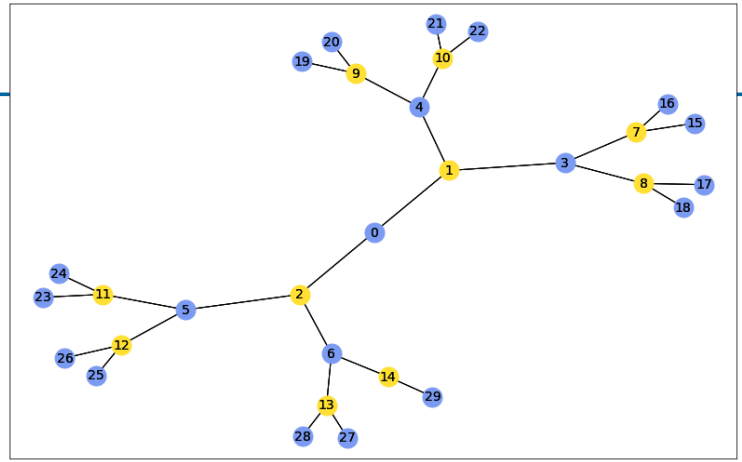
Atos QLM Fast-Start

Introduction to Simulated Quantum Annealing

2021/12/03

Gaëtan Rubez
Quantum Computing Expert
gaetan.rubez@atos.net

Combinatorial optimization



Usual specification of a *combinatorial problem:*

$$C(z) = \sum \alpha_i C_i(z) \qquad argmin_z\, C(z)$$

- z is a bitstring
- C is the *target function* (or *cost function*)
- C_i are called *clauses* and are usually $\{\mathbf{0}, \mathbf{1}\}$ **valued** and **local**
- α_i are called *weights*

These problems usually correspond to many yes/no decisions giving a value to the cost function we wish to minimize or maximize.

Combinatorial problems

In the QLM

In the QLM these cost functions are described via **CombinatorialProblem**s

```
my_problem = CombinatorialProblem()
variables = my_problem.new_vars(10) # declare an array of 10 bool vars.
for i in range(9):
    my_problem.add_clause(variables[i] | variables[i+1])
for clause in my_problem.clauses:
    print(clause)
```

```
V(0) | V(1)
V(1) | V(2)
V(2) | V(3)
V(3) | V(4)
V(4) | V(5)
V(5) | V(6)
V(6) | V(7)
V(7) | V(8)
V(8) | V(9)
```

In practice, cost functions can be restricted to a simple subclass: QUBO

Quadratic Unconstrained Binary Optimization
QUBO

First specification of QUBO problem*:*

$$argmin_x \, \mathrm{E}(x) \quad ; \quad \mathrm{E}(x) = x^T Q x \; = \sum_{i<j}^{n} Q_{ij} x_i x_j + \sum_{i}^{n} Q_{ii} x_i ; \qquad x = (x_1, \ldots, x_n)^T$$

- x is a bitstring ($x_i \in \{0; 1\}$)
- E is the *quadratic polynomial* (or *cost function*)
- Q is a real symmetric matrix

Quadratic Unconstrained Binary Optimization
Example

Minimize $E(x_1, x_2, x_3, x_4) = -4\,x_1 - 2x_2 - 5x_3 - 3x_4 + 4x_1x_2 + 6x_1x_3 + 2x_2x_3 + 10x_3x_4$
with $x_i \in \{0; 1\}$

- E is a quadratic function with **a linear part** $-4\,x_1 - 2x_2 - 5x_3 - 3x_4$ and **a quadratic part** $4x_1x_2 + 6x_1x_3 + 2x_2x_3 + 10x_3x_4$
- x_i are **binary variables** so $x_i = x_i^2$ implying that the linear part can be written:

$$-4x_1^2 - 2x_2^2 - 5x_3^2 - 3x_4^2$$

- We can write the problem with a matrix:

$$\textbf{Minimize } E(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4) \begin{pmatrix} -4 & 2 & 3 & 0 \\ 2 & -2 & 1 & 0 \\ 3 & 1 & -5 & 5 \\ 0 & 0 & 5 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Quadratic Unconstrained Binary Optimization
Example

Minimize $E(x_1, x_2, x_3, x_4) = -4\,x_1 - 2x_2 - 5x_3 - 3x_4 + 4x_1x_2 + 6x_1x_3 + 2x_2x_3 + 10x_3x_4$
with $x_i \in \{0; 1\}$

- E is a quadratic function with **a linear part** $-4\,x_1 - 2x_2 - 5x_3 - 3x_4$ and **a quadratic part** $4x_1x_2 + 6x_1x_3 + 2x_2x_3 + 10x_3x_4$
- x_i are **binary variables** so $x_i = x_i^2$ implying that the linear part can be written:

$$-4x_1^2 - 2x_2^2 - 5x_3^2 - 3x_4^2$$

- We can write the problem with a matrix:

$$\textbf{Minimize } E(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4) \begin{pmatrix} -4 & 2 & 3 & 0 \\ 2 & -2 & 1 & 0 \\ 3 & 1 & -5 & 5 \\ 0 & 0 & 5 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

- Minimize $E(x) = x^T Q x$

Quadratic Unconstrained Binary Optimization
Example

Minimize $E(x_1, x_2, x_3, x_4) = -4\ x_1 - 2x_2 - 5x_3 - 3x_4 + 4x_1x_2 + 6x_1x_3 + 2x_2x_3 + 10x_3x_4$
with $x_i \in \{0; 1\}$

- E is a quadratic function with **a linear part** $-4\ x_1 - 2x_2 - 5x_3 - 3x_4$ and **a quadratic part** $4x_1x_2 + 6x_1x_3 + 2x_2x_3 + 10x_3x_4$
- x_i are **binary variables** so $x_i = x_i^2$ implying that the linear part can be written:

$$-4x_1^2 - 2x_2^2 - 5x_3^2 - 3x_4^2$$

- We can write the problem with a matrix:

$$\textbf{Minimize } E(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4) \begin{pmatrix} -4 & 2 & 3 & 0 \\ 2 & -2 & 1 & 0 \\ 3 & 1 & -5 & 5 \\ 0 & 0 & 5 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

- Minimize $E(x) = x^T Q x$. The solution is $E(1, 0, 0, 1) = -7$

Quadratic Unconstrained Binary Optimization
Max-cut problem

- Given an undirected **graph G(V, E)**
  - **V** is a set of **vertices**
  - **E** is a set of **edges**

- The objective of the **Max-cut problem** is to find two sets from V that **maximize** the number of edges between these two sets.

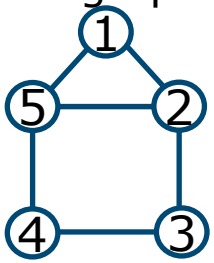
- We can use **binary variables x_j** to tell if a vertex is in a set or the other.

$$\text{Maximize } \mathrm{E}(x) = \sum_{i,j}(xi + xj - 2xixj)$$

$$\text{Maximize } \mathrm{E}(x) = x^T Q x$$

Quadratic Unconstrained Binary Optimization
Max-cut example

- Let consider the following undirected graph with 5 vertices and 6 edges:



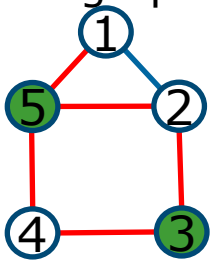
- Formulation :

$$\text{Maximize } E(x) = (x_1 + x_2 - 2x_1x_2) + (x_1 + x_5 - 2x_1x_5) + (x_2 + x_3 - 2x_2x_3)$$
$$+ (x_2 + x_5 - 2x_2x_5) + (x_3 + x_4 - 2x_3x_4) + (x_4 + x_5 - 2x_4x_5)$$
$$= 2x_1 + 3x_2 + 2x_3 + 2x_4 + 3x_5 - 2x_1x_2 - 2x_1x_5 - 2x_2x_3 - 2x_2x_5 - 2x_3x_4 - 2x_4x_5$$

$$\text{Maximize } E(x) = x^T Q x, \text{ with } Q = \begin{pmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & -1 & 0 & -1 & 3 \end{pmatrix}$$

Quadratic Unconstrained Binary Optimization
Max-cut example

- Let consider the following undirected graph with 5 vertices and 6 edges:



- Formulation :

$$\text{Maximize } E(x) = (x_1 + x_2 - 2x_1x_2) + (x_1 + x_5 - 2x_1x_5) + (x_2 + x_3 - 2x_2x_3)$$
$$+ (x_2 + x_5 - 2x_2x_5) + (x_3 + x_4 - 2x_3x_4) + (x_4 + x_5 - 2x_4x_5)$$
$$= 2x_1 + 3x_2 + 2x_3 + 2x_4 + 3x_5 - 2x_1x_2 - 2x_1x_5 - 2x_2x_3 - 2x_2x_5 - 2x_3x_4 - 2x_4x_5$$

$$\text{Maximize } E(x) = x^T Q x, \text{ with } Q = \begin{pmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & -1 & 0 & -1 & 3 \end{pmatrix}$$

- One possible solution : E(0,0,1,0,1) = 5

Quadratic Unconstraint Binary Optimization
Classical Ising model

- The QUBO model is equivalent to Ising model:

$$H(s_1,\dots,s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_i h_i s_i$$

With N spins $s_i \in \{-1; 1\}$
J_{ij} and h_i are real numbers

- The equivalence is done by a change of variable : $x_j = \frac{s_i+1}{2}$

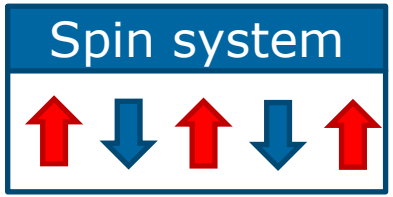
- The formulation of many NP problems has already be done.

arxiv.org:1302.5843

Quadratic Unconstraint Binary Optimization
Classical Ising model

- The QUBO model is equivalent to Ising model:

$$H(s_1,\dots,s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_i h_i s_i$$



With N spins $s_i \in \{-1; 1\}$

J_{ij} and h_i are real numbers

- The equivalence is done by a change of variable : $x_j = \frac{s_i+1}{2}$

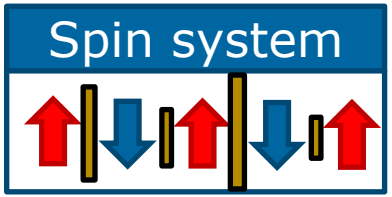
- The formulation of many NP problems has already be done.

arxiv.org:1302.5843

Quadratic Unconstraint Binary Optimization
Classical Ising model

- The QUBO model is equivalent to Ising model:

$$H(s_1,\dots,s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_i h_i s_i$$



With N spins $s_i \in \{-1; 1\}$

J_{ij} and h_i are real numbers

- The equivalence is done by a change of variable : $x_j = \frac{s_i+1}{2}$

- The formulation of many NP problems has already be done.

arxiv.org:1302.5843

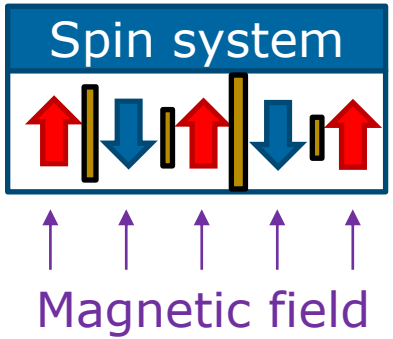
Quadratic Unconstraint Binary Optimization
Classical Ising model

- The QUBO model is equivalent to Ising model:

$$H(s_1,\dots,s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_i h_i s_i$$

With N spins $s_i \in \{-1; 1\}$

J_{ij} and h_i are real numbers



- The equivalence is done by a change of variable : $x_j = \frac{s_i+1}{2}$

- The formulation of many NP problems has already be done.

arxiv.org:1302.5843

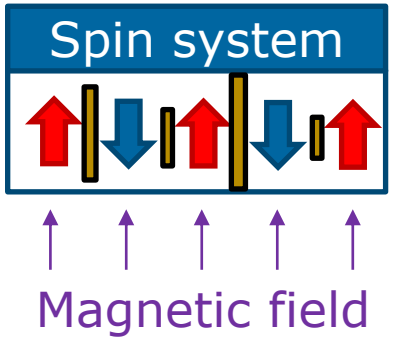
Quadratic Unconstraint Binary Optimization
Classical Ising model

- The QUBO model is equivalent to Ising model:

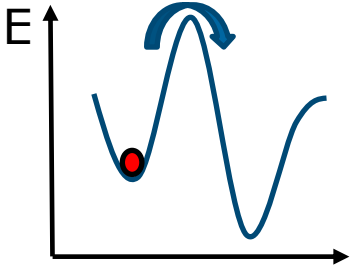
$$H(s_1,\dots,s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_i h_i s_i$$

With N spins $s_i \in \{-1; 1\}$

J_{ij} and h_i are real numbers



- The equivalence is done by a change of variable : $x_j = \frac{s_i+1}{2}$



- The formulation of many NP problems has already be done.

NP-hard problems on the **QLM**

- NP-hard problems can be formulated as **minimization or maximization** problems – with a **cost function**
- At the same time, **finding the lowest energy** of a physical system is also a **minimization problem** – with a **cost Hamiltonian**

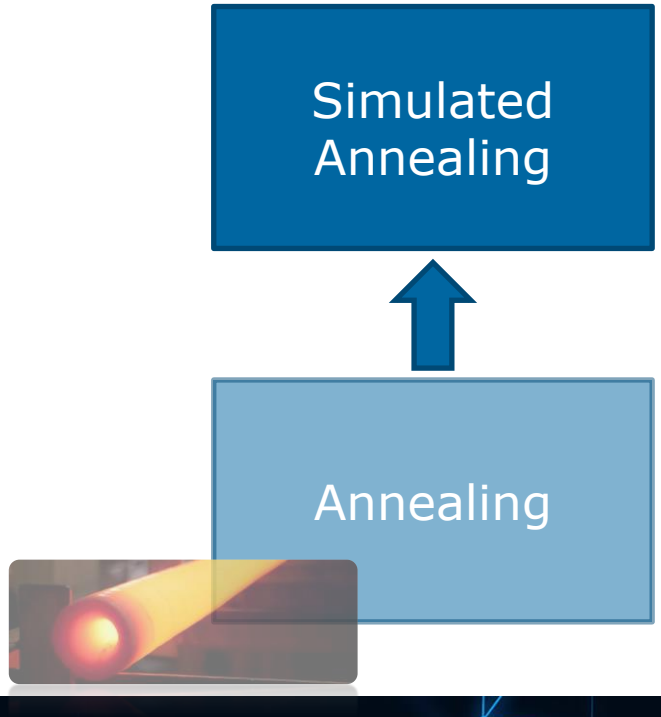
An NP problem could be therefore represented by a **cost Hamiltonian** – in an **Ising** or **QUBO** form

- The solution is encoded in the **ground state** of the **Hamiltonian**
- ***H*** is brought to this ground state by **(simulated) quantum annealing**, **QAOA**, etc.

A Story
Annealing

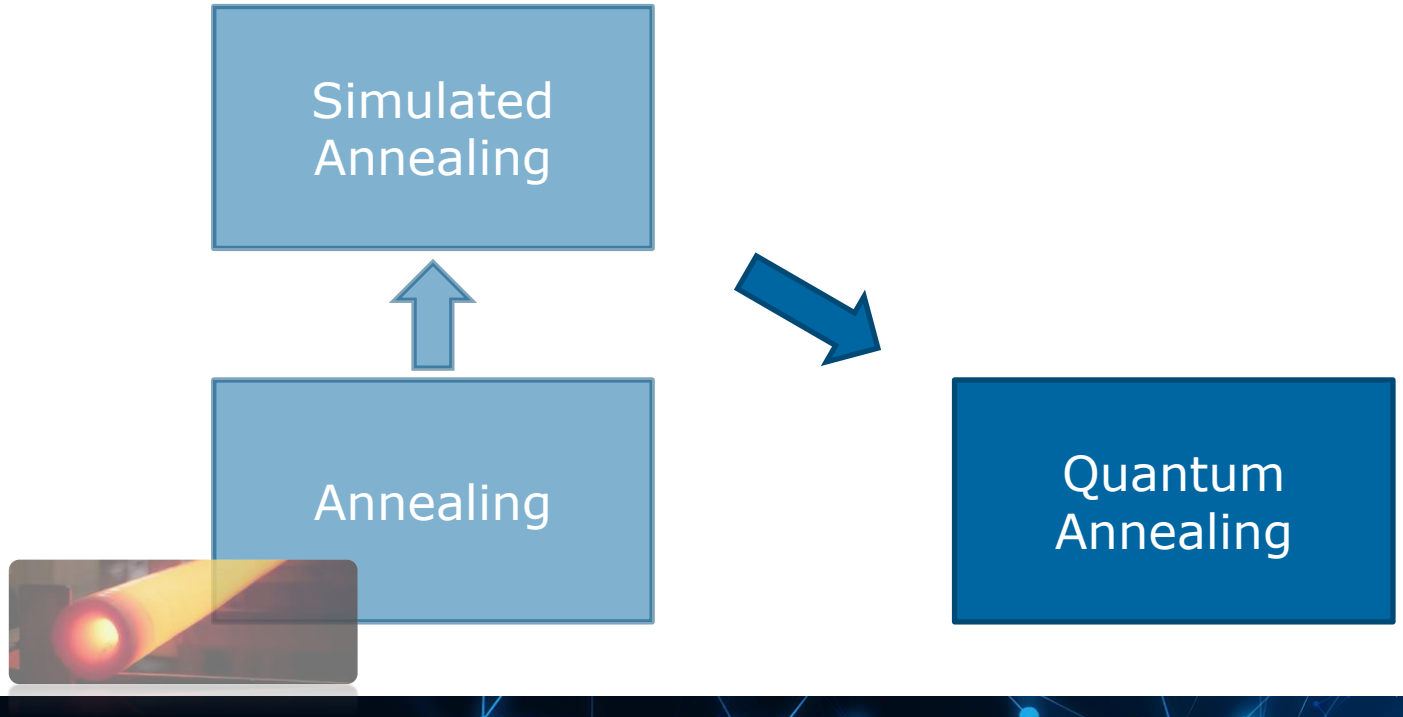
- Annealing is a technique in metallurgy to modify properties of metal.
- Annealing in metallurgy is done by heating the metal above critical temperature and slowly cooling down afterward.
- Annealing steel enhances its toughness.

A Story
Simulated Annealing



- Simulated Annealing (SA) is an metaheuristic developed in 1983 [kirkpatrick] inspired by physical phenomena.
- SA avoids local minima by using random search linked to an annealing schedule.
  - Any better modification is accepted
  - Candidates that are not better are accepted with a probability
- Spin dynamics : Markov Chain Monte Carlo (MCMC) methods

A Story
Quantum Annealing



A Story
Quantum Annealing

- Quantum Annealing (QA) is based on the adiabatic theorem.

"A physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum"

Adiabatic theorem

$H(t=0) = H_{initial}$ $H(t=1) = H_{solution}$

$H(t) = (1-t)\, H_{initial} + tH_{solution}$

A Story
Quantum Annealing

- Such Hamiltonian can be fomulated using the Transverse-Field Ising Model (TFIM) :

$$\hat{H}(t) = -\sum_{i=1}^{n} h_i \hat{\sigma}_i^z - \sum_{1 \le i < j \le n}^{n} J_{ij} \hat{\sigma}_i^z \hat{\sigma}_j^z - \Gamma(t) \sum_{i=1}^{n} \hat{\sigma}_i^x$$

Where $\hat{\sigma}_i^x = \hat{I}_2^{\otimes(i-1)} \otimes \hat{\sigma}^x \otimes \hat{I}_2^{\otimes(N-i)}$

$$\hat{\sigma}^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} ; \hat{I}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} ; \hat{\sigma}^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

A Story
Quantum Annealing

- Such Hamiltonian can be fomulated using the Transverse-Field Ising Model (TFIM) :

$$\hat{H}(t) = -\sum_{i=1}^{n} h_i \hat{\sigma}_i^z - \sum_{1 \le i < j \le n}^{n} J_{ij} \hat{\sigma}_i^z \hat{\sigma}_j^z - \Gamma(t) \sum_{i=1}^{n} \hat{\sigma}_i^x$$

$\Gamma(t = 0)$ begins at a large value so the state is dominated the x-axis.

$\Gamma(t)$ is slowly decreased until $\Gamma(t = 1) = 0$

Allowing the system to adiabatically evolves and so, stay in the eigenstate of the initial Hamiltonian

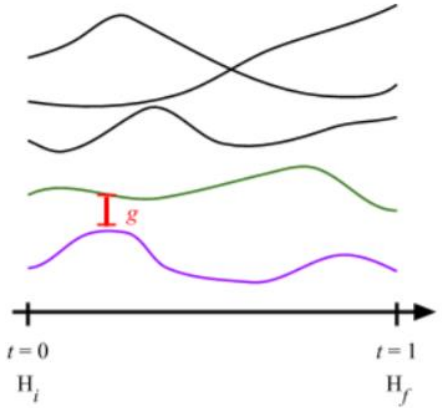
A Story
Quantum Annealing

- Quantum Annealing (QA) theoretical improvement remains unproven.

- The evolution of the system to keep the ground state during the evolution needs to be proportional to the gap between the lowest exited state and the ground state:

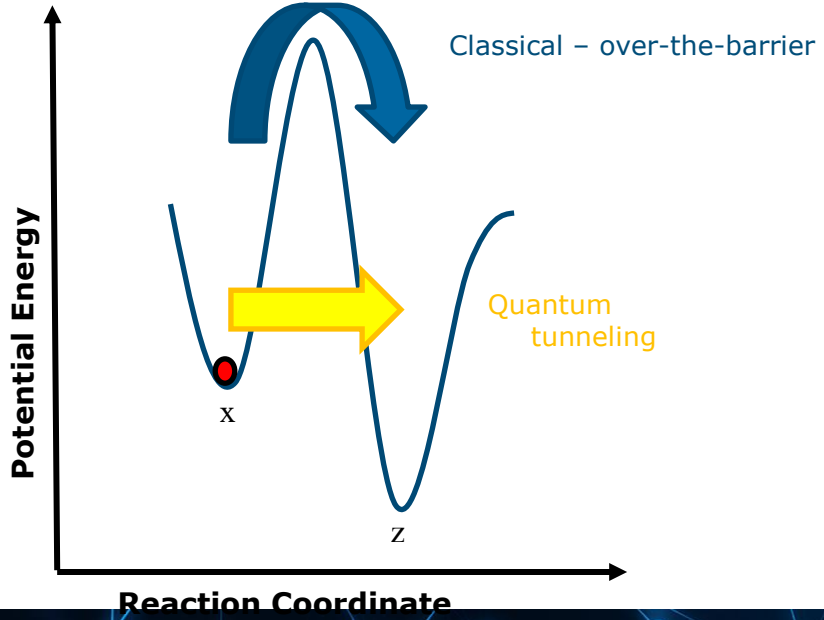
$$T = O\left(\frac{1}{g^2}\right)$$

- g is the minimum spectral gap for H(t)

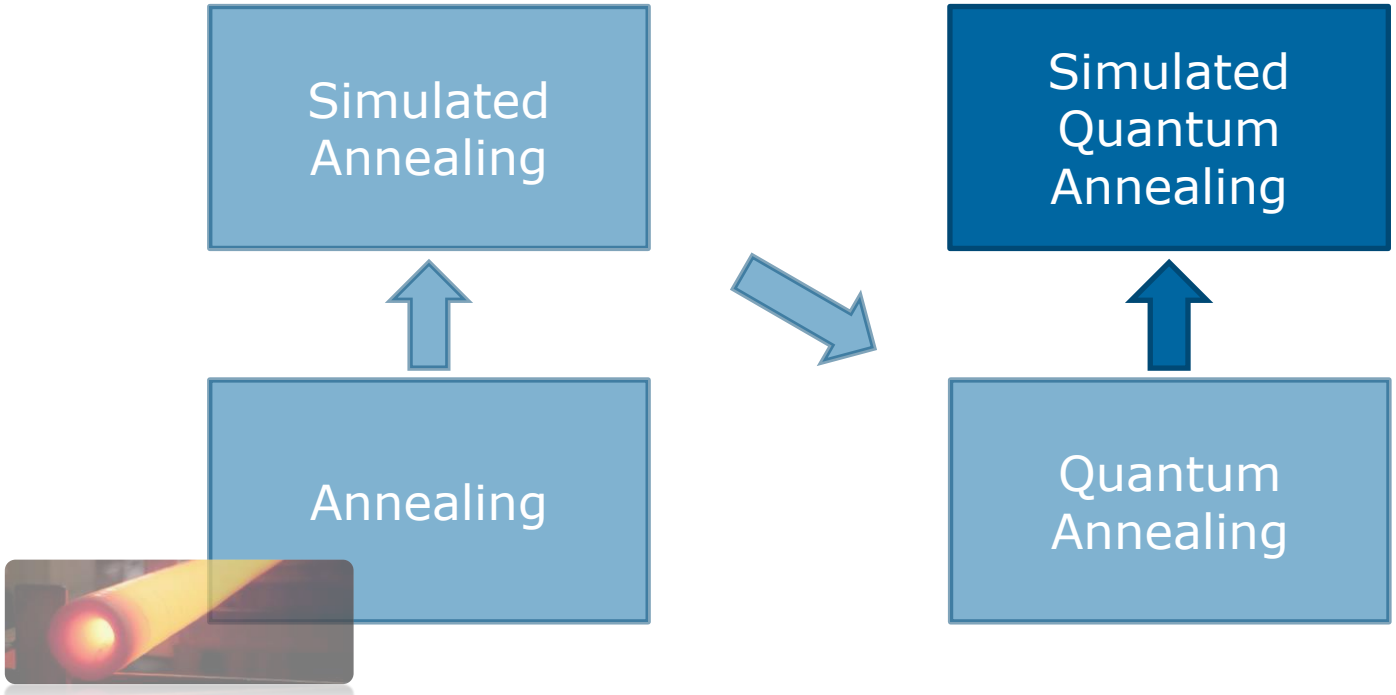


A Story
Quantum Annealing

- The hope of Quantum Annealing (QA) is to avoid being stuck in local minimal using Quantum tunnelling effect.



A Story
Simulated Quantum Annealing



Simulated Quantum Annealing

- Simulated Quantum Annealing (QA) goal is to simulate the evolution of such Hamiltonian $\widehat{H}(t)$
- SQA is a classical approach : a "quantum inspired" approach.
- Methods used: Discrete Time Path-Integral Monte Carlo (DT PIMC) algorithms.
- We can approximately map a *d*-dimensional quantum Ising system with a transverse field onto a (*d* + 1)-dimensional classical Ising system.

Simulated Quantum Annealing
Overview

- Current supported NP problems on the QLM:
  - Unconstrained Graph Problems:
    - Max Cut
    - Graph Partitioning
  - Constrained Graph Problems:
    - Graph Colouring
    - K-Clique
    - Vertex Cover
  - Other NP Problems:
    - Number Partitioning
    - Binary Integer Linear Programming

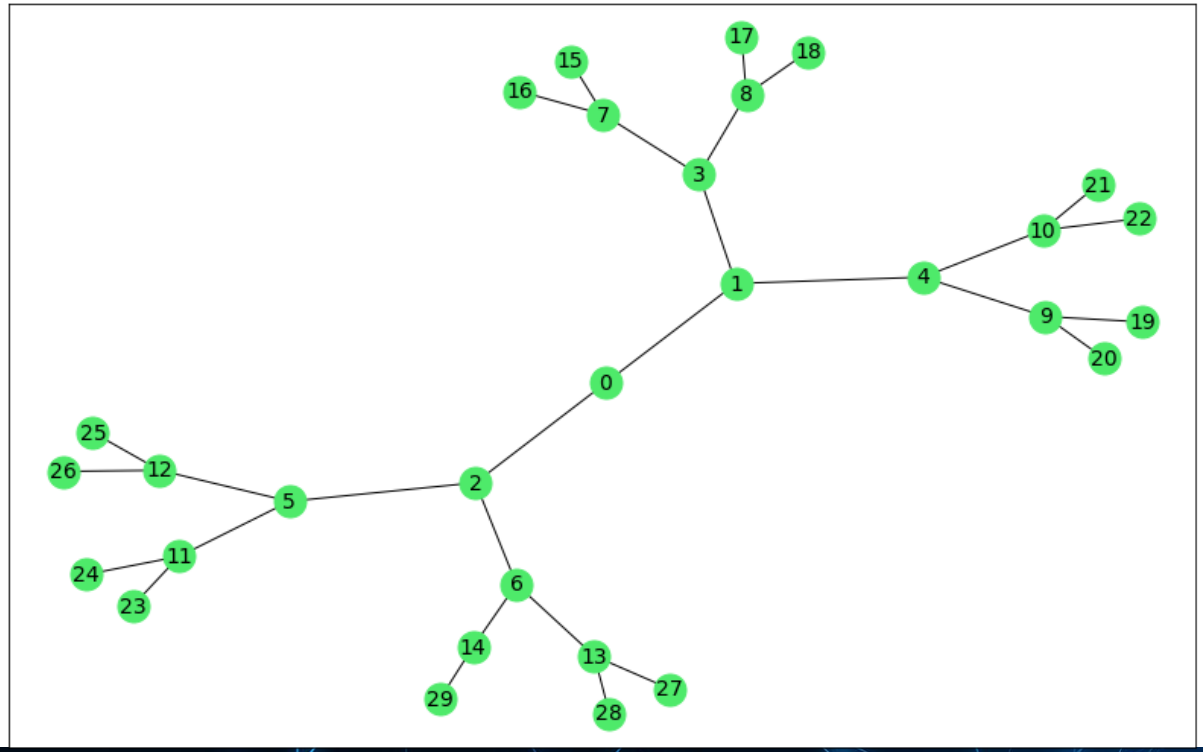
Simulated Quantum Annealing Example: Max Cut

- Given an undirected **graph G(V, E)**
  - **V** is a set of **vertices**
  - **E** is a set of **edges**

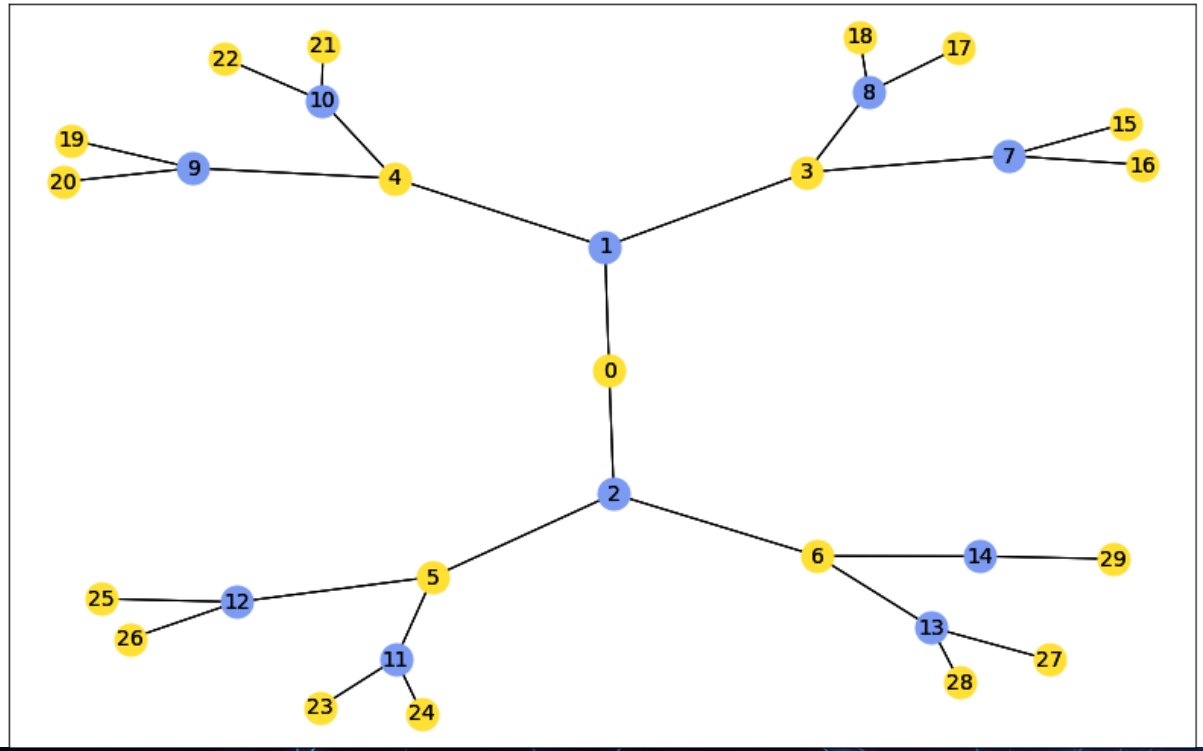
- The objective of the **Max-cut problem** is to find two sets from V that **maximize** the number of edges between these two sets.

$$H(s_1,\dots,s_N) = \sum_{(u,v)\in E} s_i s_j$$

Simulated Quantum Annealing Example: Max Cut



Simulated Quantum Annealing Example: Max Cut



Notebooks: SQA

Hands-on 6: SQA

Hands-on 6: SQA

- Log on the QLM

- Go to Hands-on 6 directory:

  http://127.0.0.1:8888/tree/notebooks/Hands-on6

- Open and complete the notebook *SQA.ipynb*

Introduction to QAOA

Trusted partner for your Digital Journey

Atos

Gate count for Shor's algorithm

How many qubits does it take to factor an integer?

- Textbook examples of quantum algorithms largely assume that qubits and gates are not subject to noise. In the **absence of noise**, to factor an integer made of 1024 bits takes about 2050 qubits and over 10^9 gates.

- In the **presence of noise**, quantum error correction has to be incorporated into the computation, adding a large overhead.

Technology	Neutral Atoms	Supercond. Qubits	Ion Traps	
Gate error	1×10^{-3}	1×10^{-5}	1×10^{-9}	
Avg. gate time	19,000 ns	25 ns	32,000 ns	
Execution time	2.62 years	10.81 hours	2.22 years	
No. qubits	5.29×10^8	4.57×10^7	1.44×10^8	
No. gates	1.02×10^{21}	2.55×10^{19}	5.10×10^{19}	
Dominant gate	$CNOT$	$CNOT$	$CNOT$	
Code distance	17	5	3	Surface code
Logical gate error	4.99×10^{-11}	2.95×10^{-11}	4.92×10^{-15}	
Logical gate time	1.29×10^5 ns	2.10×10^2 ns	5.96×10^5 ns	
No. qubits per logical	3.73×10^4	3.23×10^3	1.16×10^3	
No. gates per logical	1.11×10^5	9.60×10^3	3.46×10^3	

(from: M. Suchara et al. Arxiv:1312:3216)

The Quest for Quantum Fault-Tolerance
Why is it hard to achieve?

- For Quantum Fault-Tolerance to be achieved, one needs to be sure that physical error rates per gate are **much, much** lower than some threshold value, ie at least 100 times smaller.

The highest thresholds for **general noise** is that of the surface code are in the **1%**

- This must be compared with the noise characterized in state-of-the-art quantum processors.The shared data on *Google's Sycamore chip* is:

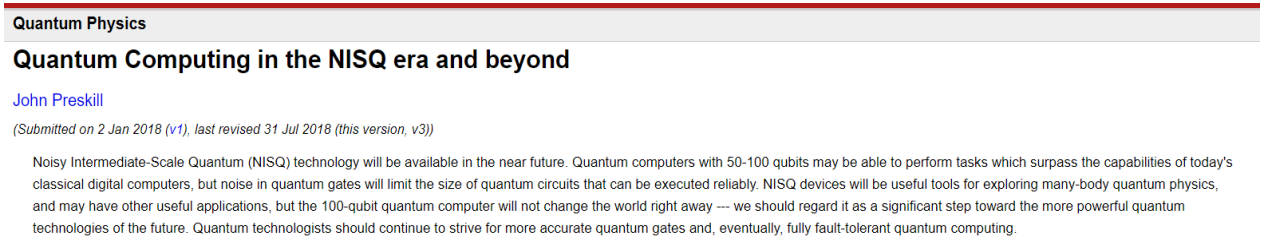
> While we are targeting 0.1% error two-qubit gates for error correction, a quantum supremacy demonstration can be achieved with 0.3-0.6% error rates.
>
> For decoherence-dominated errors, a 0.1% error means a factor of about 1000 between coherence and gate times. For example, a 25 μs coherence time implies a 25 ns gate.

(arxiv:1910.11333)

The NISQ Computing Era
A Hybrid Classical-Quantum Paradigm

- Fault-tolerant quantum computations of arbitrary length are not within reach of today's technology (need qubits in the millions)

- Algorithms for **small number of qubits** (in the ~ 100) and **limited coherence** time will need to **offload pre- and post-processing** to classical computers. Noisy Intermediate-Scale Quantum (**NISQ**) computing is a thriving field of research:



Quantum Physics

Quantum Computing in the NISQ era and beyond

John Preskill

(Submitted on 2 Jan 2018 (v1), last revised 31 Jul 2018 (this version, v3))

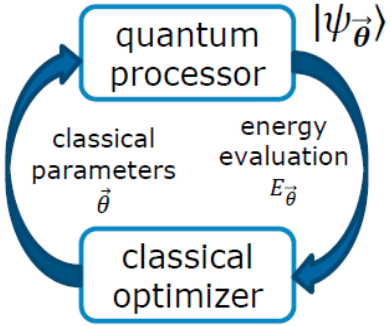
Noisy Intermediate-Scale Quantum (NISQ) technology will be available in the near future. Quantum computers with 50-100 qubits may be able to perform tasks which surpass the capabilities of today's classical digital computers, but noise in quantum gates will limit the size of quantum circuits that can be executed reliably. NISQ devices will be useful tools for exploring many-body quantum physics, and may have other useful applications, but the 100-qubit quantum computer will not change the world right away --- we should regard it as a significant step toward the more powerful quantum technologies of the future. Quantum technologists should continue to strive for more accurate quantum gates and, eventually, fully fault-tolerant quantum computing.

  - *50-500 qubits*
  - *Low depth circuits*
  - *Limited correction capabilities*
  - ***Killer apps: simulate correlated matter, machine learning, optimization***

The NISQ Computing Era
A Hybrid Paradigm: Variational Quantum Algorithms

- Goal: find **ground state energy** of a Hamiltonian H

1. Choose a (good) variational Ansatz $|\Psi(\vec{\theta})>$
2. Design a quantum circuit that implements $|\Psi(\vec{\theta})>$
3. Measure Cost Function $E_{\vec{\theta}} = <\Psi(\vec{\theta})|\mathbf{H}|\Psi(\vec{\theta})>$
4. Use classical optimizer to find optimal $\vec{\boldsymbol{\theta}}^*$

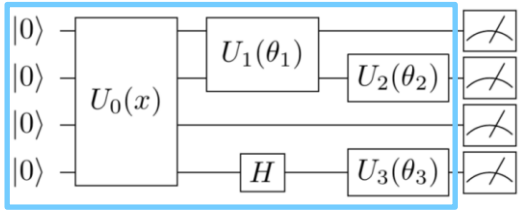


- VQAs consist of a **quantum circuit** that can prepare quantum states belonging to a **parametrized class {$|\Psi(\vec{\theta})>$}** efficiently and a **classical loop** that optimizes its parameters. The Rayleigh-Ritz variational principle ensures that the cost function converges:

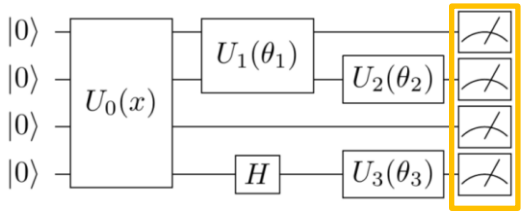
$$\langle\psi(\vec{\theta})|\, H\, |\psi(\vec{\theta})\rangle \geq E_0$$

The NISQ Computing Era
VQA: Inner Loop

- The **inner loop** prepares a state parametrized by $\vec{\theta}$ and measures its energy. It consists of
  - A quantum circuit for quantum state preparation, **given a parameter configuration** $\vec{\theta}$:



  - A series of measurements for energy evaluation. Some terms in the **Cost Function** will not commute, so cannot be measured simultaneously and the state has to be **re-prepared**.



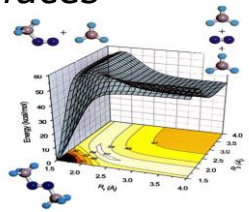
The NISQ Computing Era
VQA: Outer Loop

- The **outer loop** can be thought of as classical optimizer for the **Cost Function**:
  - Optimization of the parameter set, $\vec{\theta}$, will be **gradient-based** or **gradient-free** (BFGS, **COBYLA**, L-B, **SPSA**, Bayesian Opt.) depending on whether the energy gradient is available or not.
- In the Atos QLM you can design and plug in your own optimizer:

```python
import scipy.optimize

def my_optimizer(func, x0):
    """
    Args:
        func (function): function to optimize
        x0 (array): starting point of the optimization

    Returns:
        array, int : optimal solution found by optimizer, number of iterations
    """
    res = scipy.optimize.minimize(func, x0)
    return res.x, res.nit, None

scipy_optimizer = Optimizer(my_optimizer)

energy, _, _, _ = VQE(hamil, scipy_optimizer, simple_circuit_with_two_parameters,
                      np.array([random.random()*2*np.pi, random.random()*2*np.pi]),
                      qpu, multi_qubit=True)
print("Minimum energy =", energy)
```

The NISQ Computing Era
Some Applications

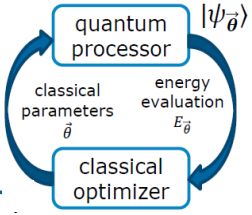
- Combinatorial Optimization
  - *Whether quantum heuristics can outperform classical approximation ratios is an open question.*
- Quantum Chemistry
  - *Improving QM/MM calculations to better approximate Potential Energy Surfaces*



- Other
  - *Certified Randomness for Proof-of-Stake Blockchain, Accelerating Sampling for ML, …*

Different kinds of Ansatz

VQA for Combinatorial Optimization
QAOA



- In the Quantum Approximate Optimization Algorithm (QAOA), the Ansatz encodes two alternating circuits, C and B, each parametrized by a number, γ or β. Ideally, the circuit outputs the solution **|γ,β>** to a combinatorial problem implicit in the circuit definition.

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = U(B, \beta_p)\, U(C, \gamma_p) \cdots U(B, \beta_1)\, U(C, \gamma_1)\ |s\rangle$$

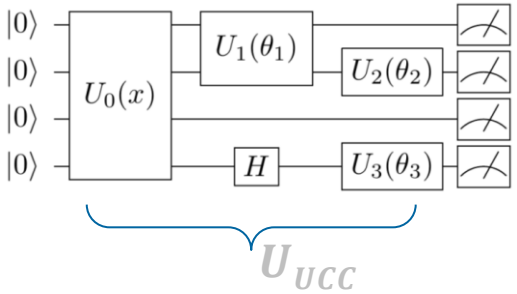
solution — Circuit (alternating circuits) — initial state

- It draws inspiration from quantum annealing, which is a quantum version of simulated annealing. It is therefore classified as a **heuristic** optimization algorithm based on **local search**.

- It remains an open question whether it can outperform classical algorithms tailored for specific problems. It is only known to work for **combinatorial problems** with **local constraints**.

VQA for Computational Chemistry
Variational Quantum Eigensolver with UCC Ansatz

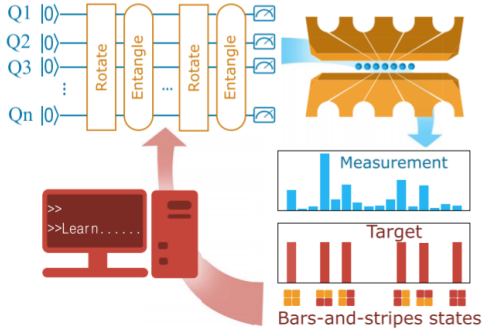
- It is posible to use VQA in combination with chemistry-inspired Ansatz, such as the **Unitary Coupled Cluster** method, in order to calculate the ground state of molecules beyond mean-field approximation, which is classically very resource consuming.



- In this case, the Ansatz is the preparation of **a tentative molecular ground state**, and the classical optimiser evaluates the fitness of the solution candidates based on their measured energy. This holds the promise to study large molecular complexes from first principles to unprecedented accuracy.

VQA for Machine Learning
Quantum Neural Networks

- If instead of optimising an energy, the goal is to minimise a **distance between probability distributions**, then it is possible to use the VQA as a quantum neural network.



(from: D.Zhu et al. Science Advances 2019)

- A neural networks takes as input a (classical) probability distribution ρ_{input}. The output of a VQA is also a probability distribution ρ_{output}. Minimising, for example, the Kullback-Leibler divergence between these distributions amounts to training a generative quantum neural net.

- In this case, the Ansatz is just a **compressed description** of the input probability distribution.

Quantum Approximate Optimization Algorithm
Introduction

- What is it?

QAOA is a quantum heuristic, general purpose algorithm that is conceived to solve general combinatorial problems on NISQ processors

- Motivation:
  1. **Error mitigation**: *Evidence points towards the fact that QAOA is robust against quantum noise beyond what would be expected for low-depth circuits.*

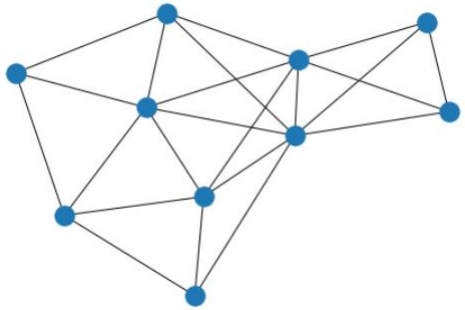
  2. **Alternating Ansatz as a workaround for small gaps**: *It could be that using the alternating Ansatz (see below) allows to find paths between eigenstates of $H_{initial}$ and $H_{solution}$ that are more robust in terms of inverse gap scaling.*

Quantum Approximate Optimization Algorithm
Problem Encoding

- As with QA, the idea is to encode the problem as an **energy landscape**. In other words, given a set of combinatorial constraints, it costs energy to violate any one constraint. Conversely minimizing the energy will be equivalent to satisfying the constraints.

- **MaxCut** was the first problem tackled by QAOA. Given a graph, solving **MaxCut** implies finding a graph partition such that the total number of edges between the transition is maximized.

$$H = \sum_{(i,j) \in E} (1 - z_i \, zj) \, /2, z \in [-1,1]$$



Quantum Approximate Optimization Algorithm
Ansatz

- The initial state in QAOA is an equal superposition of all possible partitions. This corresponds to the initial state $|s\rangle = |+\rangle|+\rangle\ldots|+\rangle \sim |00\ldots0\rangle + |00\ldots1\rangle + \ldots + |11\ldots0\rangle + |11\ldots1\rangle$, and then it applies two unitary operators, U(C,γ) and U(C,β), in alternation:

EXPLOITATION

$$U(C,\gamma) = e^{-i\gamma C} = \prod_{\alpha=1}^{m} e^{-i\gamma C_\alpha}$$

$C_{ij} = (1 - zi\ zj)/2$

EXPLORATION

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}$$

- So the total effect on the initial state is:

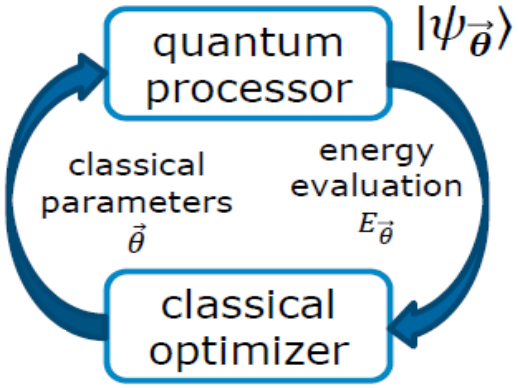
$$|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle = U(B,\beta_p)\,U(C,\gamma_p)\cdots U(B,\beta_1)\,U(C,\gamma_1)\,|s\rangle$$

solution — Circuit (alternating circuits) — initial state

(unlike in QA, in QAOA the angles at each step are not required to be small)

Quantum Approximate Optimization Algorithm
Optimizers

- In the language of NISQ processors, QAOA consists of an **inner loop**, which implements an exploration-exploitation strategy, and the **outer loop**, which can be thought about optimising the parameters for this exploration-exploitation



- Several possibilities exist for the optimisation of external parameters. In the QLM there are predefined optimisers, but you can define your own.

Quantum Approximate Optimization Algorithm
Comparison with Classical methods

- Finding a solution 20% away of the optimal one can be done classically using the **Goemans-Williamson algorithm** with runtime O(# vertices * # edges). QAOA, with a runtime of O(# vertices * # alternating steps) performed better than best known classical algorithms, but soon afterwards classical algorithms beating QAOA were found.

- This is at the bleeding edge of research. Over the last 2 years (Jan 2020):
  - Lloyd showed that QAOA is universal (arxiv.org/abs/1812.11075)
  - Hastings found a class of local search classical algorithms with better scaling than QAOA (arxiv.org/pdf/1905.07047.pdf)
  - Bravyi et al. suggested a non-local version of QAOA(arxiv.org/pdf/1910.08980.pdf) with better scaling than local QAOA

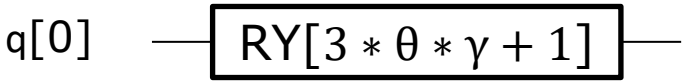
- **Challenge**: find a class of problems for which QAOA is strictly better than the best classical general purpose algorithms.

pyAQASM – Parametrized circuit

```
from qat.lang.AQASM import *
prog = Program()
#Define your variables
theta = prog.new_var(float, "\\theta")

#Apply a gate with a variable
prog.apply(RY(theta), qubits_reg[0])

#Create and display the circuit
circuit = prog.to_circ()
%qatdisplay circuit
```

pyAQASM – Parametrized circuit

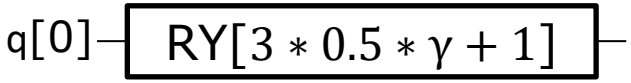
```
from qat.lang.AQASM import *
prog = Program()
#Define your variables
theta = prog.new_var(float, "\\theta")
gamma = prog.new_var(float, "\\gamma")

#Apply a gate with a variable
prog.apply(RY(3 * theta * gamma +1), qubits_reg[0])

#Create and display the circuit
circuit_tetha_gamma = prog.to_circ()
%qatdisplay circuit_tetha_gamma
```



pyAQASM – Bind variables

```
new_circuit = circuit.bind_variables({"\\theta": 0.5})
%qatdisplay new_circuit
```


```
new_circuit = circuit_tetha_gamma.bind_variables({"\\theta": 0.5})
%qatdisplay new_circuit
```




```
new_circuit = new_circuit.bind_variables({"\\gamma": 0.5})
%qatdisplay new_circuit
```


```
new_circuit = circuit_tetha_gamma.bind_variables({"\\theta": 0.5, "\\gamma": 0.1})
%qatdisplay new_circuit
```

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2)
# Add a new clause : XOR of the two variables
problem.add_clause(var1 ^ var2)

print(problem)
```

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2)
# Add a new clause : XOR of the two variables
problem.add_clause(var1 ^ var2)

print(problem)
```

```
MyProblem:
 2 variables, 2 clauses
```

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

print(problem)
```

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

print(problem)
```

```
MyProblem:
 2 variables, 1 clauses
```

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

obs = problem.get_observable()

print(obs)
```

- A diagonal Hamiltonian encoding the cost function of the problem can be extracted

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

obs = problem.get_observable()

print(obs)
```

- A diagonal Hamiltonian encoding the cost function of the problem can be extracted

```
0.125 * I^2 +
-0.125 * (Z|[0]) +
0.125 * (ZZ|[0, 1]) +
-0.125 * (Z|[1])
```

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

obs = problem.get_observable()

print(obs)
```

- A diagonal Hamiltonian encoding the cost function of the problem can be extracted

```
0.125 * I^2 +
-0.125 * (Z|[0]) +
0.125 * (ZZ|[0, 1]) +
-0.125 * (Z|[1])
```

pyAQASM – CombinatorialProblem

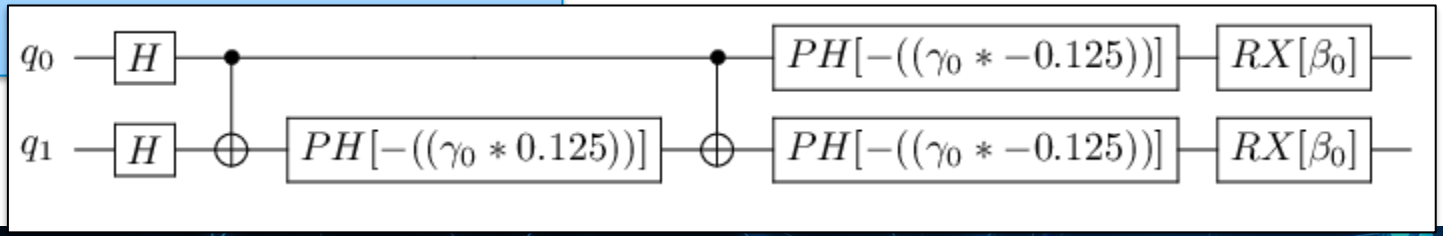
```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

ansatz = problem.qaoa_ansatz(1)
circuit = ansatz.circuit
%qatdisplay circuit
```

- And it is possible to directly generate a QAOA ansatz from a CombinatorialProblem

pyAQASM – CombinatorialProblem

```
from qat.opt import CombinatorialProblem

problem = CombinatorialProblem("MyProblem")
# Declare two fresh variables
var1, var2 = problem.new_vars(2)
# Add a new clause : logical AND of the two variables
problem.add_clause(var1 & var2, weight=0.5)

ansatz = problem.qaoa_ansatz(1)
circuit = ansatz.circuit
%qatdisplay circuit
```

- And it is possible to directly generate a QAOA ansatz from a CombinatorialProblem
- The circuit is accessible from the ansatz



pyAQASM – ScipyMinimizePlugin

```
from qat.qpus import get_default_qpu
from qat.plugins import ScipyMinimizePlugin
qpu = get_default_qpu()
stack = ScipyMinimizePlugin(method="COBYLA",
                            tol=1e-2,
                            options={"maxiter":150}) | qpu

result = stack.submit(ansatz)
print("Final energy:", result.value)
```

- Using ScipyMinimizePlugin we can directly minimize our ansatz

pyAQASM – ScipyMinimizePlugin

```
from qat.qpus import get_default_qpu
from qat.plugins import ScipyMinimizePlugin
qpu = get_default_qpu()
stack = ScipyMinimizePlugin(method="COBYLA",
                            tol=1e-2,
                            options={"maxiter":150}) | qpu

result = stack.submit(ansatz)
print("Final energy:", result.value)
```

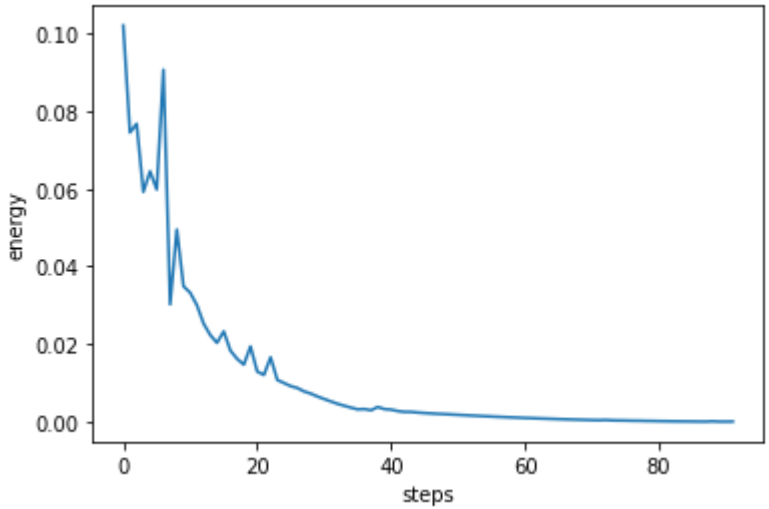
- Using ScipyMinimizePlugin we can directly minimize our ansatz

```
Final energy: 7.20408976095793e-05
```

pyAQASM – ScipyMinimizePlugin

```
import matplotlib.pyplot as plt

plt.plot(eval(result.meta_data["optimization_trace"]))
plt.xlabel("steps")
plt.ylabel("energy")
plt.show()
```

- You can print the trace of the optimization



pyAQASM – ScipyMinimizePlugin

```
from qat.qpus import get_default_qpu
from qat.plugins import ScipyMinimizePlugin
qpu = get_default_qpu()
stack = ScipyMinimizePlugin(method="COBYLA",
                            tol=1e-2,
                            options={"maxiter":150}) | qpu

result = stack.submit(ansatz)
print("Final energy:", result.value)
```

- Multiple methods are already available:

Nelder-Mead	SLSQP
Powell	trust-constr
CG	dogleg
BFGS	trust-ncg
Newton-CG	trust-exact
L-BFGS-B	trust-krylov
TNC	
COBYLA	

pyAQASM
Already implemented combinatorial problems

- Unconstrained Graph Problems:
  - Max cut
  - Graph Partitioning
- Constrained Graph Problems:
  - Graph Colouring
  - K-Clique
  - Vertex Cover
- Other problems:
  - Number Partitioning
  - Binary Integer Linear Programming

Hands-on 7: QAOA

Hands-on 7: QAOA

- Log on the QLM

- Go to Hands-on 7 directory:

  http://127.0.0.1:8888/tree/notebooks/Hands-on7

- Open and complete the notebook *QAOA.ipynb*

Thank you.

Gaëtan Rubez

Quantum Computing Expert

gaetan.rubez@atos.net