Quantum Computing
A Practical Perspective

Marco Venere

marco.venere@polimi.it

November 25th, 2024
Politecnico di Milano

Agenda

- November 7th → Theory Recap on Quantum Computing
- November 20th → Initial Setup and First Experiments
- November 21st → Grover's Algorithm
- November 25th → Combinatorial Optimization
- November 28th → VQE, QNN, QMC
- December 3rd → Quantum Error Correction & Mitigation – Projects Presentation

(it may be subject to variations)

Agenda

- November 7th → Theory Recap on Quantum Computing
- November 20th → Initial Setup and First Experiments
- November 21st → Grover's Algorithm
- November 25th → Combinatorial Optimization
- November 28th → VQE, QNN, QMC
- December 3rd → Quantum Error Correction & Mitigation – Projects Presentation

(it may be subject to variations)

Combinatorial Optimization

One of the most interesting use cases for quantum computing is Combinatorial Optimization.

This field regards solving mathematical problems, in order to optimize a number of possible key performance indicators:

- Time to perform some tasks (the less, the better)
- Amount of required resources (the less, the better)
- Amount of money a company can earn (the more, the better)

In general, we want to **maximize** or **minimize** a quantity

Mathematical Formalism

An optimization problem can be represented by the following formalism:

$$\underset{x}{\text{maximize}} \quad f(x) \qquad \textbf{objective function}$$

$$\text{subject to } g_i(x) \leq 0, i = 1, \ldots, m \qquad \textbf{inequality constraints}$$

$$h_j(x) = 0, j = 1, \ldots, p \qquad \textbf{equality constraints}$$

where:

$f(x)$: $\mathbb{R}^n \longrightarrow \mathbb{R}$ is the **objective function**,

$g_i(x)$ are **inequality constraints**, $m \geq 0$

$h_j(x)$ are **equality constraints**, $p \geq 0$

Noticeable Examples

There are several optimization problems that are well known for their industrial applications. Let's consider for example:

- Minimum Vertex Cover Problem
- Set Packing Problem
- Max 2-Sat Problem

Minimum Vertex Cover Problem

We have a graph with a set of vertices V and a set of edges E. We want to find a subset of vertices such that each edge in the graph is incident to at least one vertex in this subset. In particular, we want to find the subset with **minimum** number of vertices.

$$\underset{x}{\text{minimize}} \sum_{j \in V} x_j$$

$$\text{subject to } x_i + x_j \geq 1 \; \forall (i,j) \in E$$

x_j are Boolean variables, i.e., they can only be 0 or 1. They say whether node j belongs to the subset.

The meaning of this formulation is: take the minimum number of nodes, but keep in mind that for each edge in the graph, at least one of its vertices needs to be taken.

Set Packing Problem

We have a set of elements. We want to divide this set in different packages: every element will belong to a different package. Packages have weight: some are more important than others.

$$\underset{x}{\text{maximize}} \sum_{j=1}^{n} w_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \leq 1 \;\; for\; i = 1, \ldots, m$$

x_j are Boolean variables telling if subset j is taken, while a_{ij} is a Boolean coefficient saying if subsets i and j have elements in common. w_j is the weight of subset j.

The meaning of this formulation is: get the most important packages that you can, but keep in mind that packages sharing elements cannot be present in the same solution to the problem.

Max 2-Sat Problem

We have a set of Boolean clauses, i.e., expressions consisting of Boolean variables. We want to maximize the number of clauses that are concurrently evaluated to true. We only focus on clauses made of 2 variables.

$$\underset{x}{\text{minimize}} \sum_{j=1}^{n} f_j(\bar{x})$$

where $\bar{x}$ is the vector of the Boolean variables we want to consider (as many variables as we want), and $f_j(\bar{x})$ is the *penalty* associated with one of the clauses we want to consider. Since we are minimizing, the penalty value decreseas when the clause is satisfied. Doing so, to minimize the objective function, we need clauses evaluated to true. For example:

Clause		Quadratic Penalty
$x_1 \vee x_2$	$\rightarrow$	$(1 - x_1 - x_2 + x_1 x_2)$
$x_1 \vee \overline{x_3}$	$\rightarrow$	$(x_3 - x_1 x_3)$

QUBO Formalism

Quantum Computers exploit a specific formalism for combinatorial optimization problems: **quadratic unconstrained binary optimization** (QUBO).

In this formalism, the problem has a **quadratic** objective function, has **no constraints**, and all variables are **binary** (a.k.a. Boolean):

$$\max_x / \min \quad f(x)$$

$f(x)$ only contains linear or quadratic terms, e.g.: $f(x_1, x_2, x_3) = x_1 - 3x_1^2 + 2x_2 + + x_2^2 - 2x_1x_2$ (no other kinds of terms such as x_1^3 or $\log(x_2)$).

Notice the absence of constraints.

From Generic Formalism To QUBO Model

The three problems we have seen up to now were not in this form: they had constraints!

We can move to a QUBO formulation in the following way:

$$\max_x / \min_x \quad f(x)$$

$$\text{subject to } g_i(x) \leq 0, i = 1, \dots, m$$
$$h_j(x) = 0, j = 1, \dots, p$$

$$\longrightarrow$$

$$\max_x / \min_x \quad f(x) \mp P f_1(x)$$

$P f_1(x)$ is a penalty term: it encodes the constraints. If constraints are not respected, the penalty term decreases/increases the value of the objective function that we want to maximize/minimize, respectively.

Definition Of The Penalty

To define the penalty $Pf_1(x)$, we start from the constraints of the given problem and follow these rules:

Classical Constraint	Equivalent Penalty
$x + y \leq 1$	$P(xy)$
$x + y \geq 1$	$P(1 - x - y + xy)$
$x + y = 1$	$P(1 - x - y + 2xy)$
$x \leq y$	$P(x - xy)$
$x_1 + x_2 + x_3 \leq 1$	$P(x_1x_2 + x_1x_3 + x_2x_3)$
$x = y$	$P(xy)$

P is a constant that multiplies the penalty term. Its value depends on the problem and may be tuned in order to improve the performance of the algorithm.

Noticeable Penalty Examples

Minimum Vertex Cover Problem

$$\underset{x}{\text{minimize}} \sum_{j \in V} x_j$$

$$\text{subject to } x_i + x_j \geq 1 \; \forall (i,j) \in E$$

$$\underset{x}{\text{minimize}} \sum_{j \in V} x_j + P\left(\sum_{(i,j) \in E} \left(1 - x_i - x_j + x_i x_j\right)\right)$$

Set Packing Problem

$$\underset{x}{\text{maximize}} \sum_{j=1}^{n} w_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \leq 1 \; for \; i = 1, \ldots, m$$

$$\underset{x}{\text{maximize}} \sum_{j=1}^{n} w_j x_j - P \sum_{i,j:\; a_{i,j}=1} x_i x_j$$

Max 2-SAT Problem

$$\underset{x}{\text{minimize}} \sum_{j=1}^{n} f_j(\bar{x})$$

already as QUBO

$$\underset{x}{\text{minimize}} \sum_{j=1}^{n} f_j(\bar{x})$$

Final Q Matrix

Once we achieve the following QUBO formulation:

$$\max_x / \min \quad f(x) \mp P f_1(x)$$

we can manipulate the objective function to obtain the final QUBO formulation:

$$\max_x / \min \quad x^t C x \mp P(Ax - b)^t (Ax - b)$$

$$\max_x / \min \quad x^t C x \mp (x^t D x + c)$$

$$\max_x / \min \quad x^t Q x$$

where C, D, and Q are square matrixes, and x is a vector of binary variables.

Algorithms For Quantum Computers

After generating a QUBO model, we need to actually solve the optimization problem!

Quantum Computers provide us two different ways to solve it:

1. Quantum Approximate Optimization Algorithm (QAOA)

2. Quantum Annealing (QA)

They are both based on a specific formulation… the **Ising Model**

Ising Model

The Ising Model is a mathematical model used statistical mechanics to describe the Hamiltonian (energy) of a system. We will use it to encode our QUBO model onto the QPU.

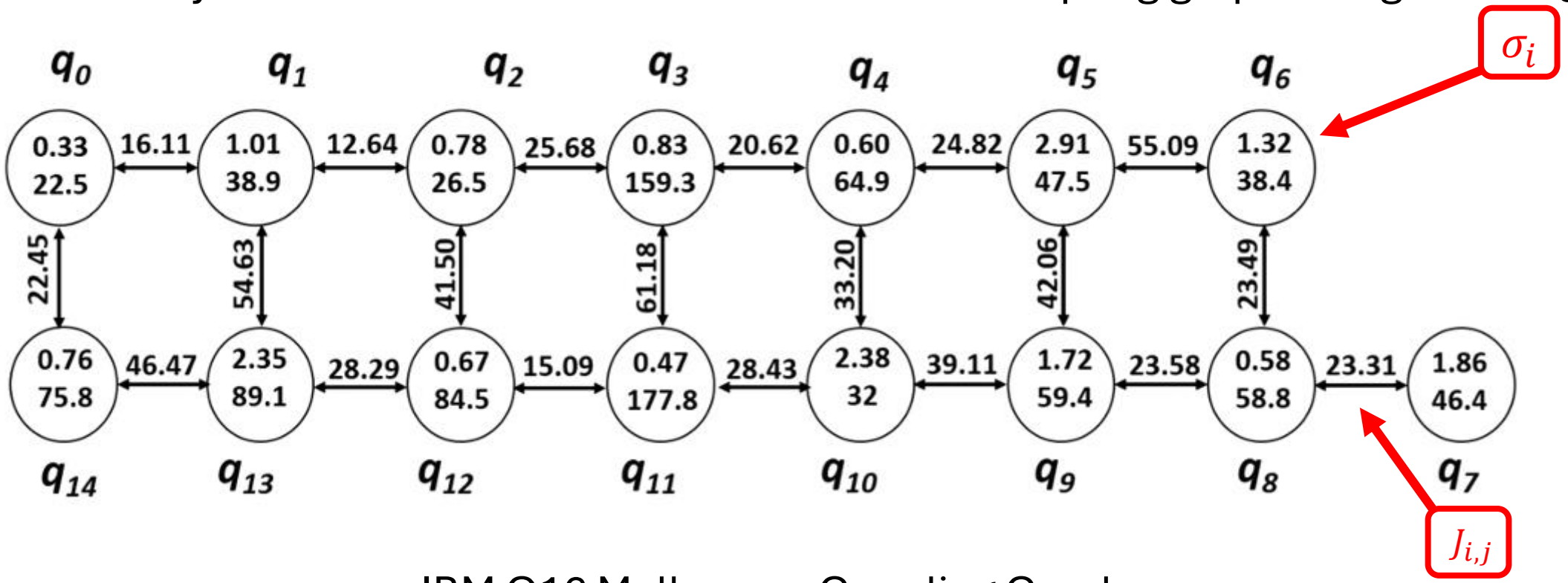
In its general form, it is defined as follows:

$$H(\sigma) = -\sum_{(i,j)\in\Lambda} J_{i,j}\sigma_i\sigma_j - \mu\sum_j h_j\sigma_j$$

where Λ is the graph describing our QPU, σ_i is the magnetic spin of qubit i (either +1 or -1), and $J_{i,j}$ is the ferromagnetic interaction between qubits i and j (that are interconnected in the coupling graph). Finally, μ is the magnetic moment.

Ising Model On The Coupling Graph

We can easily visualize such an Hamiltonian on the coupling graph of a generic QPU.



IBM Q16 Melbourne Coupling Graph

Nikita Acharya, Miroslav Urbanek, Wibe A. De Jong, and Samah Mohamed Saeed. 2021. Test Points for Online Monitoring of Quantum Circuits. J. Emerg. Technol. Comput. Syst. 18, 1, Article 14 (January 2022), 19 pages, https://doi.org/10.1145/3477928

From QUBO Model To Ising Model

Given a QUBO Model, we can easily derivate the corresponding coefficients for the Ising Model:

$$x^t Q x$$

QUBO Model

$$H(\sigma) = -\sum_{(i,j)\in\Lambda} J_{i,j}\sigma_i\sigma_j - \mu\sum_j h_j\sigma_j$$

Ising Model

$$\sigma_i = 2x_i - 1$$

$$x_i = \frac{1+\sigma_i}{2}$$

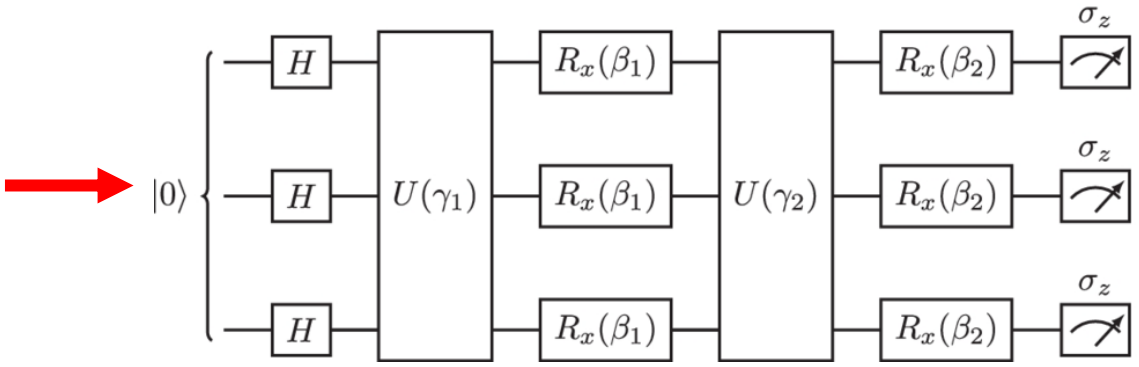
Conversion Rule

Quantum Approximate Optimization Algorithm

QAOA is a hybrid quantum-classical algorithm that solves combinatorial optimization problems.

$$\begin{aligned} \min_{x \in D} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \; i = 1, \ldots m \\ & h_j(x) = 0, \; j = 1, \ldots r \end{aligned}$$

optimization problem



ansatz: a parametric circuit

optimizer

Cobyla, Powell, Nelder-Mead, FakeNodes

The Ansatz Circuit

Starting from an optimization problem, QAOA defines a **parametric** quantum circuit, called **Ansatz**. Such a circuit represents your problem. We can create an Ansatz circuit starting from the Ising Model of the problem.

Given n qubits, initialized as $|0\rangle^{\otimes n}$, we describe the circuit as the following operator:

$$\mathcal{U}(\vec{\beta}, \vec{\gamma})|0\rangle^{\otimes n}$$

If the circuit parameters $\vec{\beta}, \vec{\gamma}$ are set up correctly, measuring qubits will give the solution to the minimization / maximization problem.

Therefore, the fundamental part is to **find the optimal parameters** of the circuit.

The Structure Of The Ansatz

Objective: devise the structure of $\mathcal{U}(\vec{\beta},\vec{\gamma})|0\rangle^{\otimes n}$

We can write $\mathcal{U}(\vec{\beta},\vec{\gamma})$ as $\mathcal{U}(\vec{\beta},\vec{\gamma}) = \mathcal{U}(\mathcal{H}_X\beta_p)\mathcal{U}(\mathcal{H}_C\gamma_p)\ldots\mathcal{U}(\mathcal{H}_X\beta_1)\mathcal{U}(\mathcal{H}_X\gamma_1)H^{\otimes n}$

Therefore, we iteratively apply $\mathcal{H}_X$ and $\mathcal{H}_C$, which have a fixed structure, but with different parameters.

We need to compute $\mathcal{U}(\mathcal{H}_X,\beta_j)$ and $\mathcal{U}(\mathcal{H}_C,\gamma_j)$as main step!

Mixing Hamiltonian

Regarding the first term, we have $\mathcal{U}(\mathcal{H}_X, \beta_j) = e^{-i\beta_j \mathcal{H}_X}$. $\mathcal{H}_X$ is called "mixing Hamiltonian", and it is defined as:

$$\mathcal{H}_X = \sum_{i=1}^{n} \sigma_i^x$$

where σ_i^x is the X operator applied to qubit i.

Therefore,

$$\mathcal{U}(\mathcal{H}_X, \beta_j) = e^{-i\beta_j \mathcal{H}_X} = \prod_{k}^{n} e^{-i\beta_j \sigma_k^x}$$

All the terms of the product simply become R_X operations of parameter $2\beta_j$. We know how to implement $\mathcal{U}(\mathcal{H}_X, \beta_j)$ as a circuit!

Cost Hamiltonian

Regarding, instead, the second term, we have $\mathcal{U}(\mathcal{H}_C, \gamma_j) = e^{-i\gamma_j \mathcal{H}_C}$. $\mathcal{H}_C$ is called "cost Hamiltonian.

Assume that your function is described by the following Ising Model:

$$f(x) = \sum_{i=1}^{n} h_i x_i + \sum_{i,j} J_{ij} x_i x_j$$

$\mathcal{H}_C$ is constructed as $\mathcal{H}_C = \sum_{i=1}^{n} h_i \sigma_i^z + \sum_{i,j} J_{ij} \sigma_i^z \sigma_j^z$.

where σ_i^z is the X operator applied to qubit i.

Therefore, $\mathcal{U}(\mathcal{H}_C, \gamma_j) = e^{-i\gamma_j \mathcal{H}_C} = e^{-i\gamma_j \left(\sum_{i=1}^{n} h_i \sigma_i^z + \sum_{i,j} J_{ij} \sigma_i^z \sigma_j^z\right)} = \prod_k^n e^{-i\gamma_j h_j \sigma_k^z} \prod_{k,l} e^{-i\gamma_j J_{kl} \sigma_k^z \sigma_l^z}$.

$e^{-i\gamma_j h_j \sigma_k^z}$ can be implemented as $R_Z(2\gamma_j h_j)$ and $e^{-i\gamma_j J_{kl} \sigma_k^z \sigma_l^z}$ as $R_{ZZ}(2\gamma_j J_{kl})$.

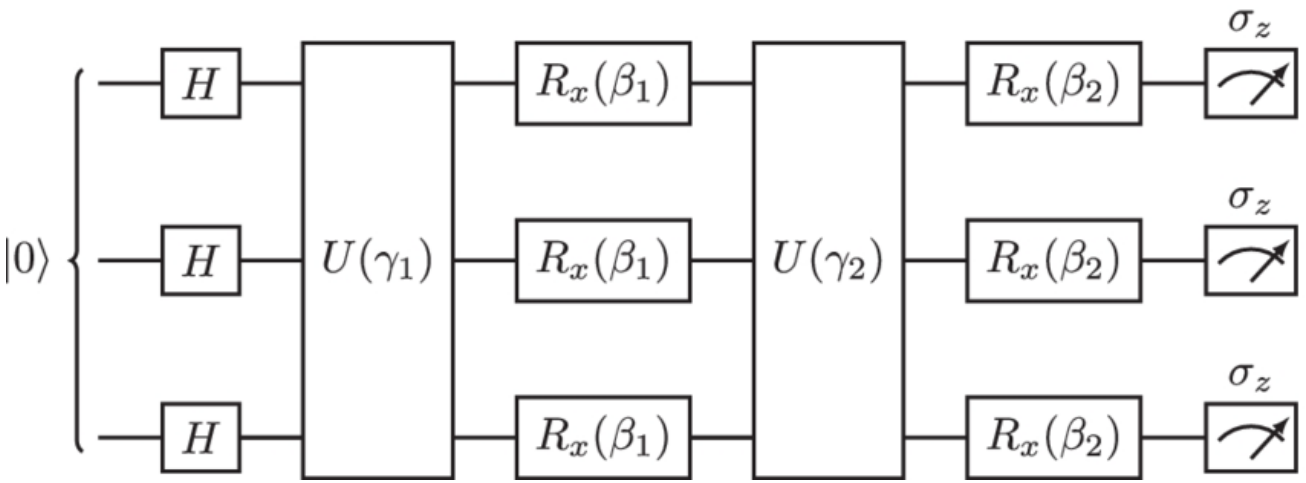
Overall Ansatz

Keep into account that $R_{ZZ}(\theta)$ is a quantum operator defined by the following matrix:

$$R_{ZZ}(\theta) = \begin{pmatrix} R_Z(\theta) & 0 \\ 0 & R_Z(\theta) \end{pmatrix}$$

At this point, we know the structure of the *ansatz*!

Indeed, we need to apply rotations around X and Z, with angles that are due to both algorithm parameters ($\vec{\beta}$ and $\vec{\gamma}$) and the structure of the problem (i.e., its Ising Model)!



The Solver

To find such parameters, we perform a classical search. A solver, running on a generic CPU, will explore the parameter space to find the best values.

At each optimization step:

1. the solver specific some parameter values
2. the corresponding quantum circuit is executed and qubits are measured
3. the value of the objective function is computed, to verify if the current solution found by the circuit is better or worse than the previous ones
4. the parameters are updated accordingly.

Parameters initialization, update rules, and conditions to terminate the exploration depend on the specific solver configuration.

Full Example

Let's see a full example to solve the **Number Partitioning Problem**.

Objective: given a set S of n numbers, we want to find two disjoint subsets S_1 and S_2, so that the difference between the sum of all numbers of each set is minimized.

For example: $S = \{3, 4, 5\} \rightarrow S_1 = \{3, 4\}$ and $S_2 = \{5\}$. Indeed, the difference between the sum of S_1 and S_2 is 2.

We want to solve the same problem by using QAOA!

Source: https://openqaoa.entropicalabs.com/what-is-the-qaoa/#example

Generating The Ansatz

We define a parametric quantum circuit which will represent our problem.

We need to make some design choices: how many parameters? What gates to use?

p tells us the size of the parameter vectors $\vec{\beta}$ and $\vec{\gamma}$ to use. In our case, let's pick $p = 1$.

The higher p is, the better the accuracy, but the higher the computation time required.

Generating The Ansatz

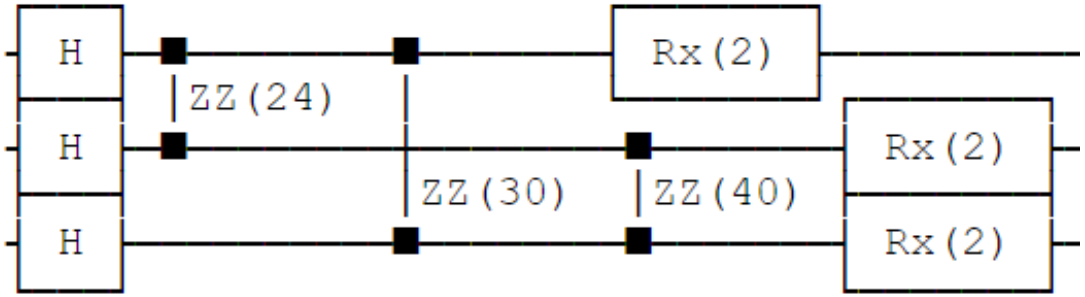
Using variables for each element of the set, the Ising Model is given by:

$$f(x) = 3 \cdot 4x_1x_2 + 3 \cdot 5\, x_1x_3 + 4 \cdot 5\, x_2x_3$$

Therefore, the cost Hamiltonian is given by:

$$\mathcal{H}_C = 3 \cdot 4\sigma_1^z\sigma_2^z + 3 \cdot 5\, \sigma_1^z\sigma_3^z + 4 \cdot 5\sigma_2^z\sigma_3^z$$

For example, in the case of initial values $\gamma_1 = 1$ and $\beta_1 = 1$, the *ansatz* is:



Optimization

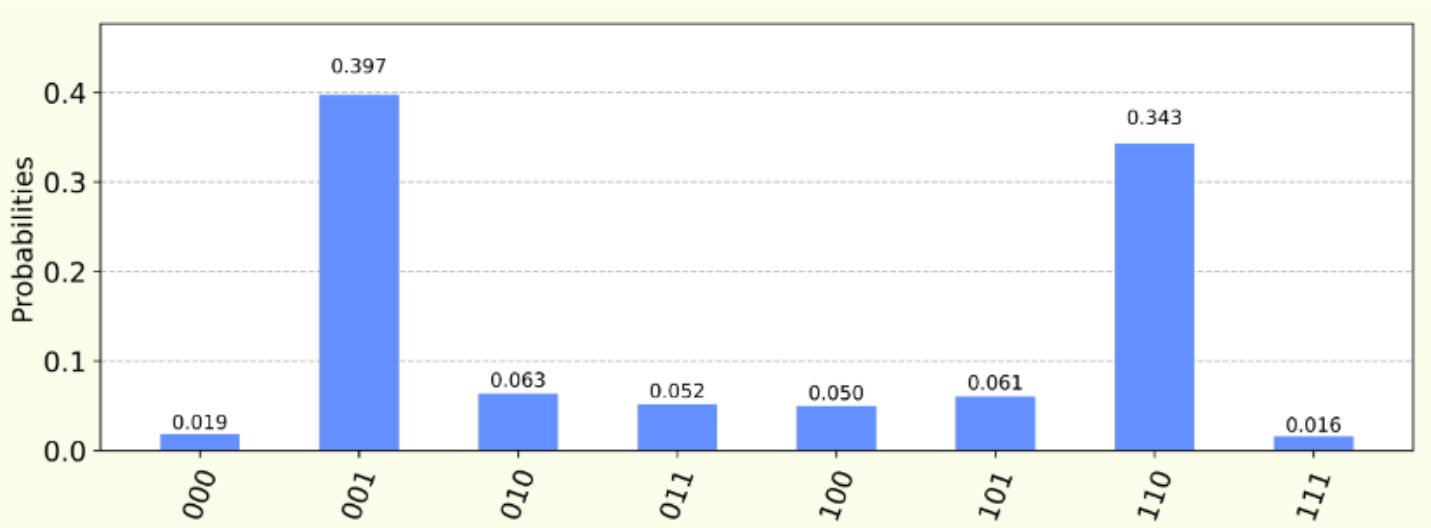
We now pick an optimizer and find for the optimal values.

For example, we can use Powell, a state-of-the-art optimizer. Other possibilities are Cobyla, Nelder-Mead, FakeNodes, etc.

Every optimizer has its own hyperparameters to tune to increase accuracy. They often have a maximum number of iterations to perform, some thresholds to decide whether to keep on with the search or not, and so far and so forth.

Results

After the overall optimization process, we can see the histogram of the probabilities of the possible solutions:



001 corresponds to $S_1 = \{3, 4\}$ and $S_2 = \{5\}$, while 110 corresponds to $S_1 = \{5\}$ and $S_2 = \{3,4\}$. Both solutions are correct, equivalent and found by QAOA!

Experiments With MATLAB

Let’s now use MATLAB to solve some QUBO problems!

First thing first, we need to understand:

1. how MATLAB API can be used to define a QUBO model

2. what underlying optimizer is used to perform the search

QUBO API

MATLAB allows for the following QUBO formulation:

$$f(x) = x^t Q x + c \cdot x + d$$

where we have the Q matrix from QUBO, plus a vector of coefficients and a constant number d.

In this way, we separate quadratic terms, linear terms, and constants.

We can create a QUBO problem with qubo(Q,c,d), and we can simply solve it with solve(problem).

No need to move to Ising Model! Everything happens transparently!

Tabu Search

The optimizer exploited by MATLAB is the Tabu Search. This is its pseudocode:

initialization →

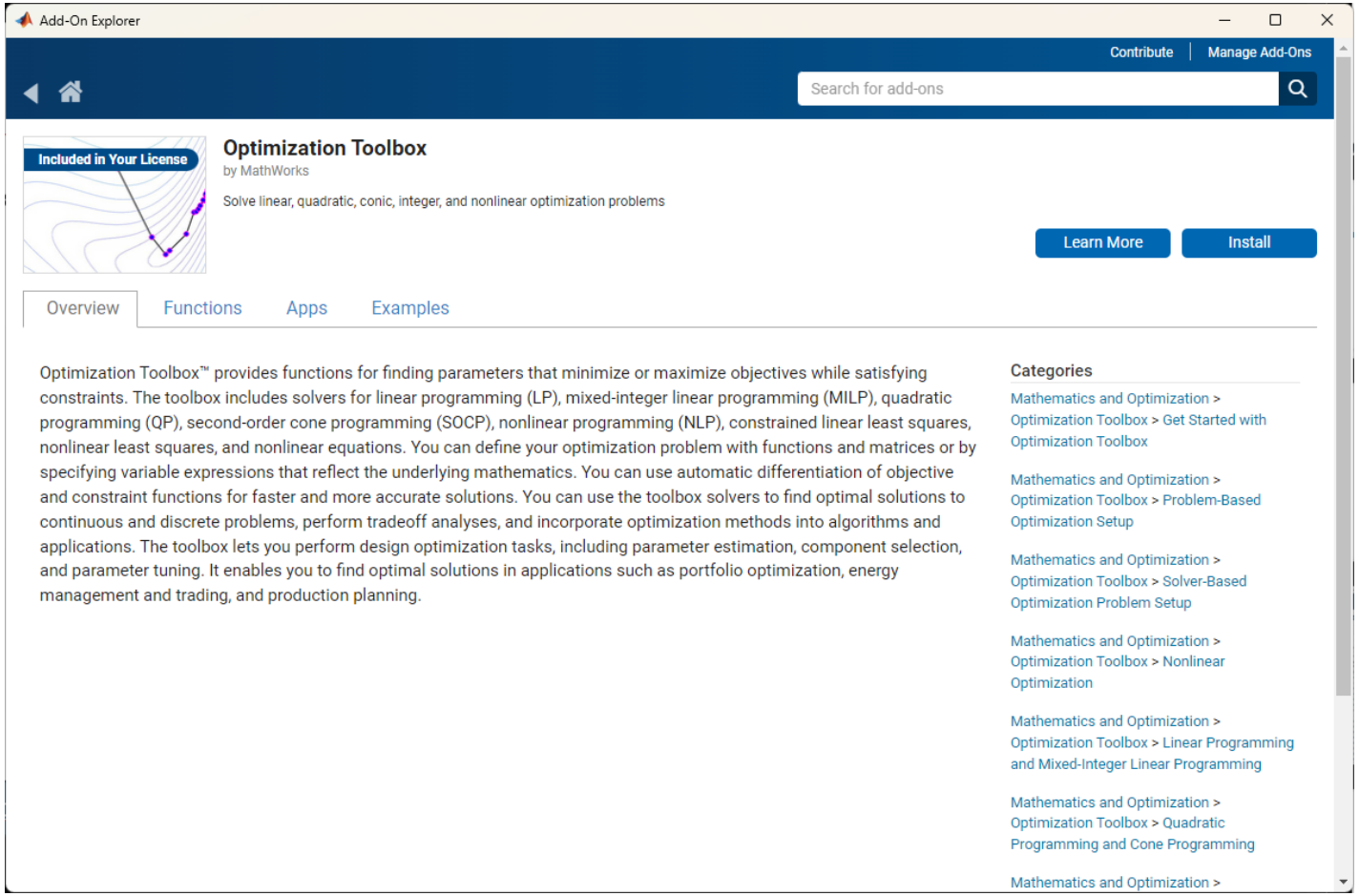
```
sBest ← s0
bestCandidate ← s0
tabuList ← []
tabuList.push(s0)
while (not stoppingCondition())
    sNeighborhood ← getNeighbors(bestCandidate)
    bestCandidateFitness ← -∞
    for (sCandidate in sNeighborhood)
        if ( (not tabuList.contains(sCandidate))
            and (fitness(sCandidate) > bestCandidateFitness) )
            bestCandidate ← sCandidate
            bestCandidateFitness ← fitness(bestCandidate)
        end
    end
    if (bestCandidateFitness is -∞)
        break;
    end
    if (bestCandidateFitness > fitness(sBest))
        sBest ← bestCandidate
    end
    tabuList.push(bestCandidate)
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return sBest
```

Search for local optima by using a fitness function (a.k.a. objective function)

In addition to this, MATLAB also adds some randomicity

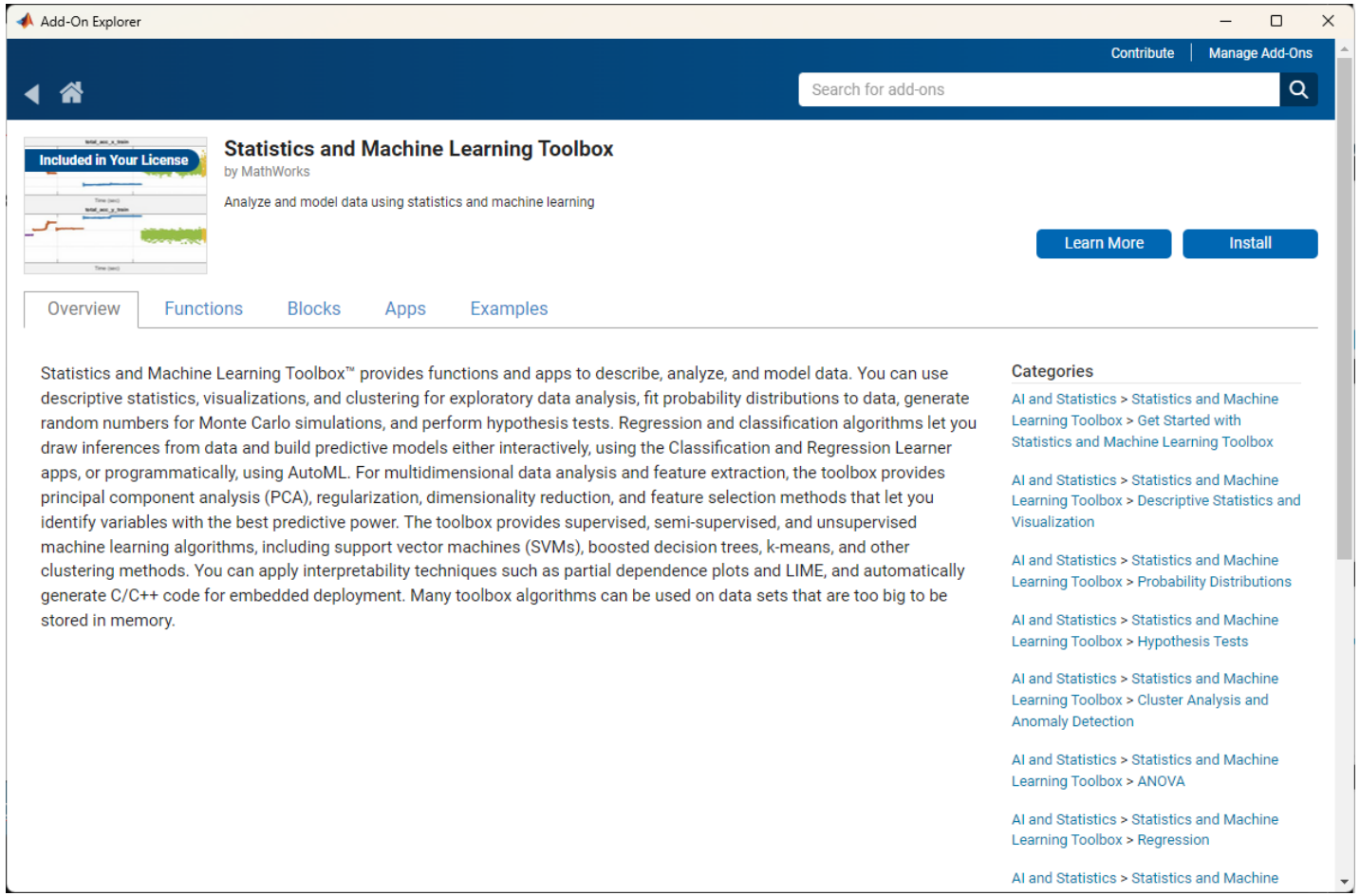
Additional MATLAB Requirements

To exploit some of the functions we will use, we need the Optimization Toolkit.



Additional MATLAB Requirements

To exploit some of the functions we will use, we also need the Stats and ML Toolkit.



Quantum Annealing

In addition to QAOA, another possible approach to solve combinatorial optimization problems is Quantum Annealing.

Quantum Annealers are specific kinds of QCs tailored to perform this operation. They are not general QCs: you cannot execute all the algorithms that are designed for generic quantum computation.

There are several possible typologies of quantum annealers, with varying number of qubits and degrees of connectivity (coupling graph). The underlying principle is always the same: we encode the problem in the Hamiltonian of the QPU, and we want to minimize it.

QA: Main Idea

Basic steps:

1. Qubits are initialized in a uniform superposition of all basis states. The energy of the system at this point is called *Initial Hamiltonian* H_0
2. The optimization problem is then encoded in a Hamiltonian H
3. The system slowly evolves from H_0 to H, according to the following law:

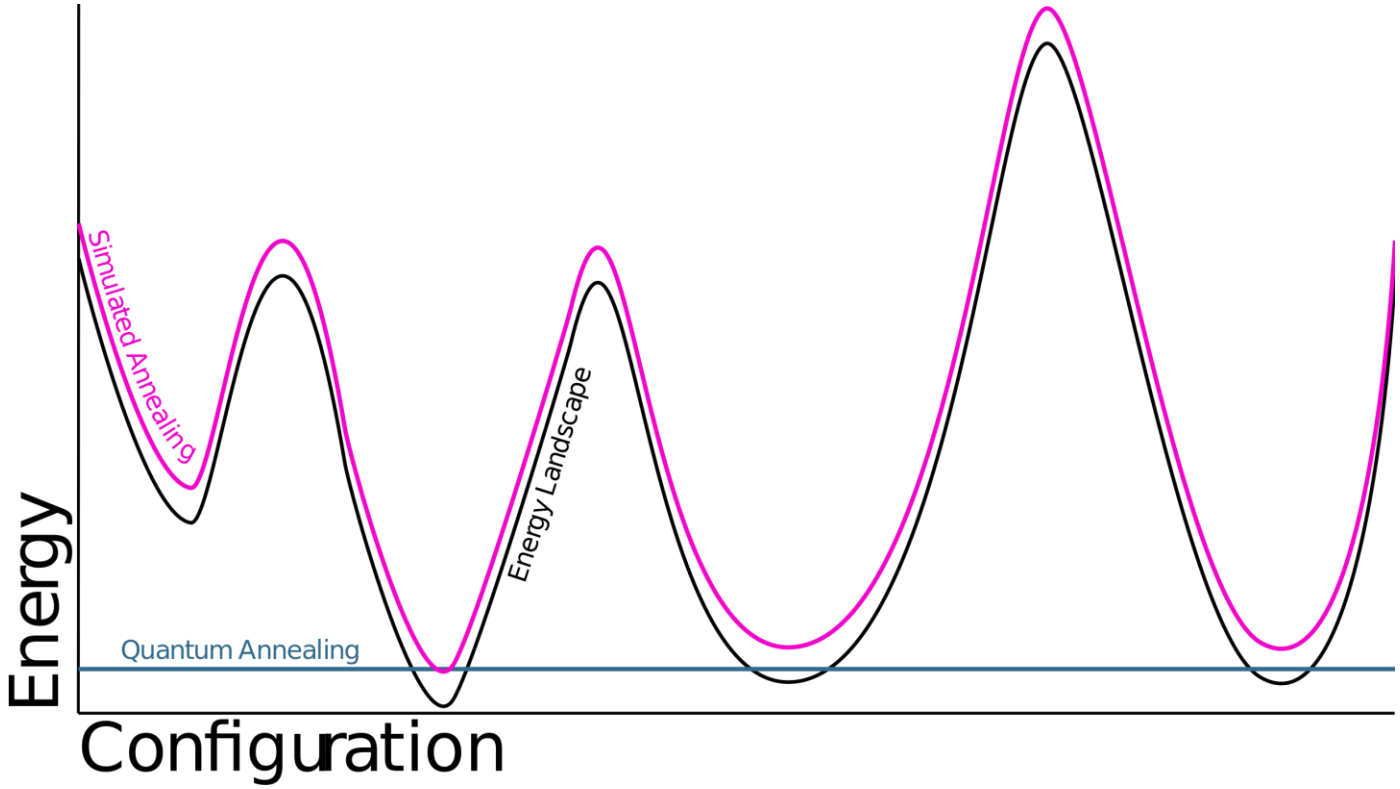
$$H(s) = A(s)H_0 + B(s)H, s = t/t_a$$

where t_a is the annealing time, and $A(s)$ is such that $A(s = 0) = 1, A(s = 1) = 0$, while $B(s)$ is such that $B(s = 0) = 0, B(s = 1) = 1$.

This follows the adiabatic quantum computation principle: if the system evolves sufficiently slow in time, then it remains in the ground state. Therefore, once the evolution terminates, the system minimizes the Hamiltonian H, thus producing the solution to the problem.

QA: Tunneling

Thanks to the tunneling principle, annealers can traverse the energy landscapes and find local minima, or even the global minimum.



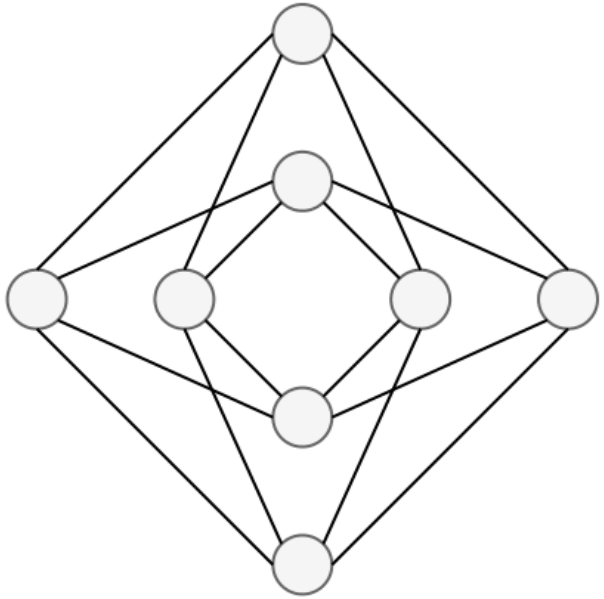
QA: General Workflow

General workflow:

1. Identify the optimization problem
2. Move problem to QUBO/Ising Model
3. Embed the problem onto the QPU
4. Read results

QA: Topology

Qubits are interconnected by couplers. The topology of the quantum annealer describes how qubits are interconnected.

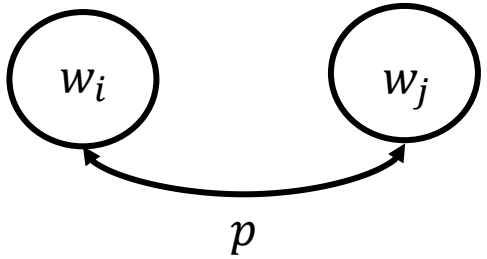


D-Wave 2000Q: topology

QA: Embedding

Once we have a QUBO/Ising Model, we need to encode the problem in the Hamiltonian of the QPU. This is performed by associating qubits and couplers with the weights of the model.

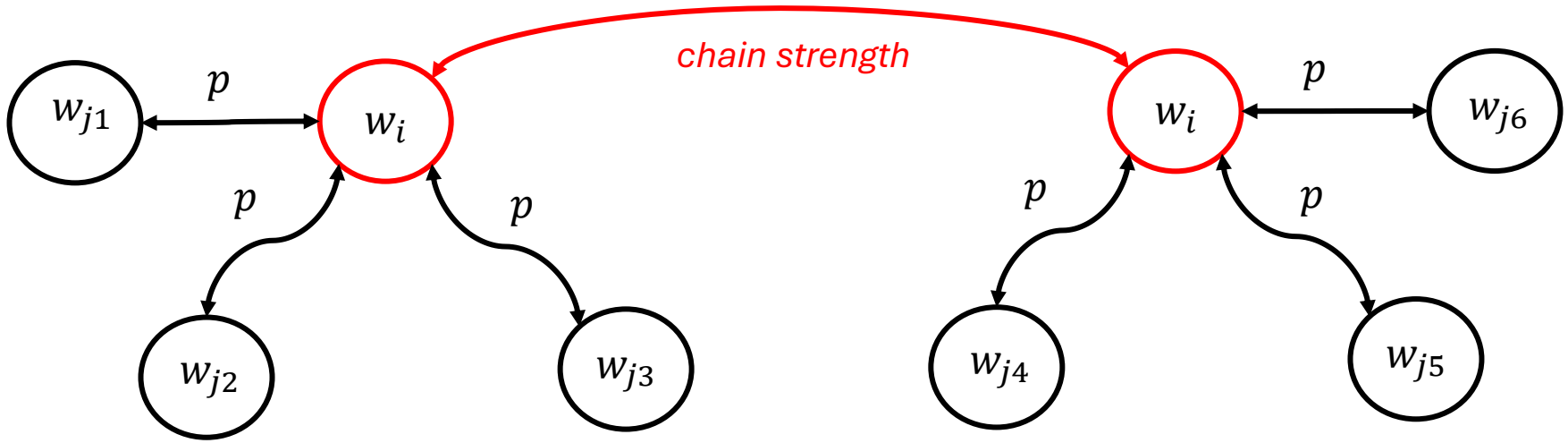
$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^{n} w_i x_i + \sum_{(i,j)} p x_i x_j$$



Unfortunately, this task is NP-hard and requires heuristics.

QA: Embedding

If constraints involve more qubits than connectivity degree of the topology of the quantum annealers, chains are created



QA: State Of The Art

Nowadays, the two most used quantum annealers are from D-Wave:

1. D-Wave 2000Q: 2000+ qubits and 6000+ couplers

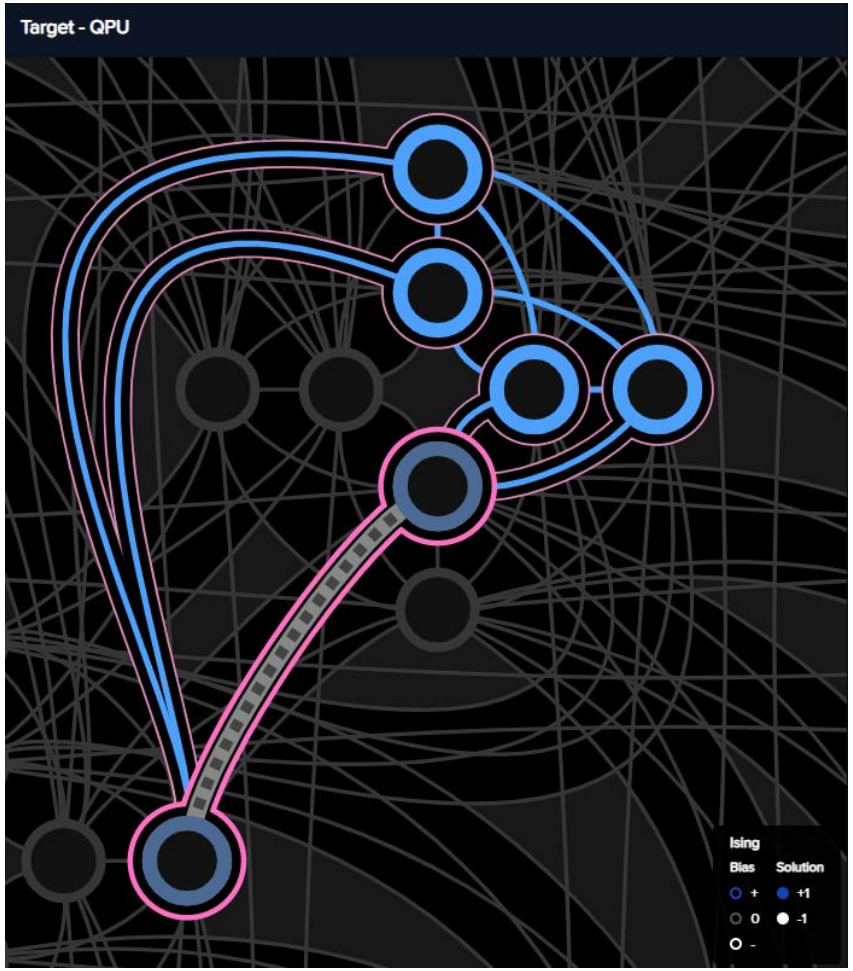
2. D-Wave Advantage: 5000+ qubits and 35000+ couplers

QA: D-Wave Dashboard

D-Wave provides a GitHub Codespaces IDE to develop software for quantum annealers.

A specific Python API is used to define variables and constraints, perform the embedding, execute the workflow on the QPU, and retrieve results.

The dashboard also allows for watching the QPU embedding results.



https://docs.ocean.dwavesys.com/en/stable/concepts/embedding.html

QA: A Practical Example

Let’s now see some code to solve the Set Packing Problem on QAs.

```
11  class SetPackingProblem:
12      """This class represents a set-packing problem,
13      in which set elements must be inserted into n different subsets,
14      such that each element is inserted in at most one subset.
15      If differents weights are associated to each subset, the goal is to achieve
16      a subsets selection of maximum weight. """
17      def __init__(self, subsets, weights, constraints):
18          """This is the constructor of the SetPackingProblem class.
19          Params:
20          sets_number: number of possible subsets
21          weights: weight associated to each different subset.
22              Defaults weights are unitary for each subset.
23          constraints: a list of sets_number constraints."""
24          self.s = subsets
25          if weights == None:
26              weights = [1] * len(subsets)
27          self.w = weights
28          self.c = constraints
```

https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example

Let’s now see some code to solve the Set Packing Problem on QAs.

```
30      def prepare(self):
31          """This method solves this problem by using the BQM API of D-Wave Leap.
32              Returns a dictionary in which subsets are the key of the dictionary, and their
33              corresponding value is 1 if that subset is selected, 0 otherwise."""
34          self.bqm = BinaryQuadraticModel({}, {}, 0, 'BINARY')
35          for i in range(len(self.s)):
36              self.bqm.offset += 1
37              self.bqm.add_variable(self.s[i], 0-(self.w[i])) #add variable for subset
38          for cons in self.c:
39              for i in range(len(cons)):
40                  for j in range(i):
41                      self.bqm.add_interaction(cons[i], cons[j], 6) #add constraint to avoid two non-disjoint subsets being selected
42          return self
```

https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example

Let’s now see some code to solve the Set Packing Problem on QAs.

```
52      def sample_composite(self, show_inspector = False, pre_factor = 2.0, num_of_reads = 100):
53          """
54              This method does the sampling of this problem, by using the 2000Q platform and DWaveSampler.
55              It must be called after the prepare() method.
56              Params:
57                  show_inspector: boolean, if set to True the inspector screen is shown;
58                  pre_factor: a parameter required by the function used to set the chain strength for the sampling;
59                  num_of_reads: number of samples asked to the solver.
60                  """
61          sampler = EmbeddingComposite(DWaveSampler(solver={'topology__type': 'chimera'}))
62          chain_strength = uniform_torque_compensation(self.bqm, sampler, pre_factor)
63          sampleset = sampler.sample(self.bqm, chain_strength, label="Set Packing", num_reads = num_of_reads)
64          if show_inspector:
65              show(sampleset)
66          return sampleset
```

https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example

Let’s now see some code to solve the Set Packing Problem on QAs.

```
68      def sample_advantage(self, show_inspector = False, pre_factor = 2.0, num_of_reads = 100):
69          """
70              This method does the sampling of this problem, by using the Advantage platform and DWaveSampler.
71              It must be called after the prepare() method.
72              Params:
73                  show_inspector: boolean, if set to True the inspector screen is shown;
74                  pre_factor: a parameter required by the function used to set the chain strength for the sampling;
75                  num_of_reads: number of samples asked to the solver."""
76          sampler = EmbeddingComposite(DWaveSampler())
77          chain_strength = uniform_torque_compensation(self.bqm, sampler, pre_factor)
78          sampleset = sampler.sample(self.bqm, chain_strength, label="Set Packing", num_reads = num_of_reads)
79          if show_inspector:
80              show(sampleset)
81          return sampleset
```

https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example

Let’s now see some code to solve the Set Packing Problem on QAs.

```
44      def sample_hybrid(self):
45          """
46              This method does the sampling of this problem, by using the LeapHybridSampler.
47              It must be called after the prepare() method."""
48          sampler = LeapHybridSampler(solver={'category': 'hybrid'})
49          sampleset = sampler.sample(self.bqm, label="Set Packing")
50          return sampleset
```

https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example

# Variables	#Qubits		Chain Length	
-	2000Q	Advantage	2000Q	Advantage
10	25	15	3	2
30	269	109	12	5
60	1151	427	28	12
90	-	920	-	13
120	-	1804	-	22
150	-	3309	-	34

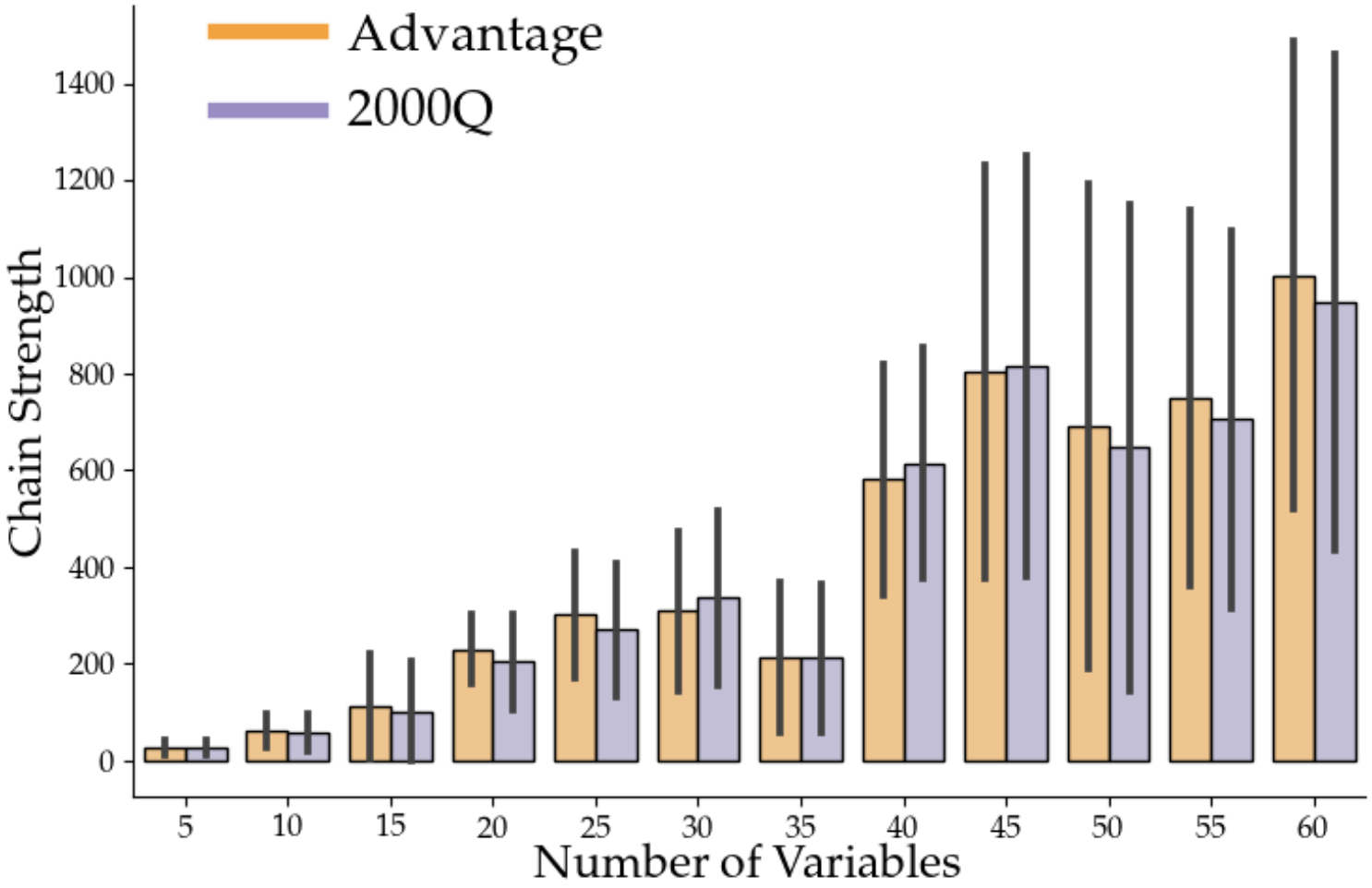
https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example

# Variables	#Qubits		Chain Length	
-	2000Q	Advantage	2000Q	Advantage
10	25	15	3	2
30	269	109	12	5
60	1151	427	28	12
90	-	920	-	13
120	-	1804	-	22
150	-	3309	-	34

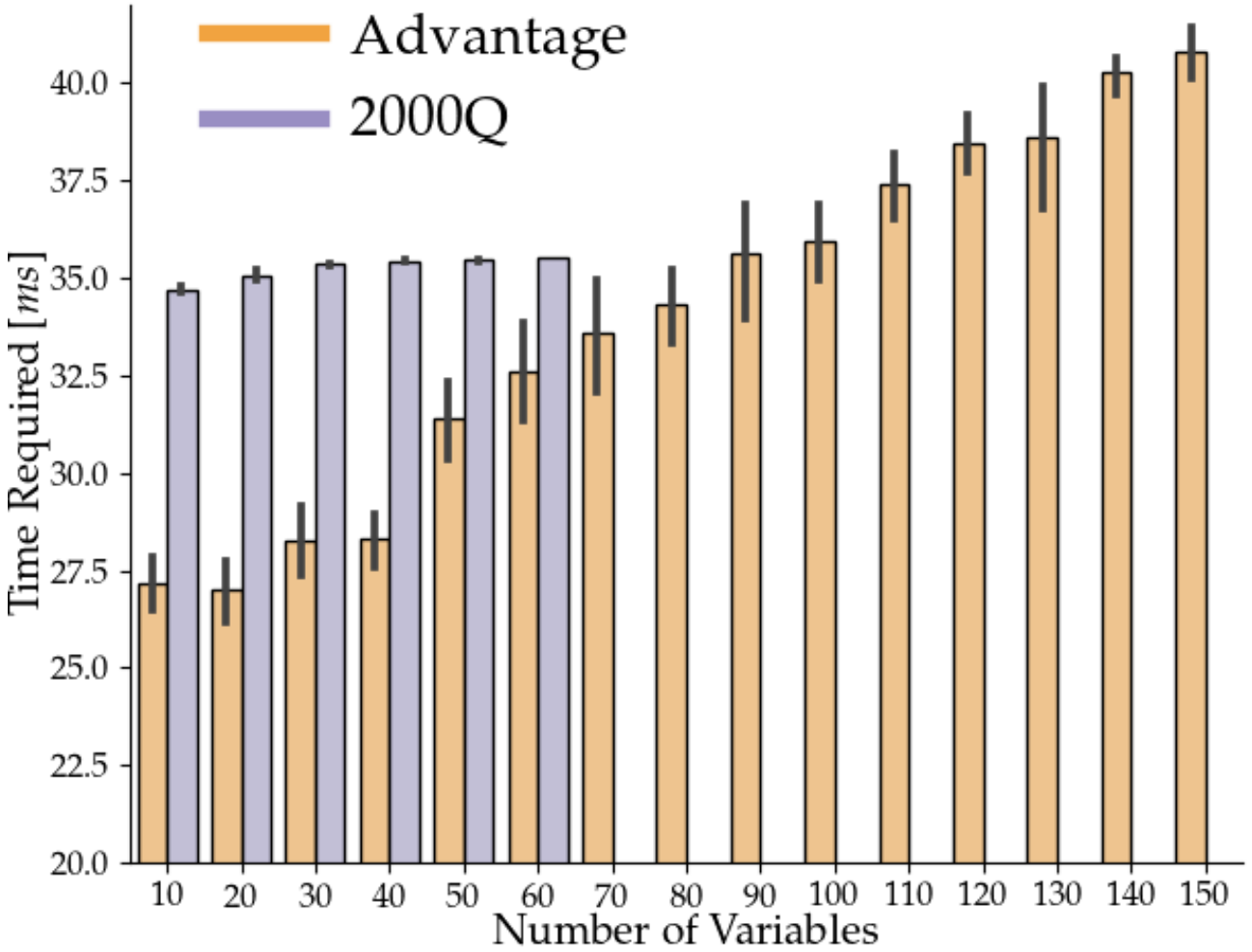
https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example



https://github.com/necst/quantum_annealer_benchmarking

QA: A Practical Example



https://github.com/necst/quantum_annealer_benchmarking

Thank you for your attention!

Quantum Computing
A Practical Perspective

Marco Venere

marco.venere@polimi.it

November 25th, 2024
Politecnico di Milano