

Classical Solvers :

To solve a problem classically on your local machine, you configure a classical solver, either one of those included in the Ocean tools or your own.

Among several samplers provided in the dimod tool for testing your code locally, is the **ExactSolver()** that calculates the energy of all possible samples for a given problem.

- A simple exact solver for testing and debugging code using your local CPU.
- This solver becomes slow for problems with 18 or more variables.
- If you use a classical solver running locally on your CPU, a single sample might provide the optimal solution.

Method : ExactSolver.sample_qubo(Q, **parameters) :

- This method is inherited from the Sampler base class.
- Converts the quadratic unconstrained binary optimization (QUBO) into a BinaryQuadraticModel and then calls sample().

Example of obj function : $2ab+2bc+2ac-a-b-c+1$

```
import dimod
from dimod import ExactSolver

Q = {(0, 1): 2, (0, 2): 2, (1, 2): 2, (0, 0): -1, (1, 1): -1, (2, 2): -1}
exactsolver = dimod.ExactSolver()
response = exactsolver.sample_qubo(Q)

# Print the best sample and its energy
best_sample = response.first.sample
best_energy = response.first.energy
print("Best sample:", best_sample)
print("Best energy:", best_energy)
print("\n\n All possible solution")

# Print the all results
for sample, energy in response.data(['sample', 'energy']):
    print(sample, energy)

Best sample: {0: 1, 1: 0, 2: 0, 3: 0}
Best energy: -1.0

All possible solution
{0: 1, 1: 0, 2: 0, 3: 0} -1.0
{0: 0, 1: 1, 2: 0, 3: 0} -1.0
{0: 0, 1: 0, 2: 1, 3: 0} -1.0
{0: 0, 1: 0, 2: 1, 3: 1} -1.0
```

Reference Link :

<https://colab.research.google.com/drive/1-Fx1RXFcHWQw4mlt3u4CmYKBwYiiQB0J?usp=sharing>

Quantum Solvers :

Ocean's dwave-system tool enables you to use a D-Wave system as a sampler. In addition to DWaveSampler, the tool provides a **EmbeddingComposite** composite that maps unstructured problems to the graph structure of the selected sampler, a process known as **minor-embedding**.

- *To solve an arbitrarily posed binary quadratic problem directly on a D-Wave system requires mapping, called minor embedding, to the QPU Topology of the system's quantum processing unit (QPU).*
- *This preprocessing can be done by a composed sampler consisting of the **DWaveSampler()** and a composite that performs minor-embedding.*
- *This step is handled automatically by **LeapHybridSampler()** and **dwave-hybrid** reference samplers.*

DWaveSampler() :

- A class for using the D-Wave system as a sampler for binary quadratic models.
- **Method :** DWaveSampler.sample_qubo(Q, **parameters)
 - This method is inherited from the Sampler base class.
 - Converts the QUBO into a BinaryQuadraticModel and then calls sample().

EmbeddingComposite :

- Maps problems to a structured sampler.
- Automatically minor-embeds a problem into a structured sampler such as a D-Wave system.
- A new minor-embedding is calculated each time one of its sampling methods is called.
- The EmbeddingComposite uses the minorminer library to map unstructured problems to a structured sampler.
- **Parameters :**
 - child_sampler (dimod.Sampler)
 - find_embedding (function, optional)
 - embedding_parameters (dict, optional)
 - scale_aware (bool, optional, default=False)
 - child_structure_search (function, optional)

Example of obj function : $2ab+2bc+2ac-a-b-c+1$

```
sampler.py > ...
1  from dwave.system import DWaveSampler, EmbeddingComposite
2
3  Q = {(0,1): 2, (0,2): 2, (1,2): 2, (0,0): -1, (1,1): -1, (2,2): -1, (3,3): 1}
4  sampler = EmbeddingComposite(DWaveSampler())
5  response = sampler.sample_qubo(Q, num_reads=1000)
6
7  sample = response.first.sample
8  energy = response.first.energy
9  print("Sample:", sample)
10 print("Energy:", energy)
11
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Leap IDE /workspace/Quantum $ /usr/local/bin/python /workspace/Quantum/sampler.py
Sample: {0: 0, 1: 1, 2: 0, 3: 0}
Energy: -1.0
Leap IDE /workspace/Quantum $
```

Reference Link :

<https://colab.research.google.com/drive/1-Fx1RXFcHWQw4mlt3u4CmYKBwYiiQB0J?usp=sharing>

Simulated Annealing Sampler :

A simple simulated annealing sampler for testing and debugging code.

A dimod sampler that uses the simulated annealing algorithm.

Method : SimulatedAnnealingSampler.sample_qubo(Q,...) :

- This method is inherited from the Sampler base class.
- Converts the quadratic unconstrained binary optimization (QUBO) into a BinaryQuadraticModel and then calls sample().

Example of obj function : $2ab+2bc+2ac-a-b-c+1$

```
▶ from dwave.samplers import SimulatedAnnealingSampler

# Define the QUBO matrix
Q = {(0,1): 2, (0,2): 2, (1,2): 2, (0,0): -1, (1,1): -1, (2,2): -1, (3,3): 1}

|
# Create a SimulatedAnnealingSampler object
sampler = SimulatedAnnealingSampler()

# Solve the QUBO problem using simulated annealing
response = sampler.sample_qubo(Q, num_reads=1000)

# Print the best sample and its energy
best_sample = response.first.sample
best_energy = response.first.energy
print("Best sample:", best_sample)
print("Best energy:", best_energy)

Best sample: {0: 1, 1: 0, 2: 0, 3: 0}
Best energy: -1.0
```

Reference Link :

<https://colab.research.google.com/drive/1-Fx1RXFcHWQw4mlt3u4CmYKBwYiiQB0J?usp=sharing>

LeapHybridSolver() :

Leap's quantum-classical hybrid solvers are intended to solve arbitrary application problems formulated as quadratic models.

Problems submitted directly to quantum computers are in the binary quadratic model (BQM) format, unconstrained with binary-valued variables and structured for the topology of the quantum processing unit (QPU). Hybrid solvers may accept arbitrarily structured quadratic models (QM), constrained or unconstrained, with real, integer, and binary variables.

- These solvers, which implement state-of-the-art classical algorithms together with intelligent allocation of the quantum computer to parts of the problem where it benefits most, are designed to accommodate even very large problems.
- Leap's hybrid solvers enable you to benefit from D-Wave's deep investment in researching, developing, optimizing, and maintaining hybrid algorithms.

Syntax and Example :

```
from dwave.system import LeapHybridSampler
```

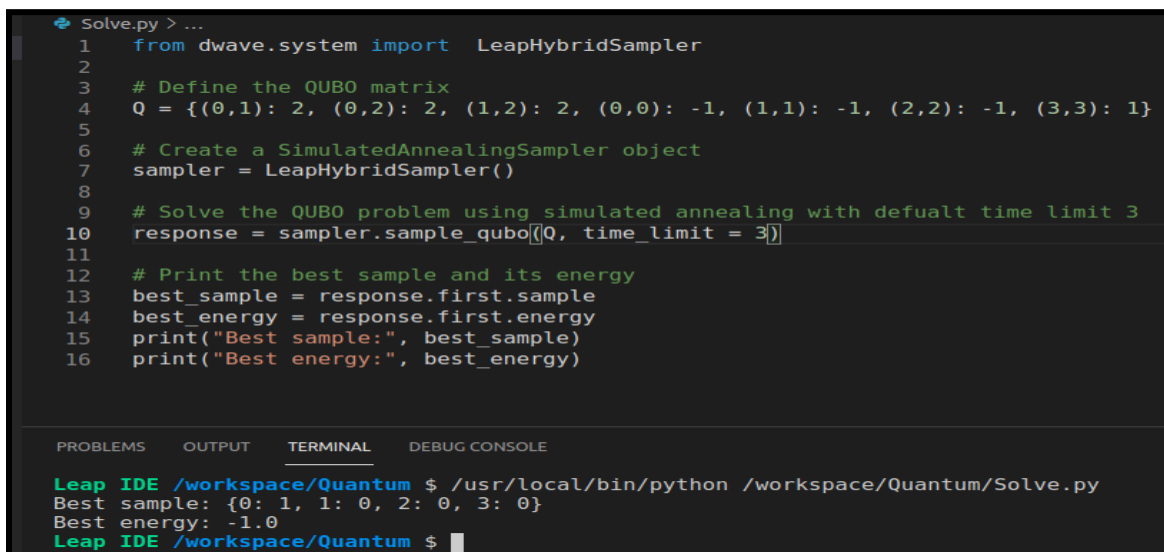
```
# Select a solver
```

```
sampler = LeapHybridSampler()
```

```
# Submit for solution
```

```
answer = sampler.sample_qubo(Q)
```

Example of obj function : $2ab+2bc+2ac-a-b-c+1$



```
Solve.py > ...
1  from dwave.system import LeapHybridSampler
2
3  # Define the QUBO matrix
4  Q = {(0,1): 2, (0,2): 2, (1,2): 2, (0,0): -1, (1,1): -1, (2,2): -1, (3,3): 1}
5
6  # Create a SimulatedAnnealingSampler object
7  sampler = LeapHybridSampler()
8
9  # Solve the QUBO problem using simulated annealing with default time limit 3
10 response = sampler.sample_qubo([Q, time_limit = 3])
11
12 # Print the best sample and its energy
13 best_sample = response.first.sample
14 best_energy = response.first.energy
15 print("Best sample:", best_sample)
16 print("Best energy:", best_energy)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Leap IDE /workspace/Quantum $ /usr/local/bin/python /workspace/Quantum/Solve.py
Best sample: {0: 1, 1: 0, 2: 0, 3: 0}
Best energy: -1.0
Leap IDE /workspace/Quantum $
```

Reference Link :

<https://colab.research.google.com/drive/1-Fx1RXFcHWQw4mlt3u4CmYKBwYiiQB0J?usp=sharing>