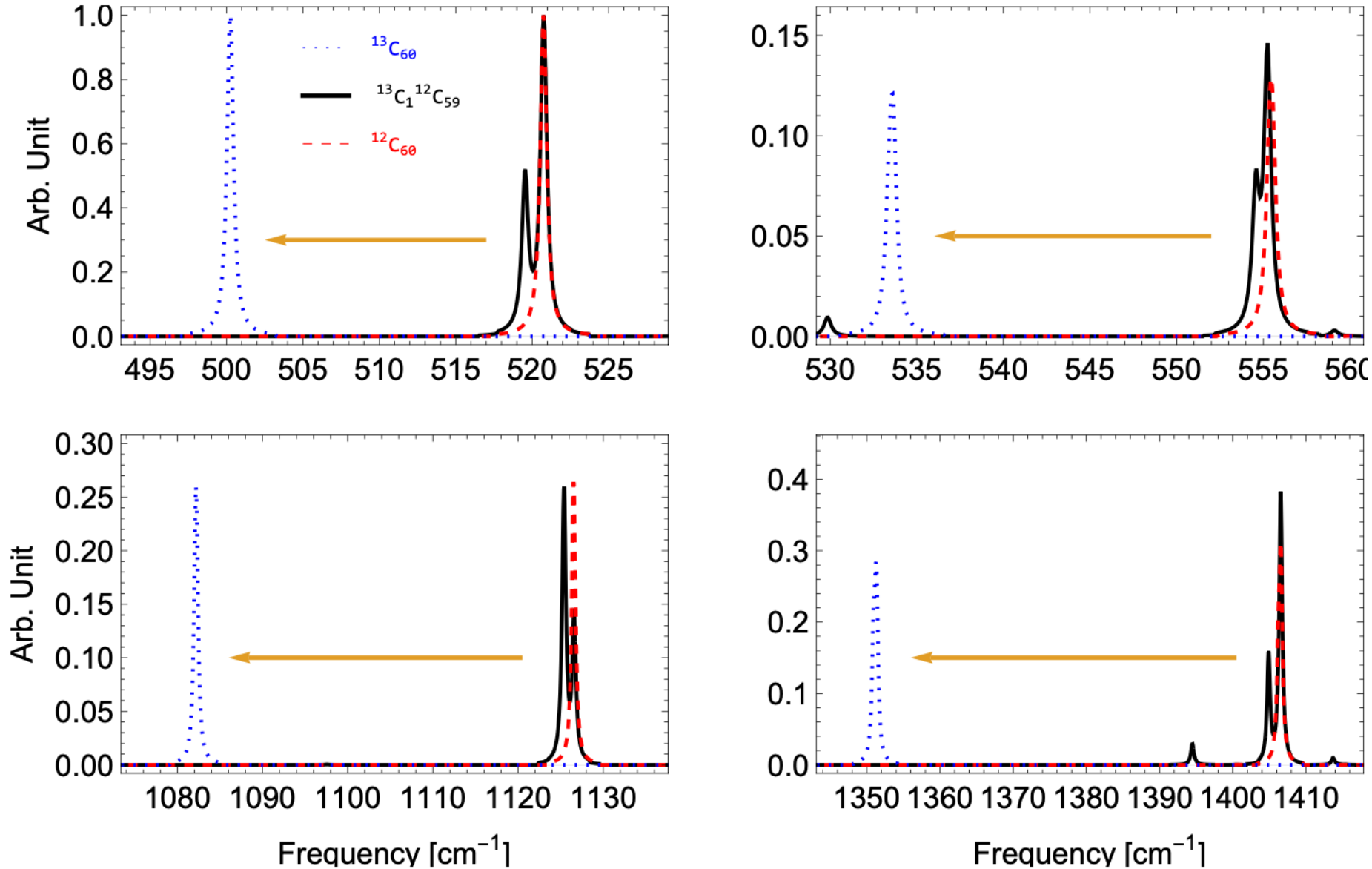
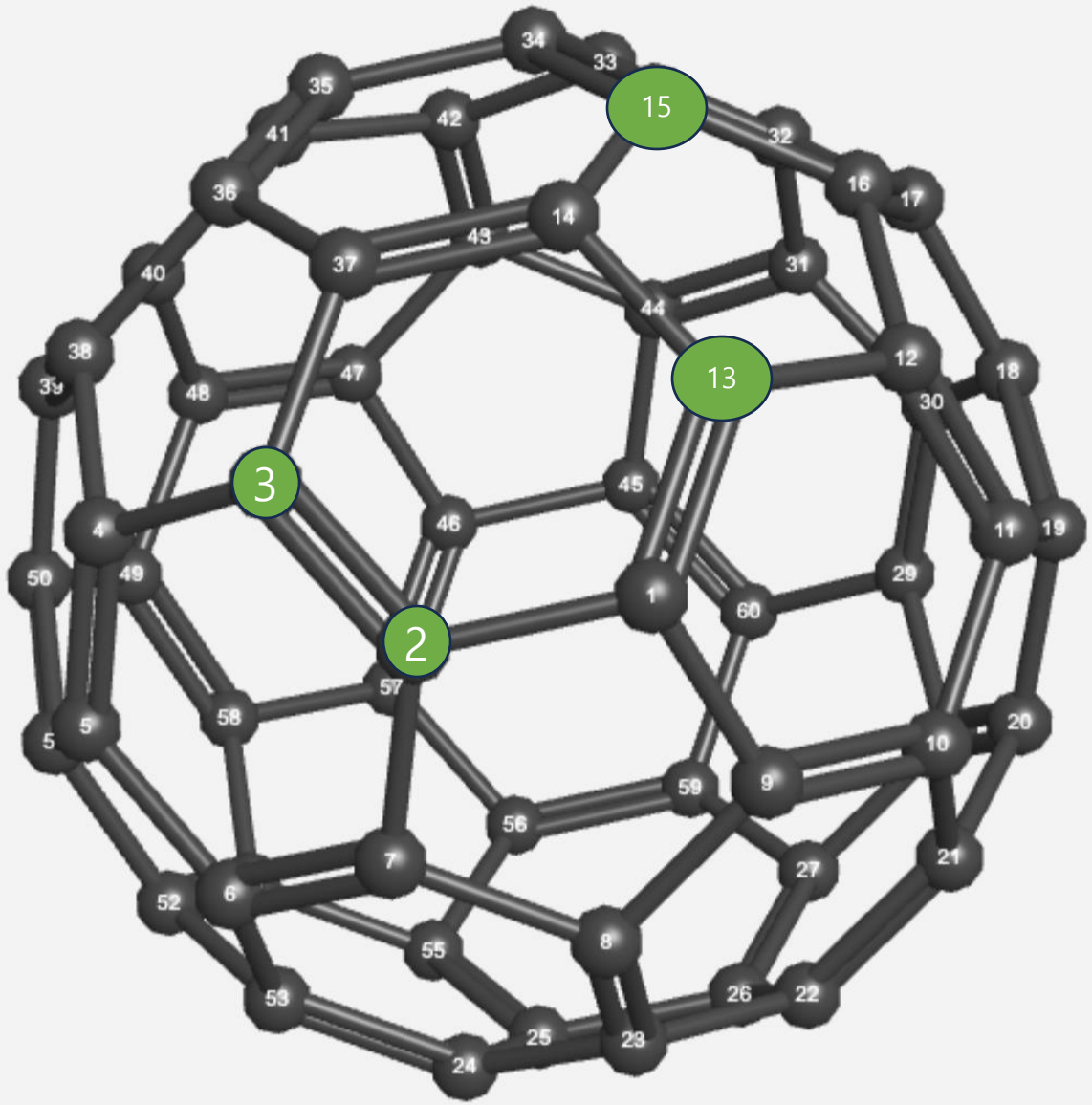


Figure 13: Calculated vibrational frequencies (cm^{-1}) of $^{13}C_1^{12}C_{59}$ (black thick line) and $^{12}C_{60}$ (red dashed line) using the RHF method with the 3-21G basis set. The frequency shifts of about -1.02 cm^{-1} can be seen based on those of pure $^{12}C_{60}$.

However, in order to obtain the IR spectrum according to the exact number of 13C, it is necessary to find the most stable configuration according to the number of 13C!

There are too many possible cases depending on the number and location of 13C.



The number of ways to choose 2 positions out of 60 is calculated using combinations as follows:

$$\binom{60}{2} = \frac{60!}{2!(60-2)!} = \frac{60 \times 59}{2 \times 1} = 1770$$

Therefore, the total number of ways to select 2 positions out of 60 is 1,770.

(For both states of 12C and 13C) ➡ $\sim 1700^2 \sim 10^{6\sim7}$

"To determine the number of ways to place stones in 30 out of 60 available positions, we can use combinations.

The number of combinations is calculated as follows:

$$\binom{60}{30} = \frac{60!}{30!(60-30)!} = \frac{60!}{30! \cdot 30!}$$

Calculating this yields an extremely large number, which can be approximated as:

$$\binom{60}{30} \approx 1.1829 \times 10^{17}$$

Therefore, the total number of ways to place stones in 30 out of 60 positions is approximately 1.1829×10^{17}."

(For both states of 12C and 13C) ➡ $\sim 10^{34}$

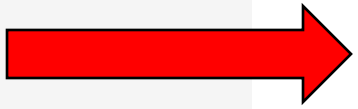
But actually, if we take symmetry into account, the number can be greatly reduced!

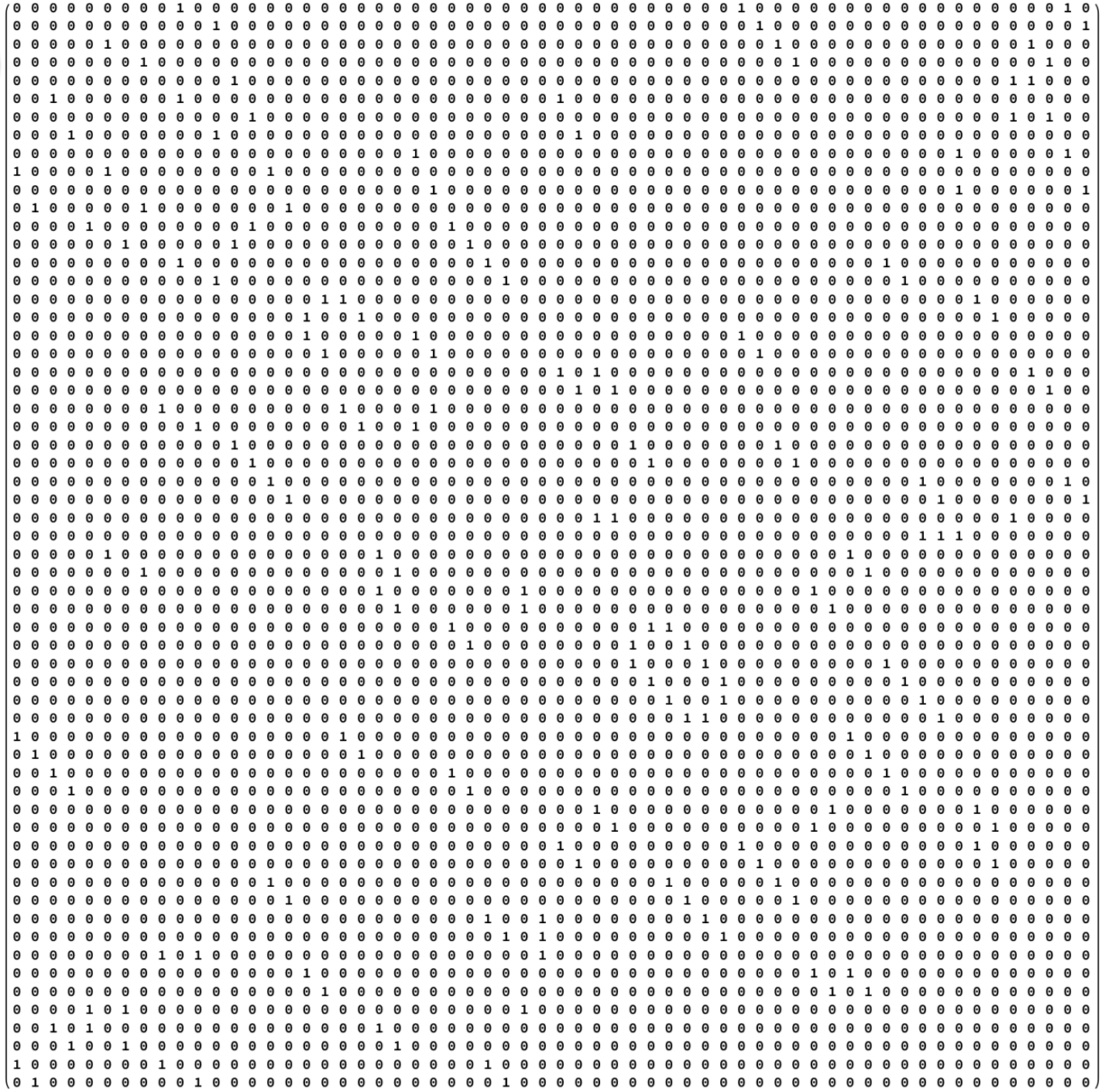
Adjacency Matrix A (60x60) for C60

```python
# Define the parameters for your QUBO
p0 = 10 # Penalty parameter (adjust based on your requirements)
p1 = 1 # Penalty parameter (adjust based on your requirements)
d = 12  # Adjust 'd' based on the problem size

# Placeholder for constants like N_alpha and adjacency matrix A
N_c = 8  # Example value for the number of the carbon atoms
N_v = 4  # Example value for the number of the vacancies

# The adjacency matrix A as provided
A = [
    [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
    [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
]
```





Using QA

```python
# Define the parameters for your QUBO
p0 = 10 # Penalty parameter (adjust based on your requirements)
p1 = 1 # Penalty parameter (adjust based on your requirements)
d = 60  # Adjust 'd' based on the problem size

# Placeholder for constants like N_alpha and adjacency matrix A
N_c12 = 60  # Example value for the number of the carbon atoms
N_c13 = 0  # Example value for the number of the vacancies

# The adjacency matrix A as provided
A = adj_matrix_np
```

```python
# QUBO 딕셔너리 초기화
Q = {}

# 첫 번째 항: p * sum((1 - x_{iαc} - x_{iαv}) ** 2)
for i in range(d):
    Q[(f'xc_{i}', f'xc_{i}')] = -p0
    Q[(f'xv_{i}', f'xv_{i}')] = -p0
    Q[(f'xc_{i}', f'xv_{i}')] = +2*p0

# 두 번째, 세 번째 항: (N_alpha - sum(x_{iαc}))**2 와 (N_alpha - sum(x_{iαv}))**2
for i in range(d):
    for j in range(i, d):
        if i == j:
            Q[(f'xc_{i}', f'xc_{i}')] += 1 -2 * N_c12
            Q[(f'xv_{i}', f'xv_{i}')] += 1 -2 * N_c13
        else:
            Q[(f'xc_{i}', f'xc_{j}')] = +2
            Q[(f'xv_{i}', f'xv_{j}')] = +2

# 네 번째 항: 인접 행렬에 따른 x_{iαc}와 x_{jαv}의 상호작용 항 (Aij * x_{iαc} * x_{jαv})
for i in range(d):
    for j in range(i+1, d):
            #print(i, j)
            Q[(f'xc_{i}', f'xv_{j}')] = A[i][j]
            Q[(f'xc_{j}', f'xv_{i}')] = A[i][j]
```

```python
# Initialize the sampler and submit the QUBO problem to D-Wave
sampler = EmbeddingComposite(DWaveSampler())
sampleset = sampler.sample_qubo(Q, num_reads=1000)

# Print the results
print(sampleset)
```

```
     xc_0 xc_1 xc_10 xc_11 xc_12 xc_13 xc_14 xc_15 ... xv_9  energy num_oc. ...
0       1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
3       1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
7       1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
10      1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
32      1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
69      1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
165     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
181     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
204     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
229     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
241     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
271     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
341     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
371     1    1     1     1     1     1     1     1 ...    0 -4200.0       1 ...
2       1    1     1     1     1     1     1     1 ...    0 -4198.0       1 ...
15      1    1     1     1     1     1     1     1 ...    0 -4198.0       1 ...
37      1    1     1     1     1     1     1     1 ...    0 -4198.0       1 ...
```

```python
# 전체 결과를 리스트로 변환 후 상위 5개의 결과 출력
samples_list = list(sampleset)
for i, sample in enumerate(samples_list[:5]):
    print(f"Sample {i}: {sample}")
```

```
Sample 0: {'xc_0': 1, 'xc_1': 1, 'xc_10': 1, 'xc_11': 1, 'xc_12': 1, 'xc_13': 1, 'xc_14': 1, 'xc_15': 1, 'xc_16': 1, 'xc_17': 1, 'xc_18': 1,
'xc_19': 1, 'xc_2': 1, 'xc_20': 1, 'xc_21': 1, 'xc_22': 1, 'xc_23': 1, 'xc_24': 1, 'xc_25': 1, 'xc_26': 1, 'xc_27': 1, 'xc_28': 1, 'xc_29':
1, 'xc_3': 1, 'xc_30': 1, 'xc_31': 1, 'xc_32': 1, 'xc_33': 1, 'xc_34': 1, 'xc_35': 1, 'xc_36': 1, 'xc_37': 1, 'xc_38': 1, 'xc_39': 1, 'xc_4':
1, 'xc_40': 1, 'xc_41': 1, 'xc_42': 1, 'xc_43': 1, 'xc_44': 1, 'xc_45': 1, 'xc_46': 1, 'xc_47': 1, 'xc_48': 1, 'xc_49': 1, 'xc_5': 1, 'xc_5
0': 1, 'xc_51': 1, 'xc_52': 1, 'xc_53': 1, 'xc_54': 1, 'xc_55': 1, 'xc_56': 1, 'xc_57': 1, 'xc_58': 1, 'xc_59': 1, 'xc_6': 1, 'xc_7': 1, 'xc_
8': 1, 'xc_9': 1, 'xv_0': 0, 'xv_1': 0, 'xv_10': 0, 'xv_11': 0, 'xv_12': 0, 'xv_13': 0, 'xv_14': 0, 'xv_15': 0, 'xv_16': 0, 'xv_17': 0, 'xv_1
8': 0, 'xv_19': 0, 'xv_2': 0, 'xv_20': 0, 'xv_21': 0, 'xv_22': 0, 'xv_23': 0, 'xv_24': 0, 'xv_25': 0, 'xv_26': 0, 'xv_27': 0, 'xv_28': 0, 'xv
_29': 0, 'xv_3': 0, 'xv_30': 0, 'xv_31': 0, 'xv_32': 0, 'xv_33': 0, 'xv_34': 0, 'xv_35': 0, 'xv_36': 0, 'xv_37': 0, 'xv_38': 0, 'xv_39': 0,
'xv_4': 0, 'xv_40': 0, 'xv_41': 0, 'xv_42': 0, 'xv_43': 0, 'xv_44': 0, 'xv_45': 0, 'xv_46': 0, 'xv_47': 0, 'xv_48': 0, 'xv_49': 0, 'xv_5': 0,
'xv_50': 0, 'xv_51': 0, 'xv_52': 0, 'xv_53': 0, 'xv_54': 0, 'xv_55': 0, 'xv_56': 0, 'xv_57': 0, 'xv_58': 0, 'xv_59': 0, 'xv_6': 0, 'xv_7': 0,
'xv_8': 0, 'xv_9': 0}
Sample 1: {'xc_0': 1, 'xc_1': 1, 'xc_10': 1, 'xc_11': 1, 'xc_12': 1, 'xc_13': 1, 'xc_14': 1, 'xc_15': 1, 'xc_16': 1, 'xc_17': 1, 'xc_18': 1,
'xc_19': 1, 'xc_2': 1, 'xc_20': 1, 'xc_21': 1, 'xc_22': 1, 'xc_23': 1, 'xc_24': 1, 'xc_25': 1, 'xc_26': 1, 'xc_27': 1, 'xc_28': 1, 'xc_29':
1, 'xc_3': 1, 'xc_30': 1, 'xc_31': 1, 'xc_32': 1, 'xc_33': 1, 'xc_34': 1, 'xc_35': 1, 'xc_36': 1, 'xc_37': 1, 'xc_38': 1, 'xc_39': 1, 'xc_4':
1, 'xc_40': 1, 'xc_41': 1, 'xc_42': 1, 'xc_43': 1, 'xc_44': 1, 'xc_45': 1, 'xc_46': 1, 'xc_47': 1, 'xc_48': 1, 'xc_49': 1, 'xc_5': 1, 'xc_5
0': 1, 'xc_51': 1, 'xc_52': 1, 'xc_53': 1, 'xc_54': 1, 'xc_55': 1, 'xc_56': 1, 'xc_57': 1, 'xc_58': 1, 'xc_59': 1, 'xc_6': 1, 'xc_7': 1, 'xc_
8': 1, 'xc_9': 1, 'xv_0': 0, 'xv_1': 0, 'xv_10': 0, 'xv_11': 0, 'xv_12': 0, 'xv_13': 0, 'xv_14': 0, 'xv_15': 0, 'xv_16': 0, 'xv_17': 0, 'xv_1
8': 0, 'xv_19': 0, 'xv_2': 0, 'xv_20': 0, 'xv_21': 0, 'xv_22': 0, 'xv_23': 0, 'xv_24': 0, 'xv_25': 0, 'xv_26': 0, 'xv_27': 0, 'xv_28': 0, 'xv
_29': 0, 'xv_3': 0, 'xv_30': 0, 'xv_31': 0, 'xv_32': 0, 'xv_33': 0, 'xv_34': 0, 'xv_35': 0, 'xv_36': 0, 'xv_37': 0, 'xv_38': 0, 'xv_39': 0,
'xv_4': 0, 'xv_40': 0, 'xv_41': 0, 'xv_42': 0, 'xv_43': 0, 'xv_44': 0, 'xv_45': 0, 'xv_46': 0, 'xv_47': 0, 'xv_48': 0, 'xv_49': 0, 'xv_5': 0,
'xv_50': 0, 'xv_51': 0, 'xv_52': 0, 'xv_53': 0, 'xv_54': 0, 'xv_55': 0, 'xv_56': 0, 'xv_57': 0, 'xv_58': 0, 'xv_59': 0, 'xv_6': 0, 'xv_7': 0,
'xv_8': 0, 'xv_9': 0}
Sample 2: {'xc_0': 1, 'xc_1': 1, 'xc_10': 1, 'xc_11': 1, 'xc_12': 1, 'xc_13': 1, 'xc_14': 1, 'xc_15': 1, 'xc_16': 1, 'xc_17': 1, 'xc_18': 1,
'xc_19': 1, 'xc_2': 1, 'xc_20': 1, 'xc_21': 1, 'xc_22': 1, 'xc_23': 1, 'xc_24': 1, 'xc_25': 1, 'xc_26': 1, 'xc_27': 1, 'xc_28': 1, 'xc_29':
1, 'xc_3': 1, 'xc_30': 1, 'xc_31': 1, 'xc_32': 1, 'xc_33': 1, 'xc_34': 1, 'xc_35': 1, 'xc_36': 1, 'xc_37': 1, 'xc_38': 1, 'xc_39': 1, 'xc_4':
1, 'xc_40': 1, 'xc_41': 1, 'xc_42': 1, 'xc_43': 1, 'xc_44': 1, 'xc_45': 1, 'xc_46': 1, 'xc_47': 1, 'xc_48': 1, 'xc_49': 1, 'xc_5': 1, 'xc_5
0': 1, 'xc_51': 1, 'xc_52': 1, 'xc_53': 1, 'xc_54': 1, 'xc_55': 1, 'xc_56': 1, 'xc_57': 1, 'xc_58': 1, 'xc_59': 1, 'xc_6': 1, 'xc_7': 1, 'xc_
8': 1, 'xc_9': 1, 'xv_0': 0, 'xv_1': 0, 'xv_10': 0, 'xv_11': 0, 'xv_12': 0, 'xv_13': 0, 'xv_14': 0, 'xv_15': 0, 'xv_16': 0, 'xv_17': 0, 'xv_1
8': 0, 'xv_19': 0, 'xv_2': 0, 'xv_20': 0, 'xv_21': 0, 'xv_22': 0, 'xv_23': 0, 'xv_24': 0, 'xv_25': 0, 'xv_26': 0, 'xv_27': 0, 'xv_28': 0, 'xv
_29': 0, 'xv_3': 0, 'xv_30': 0, 'xv_31': 0, 'xv_32': 0, 'xv_33': 0, 'xv_34': 0, 'xv_35': 0, 'xv_36': 0, 'xv_37': 0, 'xv_38': 0, 'xv_39': 0,
'xv_4': 0, 'xv_40': 0, 'xv_41': 0, 'xv_42': 0, 'xv_43': 0, 'xv_44': 0, 'xv_45': 0, 'xv_46': 0, 'xv_47': 0, 'xv_48': 0, 'xv_49': 0, 'xv_5': 0,
'xv_50': 0, 'xv_51': 0, 'xv_52': 0, 'xv_53': 0, 'xv_54': 0, 'xv_55': 0, 'xv_56': 0, 'xv_57': 0, 'xv_58': 0, 'xv_59': 0, 'xv_6': 0, 'xv_7': 0,
'xv_8': 0, 'xv_9': 0}
```

oc1=(1111111111111111111111111111111111.....111111)
ov1=(0000000000000000000000000000000000.....000000)

```
# Placeholder for constants like N_alpha and adjacency matrix A
N_c12 = 60  # Example value for the number of the carbon atoms
N_c13 = 0  # Example value for the number of the vacancies
```

12 qubits(12C) + 12 qubits(vacancies)

Problem Parameters | Solution | Timing

QPU_SAMPLING_TIME
165.34 ms

QPU_ANNEAL_TIME_PER_SAMPLE
20.0 µs

QPU_READOUT_TIME_PER_SAMPLE
124.76 µs

QPU_ACCESS_TIME
181.10276 ms

QPU_ACCESS_OVERHEAD_TIME
1.45624 ms

QPU_PROGRAMMING_TIME
15.76276 ms

QPU_DELAY_TIME_PER_SAMPLE
20.58 µs

POST_PROCESSING_OVERHEAD_TIME
48.0 µs

TOTAL_POST_PROCESSING_TIME
48.0 µs

60 qubits(12C) + 60 qubits(13C)

Problem Parameters | Solution | Timing

QPU_SAMPLING_TIME
269.42 ms

QPU_ANNEAL_TIME_PER_SAMPLE
20.0 µs

QPU_READOUT_TIME_PER_SAMPLE
228.84 µs

QPU_ACCESS_TIME
285.18356 ms

QPU_ACCESS_OVERHEAD_TIME
8.09944 ms

QPU_PROGRAMMING_TIME
15.76356 ms

QPU_DELAY_TIME_PER_SAMPLE
20.58 µs

POST_PROCESSING_OVERHEAD_TIME
61.0 µs

TOTAL_POST_PROCESSING_TIME
61.0 µs

```
# Placeholder for constants like N_alpha and adjacency matrix A
N_c12 = 30  # Example value for the number of the carbon atoms
N_c13 = 30  # Example value for the number of the vacancies
```

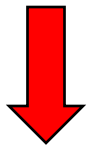
```
# Initialize the sampler and submit the QUBO problem to D-Wave
sampler2 = EmbeddingComposite(DWaveSampler())
sampleset2 = sampler2.sample_qubo(Q, num_reads=1000)

# Print the results
print(sampleset2)
```

```
     xc_0 xc_1 xc_10 xc_11 xc_12 xc_13 xc_14 xc_15 ... xv_9  energy num_oc. ...
535     0    1     1     1     0     1     0     0 ...    1 -2244.0       1 ...
119     0    1     0     1     0     1     0     0 ...    0 -2229.0       1 ...
189     0    1     0     1     0     1     0     0 ...    1 -2220.0       1 ...
776     0    1     1     1     0     1     0     0 ...    0 -2220.0       1 ...
809     0    1     0     1     0     1     0     0 ...    0 -2220.0       1 ...
440     0    1     1     1     0     1     0     0 ...    1 -2218.0       1 ...
85      0    1     1     1     0     1     0     0 ...    1 -2215.0       1 ...
160     0    1     1     1     0     1     0     0 ...    0 -2214.0       1 ...
138     0    1     1     1     0     1     0     0 ...    0 -2212.0       1 ...
528     0    1     1     1     0     1     0     0 ...    0 -2212.0       1 ...
679     0    0     0     1     0     1     0     0 ...    1 -2211.0       1 ...
163     0    0     1     1     0     1     0     0 ...    1 -2210.0       1 ...
669     0    1     1     1     0     1     0     0 ...    1 -2210.0       1 ...
321     0    1     0     1     0     1     0     0 ...    0 -2208.0       1 ...
47      0    1     0     1     0     1     0     0 ...    0 -2207.0       1 ...
304     0    1     1     1     0     1     0     0 ...    0 -2205.0       1 ...
582     0    1     0     1     0     1     0     0 ...    1 -2205.0       1 ...
```

```python
# 전체 결과를 리스트로 변환 후 상위 5개의 결과 출력
samples_list2 = list(sampleset2)
for i, sample in enumerate(samples_list2[:5]):
    print(f"Sample {i}: {sample}")
```

```
Sample 0: {'xc_0': 0, 'xc_1': 1, 'xc_10': 1, 'xc_11': 1, 'xc_12': 0, 'xc_13': 1, 'xc_14': 0, 'xc_15': 0, 'xc_16': 0, 'xc_17': 0, 'xc_18': 0,
'xc_19': 0, 'xc_2': 1, 'xc_20': 1, 'xc_21': 1, 'xc_22': 1, 'xc_23': 0, 'xc_24': 1, 'xc_25': 1, 'xc_26': 0, 'xc_27': 0, 'xc_28': 0, 'xc_29':
1, 'xc_3': 1, 'xc_30': 1, 'xc_31': 1, 'xc_32': 0, 'xc_33': 0, 'xc_34': 1, 'xc_35': 0, 'xc_36': 0, 'xc_37': 1, 'xc_38': 1, 'xc_39': 0, 'xc_4':
1, 'xc_40': 1, 'xc_41': 0, 'xc_42': 0, 'xc_43': 1, 'xc_44': 0, 'xc_45': 0, 'xc_46': 1, 'xc_47': 1, 'xc_48': 1, 'xc_49': 1, 'xc_5': 1, 'xc_5
0': 0, 'xc_51': 0, 'xc_52': 0, 'xc_53': 0, 'xc_54': 1, 'xc_55': 1, 'xc_56': 1, 'xc_57': 0, 'xc_58': 0, 'xc_59': 1, 'xc_6': 1, 'xc_7': 0, 'xc_
8': 0, 'xc_9': 0, 'xv_0': 1, 'xv_1': 0, 'xv_10': 0, 'xv_11': 0, 'xv_12': 1, 'xv_13': 0, 'xv_14': 1, 'xv_15': 1, 'xv_16': 1, 'xv_17': 1, 'xv_1
8': 1, 'xv_19': 1, 'xv_2': 0, 'xv_20': 1, 'xv_21': 0, 'xv_22': 0, 'xv_23': 1, 'xv_24': 0, 'xv_25': 0, 'xv_26': 1, 'xv_27': 0, 'xv_28': 1, 'xv
_29': 0, 'xv_3': 0, 'xv_30': 0, 'xv_31': 1, 'xv_32': 1, 'xv_33': 1, 'xv_34': 0, 'xv_35': 1, 'xv_36': 1, 'xv_37': 0, 'xv_38': 1, 'xv_39': 1,
'xv_4': 0, 'xv_40': 0, 'xv_41': 1, 'xv_42': 1, 'xv_43': 0, 'xv_44': 1, 'xv_45': 1, 'xv_46': 0, 'xv_47': 0, 'xv_48': 1, 'xv_49': 0, 'xv_5': 0,
'xv_50': 1, 'xv_51': 1, 'xv_52': 0, 'xv_53': 0, 'xv_54': 1, 'xv_55': 0, 'xv_56': 0, 'xv_57': 1, 'xv_58': 1, 'xv_59': 0, 'xv_6': 1, 'xv_7': 1,
'xv_8': 0, 'xv_9': 1}
Sample 1: {'xc_0': 0, 'xc_1': 1, 'xc_10': 0, 'xc_11': 1, 'xc_12': 0, 'xc_13': 1, 'xc_14': 0, 'xc_15': 0, 'xc_16': 0, 'xc_17': 0, 'xc_18': 0,
'xc_19': 0, 'xc_2': 1, 'xc_20': 0, 'xc_21': 1, 'xc_22': 1, 'xc_23': 0, 'xc_24': 1, 'xc_25': 1, 'xc_26': 0, 'xc_27': 1, 'xc_28': 1, 'xc_29':
0, 'xc_3': 1, 'xc_30': 1, 'xc_31': 0, 'xc_32': 0, 'xc_33': 0, 'xc_34': 1, 'xc_35': 0, 'xc_36': 1, 'xc_37': 1, 'xc_38': 0, 'xc_39': 0, 'xc_4':
1, 'xc_40': 1, 'xc_41': 1, 'xc_42': 0, 'xc_43': 1, 'xc_44': 0, 'xc_45': 0, 'xc_46': 1, 'xc_47': 1, 'xc_48': 1, 'xc_49': 1, 'xc_5': 1, 'xc_5
0': 0, 'xc_51': 0, 'xc_52': 0, 'xc_53': 1, 'xc_54': 1, 'xc_55': 1, 'xc_56': 1, 'xc_57': 0, 'xc_58': 0, 'xc_59': 1, 'xc_6': 1, 'xc_7': 0, 'xc_
8': 0, 'xc_9': 0, 'xv_0': 1, 'xv_1': 0, 'xv_10': 1, 'xv_11': 0, 'xv_12': 1, 'xv_13': 0, 'xv_14': 1, 'xv_15': 0, 'xv_16': 1, 'xv_17': 1, 'xv_1
8': 1, 'xv_19': 1, 'xv_2': 1, 'xv_20': 1, 'xv_21': 0, 'xv_22': 0, 'xv_23': 1, 'xv_24': 0, 'xv_25': 0, 'xv_26': 1, 'xv_27': 0, 'xv_28': 0, 'xv
_29': 1, 'xv_3': 0, 'xv_30': 0, 'xv_31': 1, 'xv_32': 1, 'xv_33': 1, 'xv_34': 0, 'xv_35': 0, 'xv_36': 1, 'xv_37': 0, 'xv_38': 1, 'xv_39': 1,
'xv_4': 0, 'xv_40': 0, 'xv_41': 1, 'xv_42': 1, 'xv_43': 0, 'xv_44': 1, 'xv_45': 1, 'xv_46': 0, 'xv_47': 1, 'xv_48': 1, 'xv_49': 0, 'xv_5': 0,
'xv_50': 1, 'xv_51': 1, 'xv_52': 0, 'xv_53': 1, 'xv_54': 1, 'xv_55': 0, 'xv_56': 0, 'xv_57': 1, 'xv_58': 1, 'xv_59': 0, 'xv_6': 0, 'xv_7': 1,
'xv_8': 1, 'xv_9': 0}
```



oc1=(01111110001101000000111011000111001001101001001111000011100 1)
ov1=(10000001110010111111000100111000110110010110110000111100011 0)

60 qubits(12C) + 60 qubits(13C)

N_c12 = 60, N_c13 = 0

Problem Parameters Solution Timing

QPU_SAMPLING_TIME
269.42 ms

QPU_ANNEAL_TIME_PER_SAMPLE
20.0 µs

QPU_READOUT_TIME_PER_SAMPLE
228.84 µs

QPU_ACCESS_TIME
285.18356 ms

QPU_ACCESS_OVERHEAD_TIME
8.09944 ms

QPU_PROGRAMMING_TIME
15.76356 ms

QPU_DELAY_TIME_PER_SAMPLE
20.58 µs

POST_PROCESSING_OVERHEAD_TIME
61.0 µs

TOTAL_POST_PROCESSING_TIME
61.0 µs

60 qubits(12C) + 60 qubits(13C)

N_c12 = 30, N_c13 = 30

Problem Parameters Solution Timing

QPU_SAMPLING_TIME
234.08 ms

QPU_ANNEAL_TIME_PER_SAMPLE
20.0 µs

QPU_READOUT_TIME_PER_SAMPLE
193.5 µs

QPU_ACCESS_TIME
249.84436 ms

QPU_ACCESS_OVERHEAD_TIME
5.34164 ms

QPU_PROGRAMMING_TIME
15.76436 ms

QPU_DELAY_TIME_PER_SAMPLE
20.58 µs

POST_PROCESSING_OVERHEAD_TIME
78.0 µs

TOTAL_POST_PROCESSING_TIME
78.0 µs

Conclusion

- We demonstrated the effective use of **quantum annealing** in solving complex optimization problems in materials science.

- In the **graphene vacancy model**, the D-Wave quantum annealer successfully identified energetically favorable configurations in microseconds—showing excellent agreement with classical methods while achieving significant speedup.

- For **C_{60} isotopologues**, we tackled the combinatorial explosion in ^{13}C substitution configurations. Quantum annealing allowed us to efficiently identify stable atomic arrangements relevant to vibrational spectroscopy.

- Our approach shows how **DFT-based spectral analysis** and **quantum optimization** can be synergistically combined, offering a promising direction for **isotopic modeling in astrochemistry and nanomaterials**.

Thank you for your attention!