

1

SETTING UP

- 2 In this chapter we describe the environment, libraries and tools we use to
3 execute our tests.
4 In the following sections we install the SDKs to develop and interact with
5 quantum computers from IBM and D-Wave. We also present two other use-
6 ful tools to easily write optimization problems.

7 1.1 PYTHON ENVIRONMENT

- 8 The language used to interface with quantum computers is usually Python.
9 In this section we create a virtual environment in Python in order to commu-
10 nicate with the IBM quantum computer and the D-Wave quantum computer.
11 For our tests we manage Python environments with `conda`. Let's start by
12 creating the virtual environment named `quantum` and activating it with:

```
1 conda create --name quantum python=3.12 pip  
2 conda activate quantum
```

- 13 For our tests and to follow the various examples presented both by IBM and
14 D-Wave, it is also useful to be able to run a Jupyter notebook. We can install
15 Jupyter with:

```
1 pip install jupyter
```

16 1.2 IBM QISKIT

- 17 To program a gate-based architecture and to access IBM quantum computers
18 we use the *Qiskit* software stack. The name Qiskit is a general term referring
19 to a collection of softwares for executing programs on quantum computers.

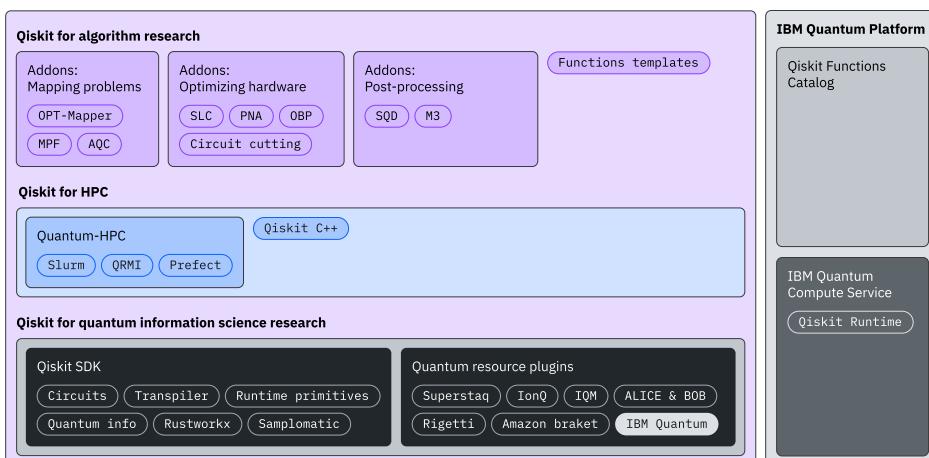


Figure 1.1: Qiskit software stack

20 The core components are *Qiskit SDK* and *Qiskit Runtime*. The first one is
 21 completely open source and allows the developer to define his circuit; the
 22 second one is a cloud-based service for executing quantum computations on
 23 IBM quantum computers.

24 **1.2.1 Hello World**

25 Following the IBM documentation¹ we can install the SDK and the Runtime
 26 with:

```
1 pip install qiskit matplotlib qiskit[visualization]
2 pip install qiskit-ibm-runtime
3 pip install qiskit-aer
```

27 Line 3 installs Aer, which is a high-performance simulator for quantum
 28 circuits written in Qiskit. Aer includes realistic noise models, and we will
 29 use it later to test our circuit.

30 Sometimes the Qiskit stack suffers from incompatibilities between the
 31 various software components that compose the environment. At the mo-
 32 ment of writing, the latest packages seem to work without any problem.
 33 For our tests we will use `qiskit: 2.2.3`, `qiskit-ibm-runtime: 0.43.1` and
 34 `qiskit-aer: 0.17.2`.

35 If the setup is successful we are now able to run a small test to build a Bell
 36 state (two entangled qubits). The following code assembles the gates, shows
 37 the final circuit and uses a sampler to simulate on the CPU the result of 1024
 38 runs of the program.

```
1 from qiskit import QuantumCircuit
2 from qiskit.primitives import StatevectorSampler
3
4 qc = QuantumCircuit(2)
5 qc.h(0)
6 qc.cx(0, 1)
7 qc.measure_all()
8
9 sampler = StatevectorSampler()
10 result = sampler.run([qc], shots=1024).result()
11 print(result[0].data.meas.get_counts())
12 qc.draw("mpl")
```

Listing 1: Building Bell state

39 **1.2.2 Transpilation**

40 Each Quantum Processing Unit (QPU) has a specific topology. We need to
 41 rewrite our quantum circuit in order to match the topology of the selected
 42 device on which we want to run our program. This phase of rewriting,
 43 followed by an optimization, is called transpilation.

¹ <https://quantum.cloud.ibm.com/docs/en/guides/install-qiskit>

44 Considering, for now, a fake hardware (so we do not need an API key)
 45 we can transpile the quantum circuit qc, from the code above, to match the
 46 topology of a specific QPU:

```

1  from qiskit_ibm_runtime.fake_provider import FakeWashingtonV2
2  from qiskit.transpiler import generate_preset_pass_manager
3
4  backend = FakeWashingtonV2()
5  pass_manager = generate_preset_pass_manager(backend=backend)
6
7  transpiled = pass_manager.run(qc)
8  transpiled.draw("mpl")

```

Listing 2: Transpilation

47 The following picture shows (1.2a) the quantum circuit that builds a Bell
 48 state, and (1.2b) the transpiled version where the Hadamard gate is replaced
 to match the actual topology of the QPU.

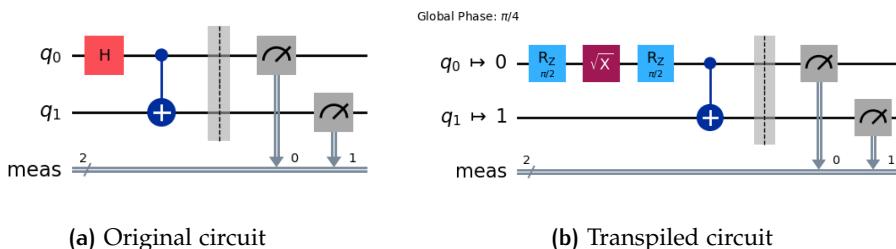


Figure 1.2: Transpilation example

49

50 1.2.3 Execution

51 To test our transpiled circuit we use Aer, which allows us to simulate also
 52 the noise of real quantum hardware. We can execute our program with:

```

1  from qiskit_aer.primitives import SamplerV2
2
3  sampler = SamplerV2.from_backend(backend)
4  job = sampler.run([transpiled], shots=1024)
5  result = job.result()
6  print(f"counts for Bell circuit : {result[0].data.meas.get_counts()}")

```

Listing 3: Simulated execution

53 If we look at the results of the execution we can observe that some answers
 54 present non-entangled qubits; this is caused by the (simulated) noise of the
 55 quantum device. A typical output of the execution could be:

```
> counts for Bell circuit : {'00': 504, '11': 503, '01': 10, '10': 7}
```

56 Where states 01 and 10 should not be present in an ideal execution with no
 57 errors.

58 1.2.4 A complete example on real hardware

59 1.3 D-WAVE OCEAN

60 To define an optimization problem that can be solved on a D-Wave quantum
 61 computer we use the Ocean software stack. Ocean also allows us to interact
 62 with D-Wave hardware, submit a problem, and simulate the execution on a
 classical CPU.

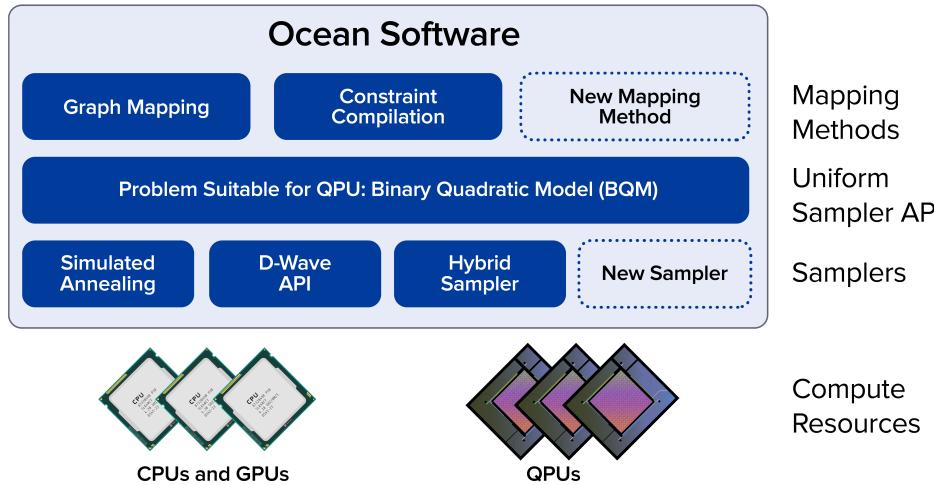


Figure 1.3: Ocean software stack

63
 64 All tools that implement the steps needed to solve your problem on a CPU,
 65 a D-Wave quantum computer, or a quantum-classical hybrid solver can be
 66 installed with:

```
1 pip install dwave-ocean-sdk
```

67 After the installation, running the command `dwave setup` will start an in-
 68 teractive prompt that guides us through a full setup. During the setup it
 69 is also possible to add an API token or connect to the D-Wave account to
 70 import a key directly to use the quantum hardware.

71 1.3.1 Hello World

72 To present a simple optimization program we consider the minimum vertex
 73 cover (MVC) problem. Given a graph $G = (V, E)$, the problem asks to find
 74 a subset $V' \subseteq V$ such that, for each edge $\{u, v\} \in E$, at least one of u or v
 75 belongs to V' , and the number of nodes in V' ($|V'|$) is the lowest possible.

76 The reduction from MVC to an Ising formulation is well known. The cost
 77 function that we want to minimize can be expressed by:

$$\text{cost} = \sum_{i=1}^{|V|} v_i + 2 \cdot \sum_{\{i,j\} \in E} (1 - v_i - v_j + v_i v_j)$$

78 where $v_i \in \{-1, 1\}$ and $v_i = 1$ means that $v_i \in V'$, otherwise $v_i = -1$.

79 Like all problems in Ising form we can express the cost as a symmetric
 80 matrix, so our function becomes

$$\text{cost} = v^T \times \mathbf{M} \times v$$

81 where v is the vector containing the binary variables v_i .

82 The figure shows an example graph (1.4a) and the corresponding matrix
 83 (1.4b) expressing the cost function.

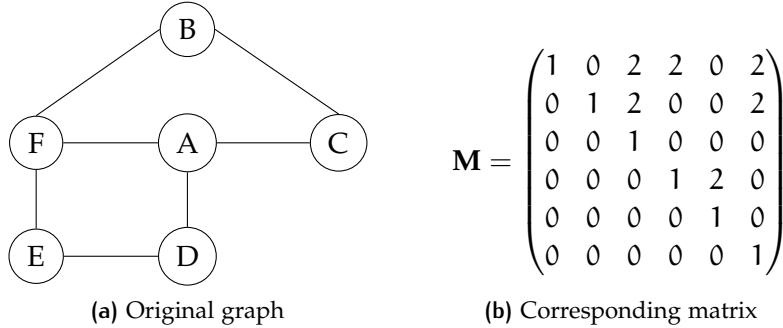


Figure 1.4: Ising formulation

84 The following code presents a possible implementation of the Ising model
 85 described above. We have defined two dictionaries to store the matrix coeffi-
 86 cients. The last line of code finds ten possible answers to the problem using
 87 the simulated annealing function implemented by D-Wave.

```
1 from dwave.samplers import SimulatedAnnealingSampler
2 linear = {'A': 1, 'B': 1, 'C': 1, 'D': 1, 'E': 1, 'F': 1}
3 quadratic = {('B', 'C'): 2, ('B', 'F'): 2, ('C', 'A'): 2, ('D',
4   ↵ 'A'): 2, ('E', 'D'): 2, ('E', 'F'): 2, ('F', 'A'): 2}
5 sampler = SimulatedAnnealingSampler()
6 result = sampler.sample_ising(linear, quadratic, num_reads=10)
```

Listing 4: Ising example

88 If we print the results with `print(result.aggregate())` we can observe
 89 something similar to this:

```
1 A B C D E F energy num_oc.
2 0 -1 -1 +1 +1 -1 +1 -14.0 6
3 1 +1 +1 -1 -1 +1 -1 -14.0 4
4 ['SPIN', 2 rows, 10 samples, 6 variables]
```

90 The two different results represent the two correct answers to our particular
 91 instance of the MVC problem.

92 1.3.2 Example on real hardware

93 1.3.3 Minor embedding

94 1.4 PYQUBO AND QUBOVERT

95 In listing 4 we have manually built the matrix representing the function that
 96 we want to minimize. It can be useful to have some tools that allow us to

97 work at a higher level, defining cost functions like ?? that we defined in the
 98 section about quantum annealing (??).

99 Considering again the MVC problem, the objective function tends to min-
 100 imize the number of nodes in our subset, while the penalty increases the
 101 cost if we leave out some edges. This interpretation allows us to transform
 102 the Ising model into the more familiar —from the point of view of a com-
 103 puter scientist— QUBO model, where all variables $x_i \in \{0, 1\}$. Let's see how
 104 PyQUBO and qubovert help us in this task.

105 **1.4.1 PyQUBO**

106 Reading from the documentation on the PyQUBO site², PyQUBO allows
 107 us to create QUBOs or Ising models from flexible mathematical expressions
 108 easily. Some of the features of PyQUBO are:

- 109 • Python based (C++ backend);
- 110 • Fully integrated with Ocean SDK;
- 111 • Automatic validation of constraints;
- 112 • Placeholder for parameter tuning.

113 We can install PyQUBO with `pip install pyqubo` and rewrite our MVC
 114 problem by defining the Hamiltonian that we want to minimize.

```

1 from pyqubo import Binary, Placeholder, Constraint
2 from dwave.samplers import SimulatedAnnealingSampler
3
4 A, B, C, D, E, F = Binary('A'), Binary('B'), Binary('C'), Binary('D'),
5   ↳ Binary('E'), Binary('F')
6
7 H_objective = (A + B + C + D + E + F)
8 H_penalty = Constraint(((1 - A - C + A*C) +
9   (1 - A - D + A*D) +
10  (1 - A - F + A*F) +
11  (1 - B - C + B*C) +
12  (1 - B - F + B*F) +
13  (1 - D - E + D*E) +
14  (1 - E - F + E*F)), label='cnstr0')
15
16 L = Placeholder('L')
17 H = H_objective + L*H_penalty
18 H_internal = H.compile()
19 bqm = H_internal.to_bqm(feed_dict={'L': 2})
20
21 sampler = SimulatedAnnealingSampler()
22 result = sampler.sample(bqm, num_reads=10)
```

Listing 5: Rewriting MVC with pyQUBO

² <https://pyqubo.readthedocs.io/en/latest/>

115 Listing 5 presents a possible re-implementation of listing 4, where we also
 116 see how PyQUBO interfaces with the Ocean SDK (line 17), and how to create
 117 (lines 14–16) and instantiate (line 17) a parametric Hamiltonian.

118 **1.4.2 qubovert**

119 As written in the documentation³, qubovert is the one-stop package for for-
 120 mulating, simulating, and solving problems in boolean and spin form. Using
 121 our nomenclature, boolean and spin form are respectively QUBO and Ising
 122 form.

123 Qubovert allows us to define various types of optimization problems that
 124 can be solved by brute force, with qubovert’s simulated annealing, or with
 125 D-Wave’s solver. Models defined in qubovert are:

126 **QUBO**: Quadratic Unconstrained Boolean Optimization;
 127 **QUSO**: Quadratic Unconstrained Spin Optimization (Ising model);
 128 **PUBO**: Polynomial Unconstrained Boolean Optimization;
 129 **PUSO**: Polynomial Unconstrained Spin Optimization;
 130 **PCBO**: Polynomial Constrained Boolean Optimization;
 131 **PCSO**: Polynomial Constrained Spin Optimization.

132 In addition to generic models, qubovert has a library of famous NP-complete
 133 problems mapped to QUBO and Ising forms.

```

1 from qubovert import boolean_var
2 from dwave.samplers import SimulatedAnnealingSampler
3
4 A, B, C, D, E, F = boolean_var('A'), boolean_var('B'),
5   ↪ boolean_var('C'), boolean_var('D'), boolean_var('E'),
6   ↪ boolean_var('F')
7
8 model = A + B + C + D + E + F
9 model.add_constraint_OR(A, C, lam=2)
10 model.add_constraint_OR(A, D, lam=2)
11 model.add_constraint_OR(A, F, lam=2)
12 model.add_constraint_OR(B, C, lam=2)
13 model.add_constraint_OR(B, F, lam=2)
14 model.add_constraint_OR(D, E, lam=2)
15 model.add_constraint_OR(E, F, lam=2)
16
17 qubo = model.to_qubo()
18 dwave_qubo = qubo.Q
19 sampler = SimulatedAnnealingSampler()
20 result = sampler.sample_qubo(dwave_qubo, num_reads=10)
```

Listing 6: Rewriting MVC with qubovert

3 <https://qubovert.readthedocs.io/en/latest/index.html>

134 Listing 6 shows a possible implementation of the MVC problem using the
135 tools provided by qubovert. Qubovert allows us to express our problem as a
136 PCBO; we use this formulation to express constraints in a more natural way.
137 In our example we ensure that each edge is covered simply by enforcing that
138 at least one of the nodes linked by the edge is present in the solution. This
139 constraint is repeated for each edge in the graph (lines 7–13). To specify the
140 Lagrange multiplier (equation ??) we use the keyword `lam`.

141 Qubovert, like PyQUBO, can interface with the Ocean SDK, transforming
142 a PCBO problem into a QUBO problem (line 15) and then rewriting it in the
143 format accepted by the D-Wave solver (or sampler).

144 **1.5 CONCLUSION**

145 In this chapter we have set up an environment to run our future experiments
146 and tests. We have also shown some small examples to present the main
147 characteristics and test the tools we will use in our work.

148 Following this setup allows anyone to recreate exactly the same configura-
149 tion we use, avoiding (for what we know and test) incompatibilities between
150 Python packages.