Applications court terme du calcul quantique

Ou comment travailler avec des qubits bruités

Simon Martiel

10-04-2019

Short term?

- QEC
- 10^6 qubits
- large circuits

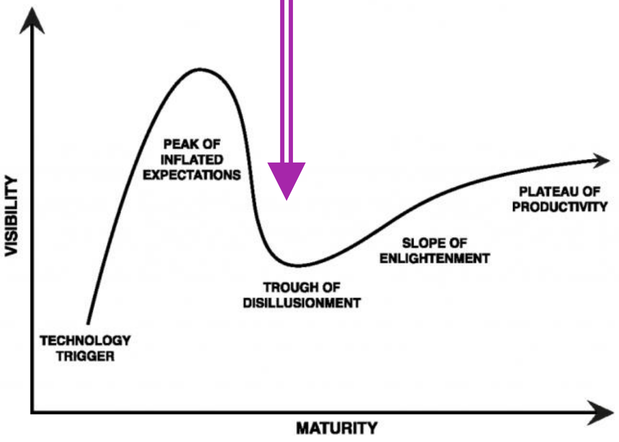
- no QEC
- < 100 qubits
- $\frac{T_1}{gate\ time} < 10^3$

Why work with the paddle boat?

- Quantum $upremacy might be closer than it looks

- Making early progress might bootstrap the industrialisation process (or at least slow down the investment trough)

this guy here



- Help & improve hardware design:
  - connectivity?
  - premices of QEC

Very different software approaches…

Compilation	HW dep. (CNOTS -> CZ)	QEC dep. (H/T synthesis, lattice surgery)
Error correction/mitigation	HW dep. mitigation (DFS, Monte Carlo, etc)	Full blown QEC (surface code, color code)
Ressources	Scarce (e.g reuse ancillas)	« less » scarce (e.g runtime ancilla handling)
Software stack	Ad hoc, low level optimization	Highly optimized libraries quantum g++ style

… with a similar end goal
(on the programing side, at least)

- Most users won’t/shouldn’t write quantum circuits
  (how many of you write code in x64?)

- Need for high level interfaces of quantum « backends »

- Think of BLAS/MKL or Cuda

- Can this be achieved for short term applications?

Variational Quantum Eigensolver
for combinatorial optimization (aka QAOA)

this talk

- Applications
- High level library
- ad hoc synthesis
- pyAQASM
- quantum processor

```
my_problem = Problem()

var1 = my_problem.new_var()
var2 = my_problem.new_var()
var3 = my_problem.new_var()

my_problem.add_clause(var1 ^ var2)
my_problem.add_clause(var2 | var3 & var1)
my_problem.add_clause(~var1)
```

```
circuit = my_problem.generate_circuit([0.1, 1.6, 2.4], [1., 2., 3.],
                                      optimization_strategy="coloring")
```

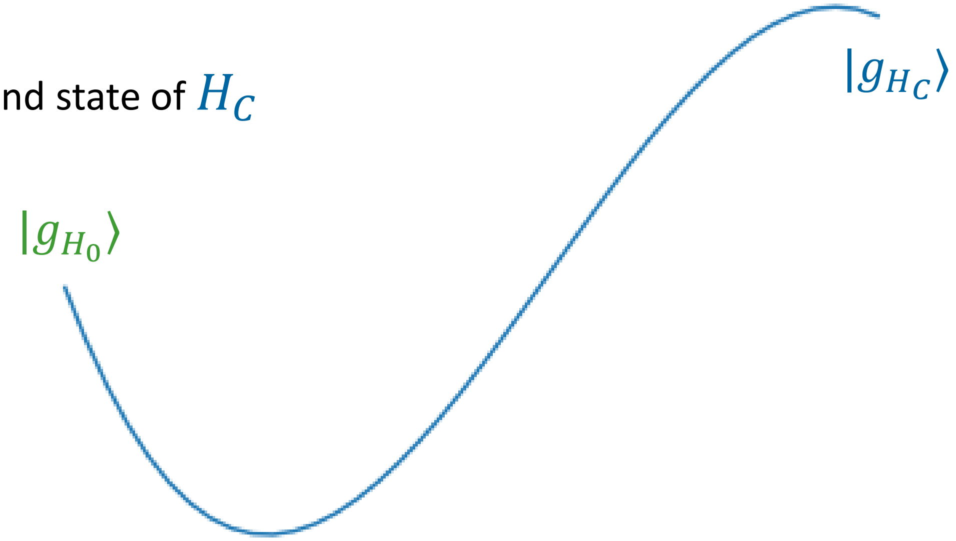
```
val = my_problem.evaluate([0.1, 1.6, 2.4], [1., 2., 3.],
                          qpu=qat.linalg.get_qpu_server(),
                          optimization_strategy="parity")
```

1

From adiabatic Quantum Computation to quantum circuits

Turning analogic computation into discrete time computation

- Some Hamiltonian H_C encoding your problem
- Solution of your problem : $\lambda_g, |g_{H_C}\rangle$ the ground state of H_C
- Adiabatic analog approach to find $|g\rangle$:
  - start by driving $H_0 = -\sum \sigma_x^i$
  - change the drive from H_0 to H_C:



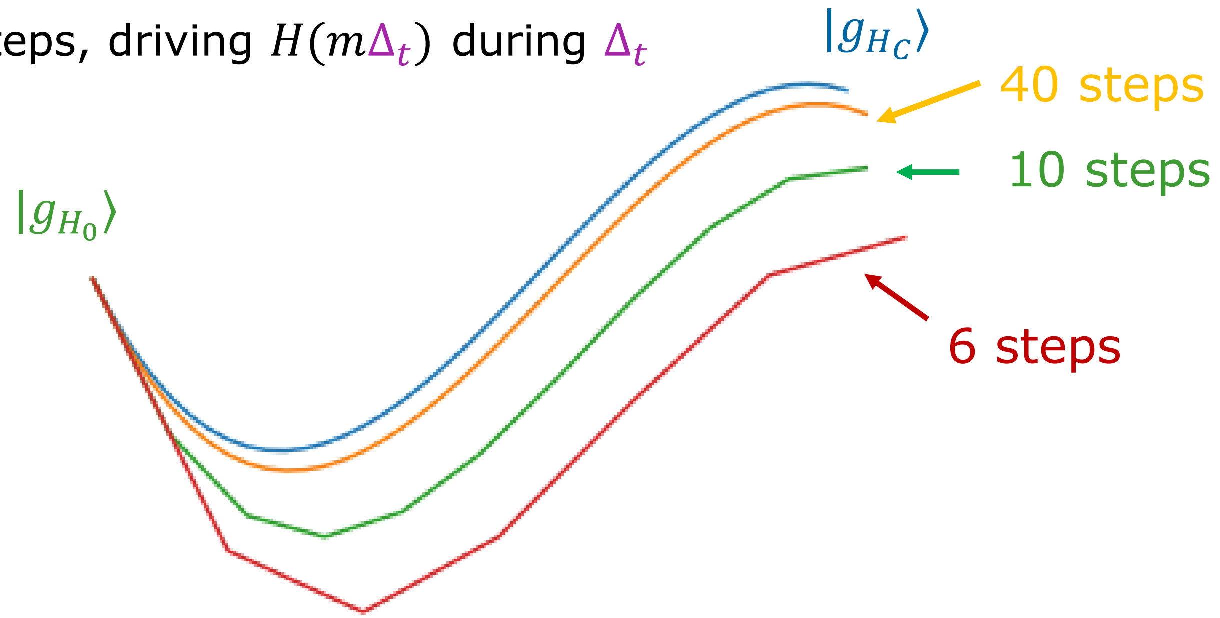
$$H(t) = (t_{max} - t)H_0 + t\, H_C$$

- <u>Adiabatic Theorem:</u>

$\left|\langle g_{H_C}|\psi(t_m)\rangle\right| \to 1$ when $t_{max} \to \infty$ « if it is slow enough, we will end up with $|g_{H_C}\rangle$»

Turning analogic computation into discrete time computation

- We want to discretize this path
- Lets do baby steps, driving $H(m\Delta_t)$ during Δ_t



Turning analogic computation into discrete time computation

- Our path is, in fact, some unitary operator (Shrödinger):

$$U_{tot} = exp\left(-i \int_{t=0}^{t_{max}} H(t)dt\right)$$

We split the computation into many baby steps of duration Δ_t:

$$U_m = \exp(-i \, \Delta_t H_m)$$

- We can do a first approximation:

$$U'_{tot} = \prod_m U_m \qquad \text{with error } O(\sqrt{\Delta_t poly(n)})$$

- Followed by another approximation:

$$U'_m = \exp\left(-i\Delta_t \left(1 - \frac{m\Delta_t}{t_{max}}\right) \mathrm{H}_0\right) \exp\left(-i\Delta_t \, \frac{m\Delta_t}{t_{max}} \mathrm{H}_C\right) \quad \text{with error } O(\Delta_t^2 \| H_0 H_C \|)$$

$U'_{m,0}$: sequence of R_x $\qquad$ $U'_{m,C}$: ???

Adiabatic quantum computation, Albash & Lidar

A discrete time algorithm for combinatorial optimization
or the so-called QAOA algorithm

- Given a function $C : \{0,1\}^n \rightarrow \mathbb{R}$ to optimize (i.e find $z^* = \mathrm{argmax_z}\ \{\mathrm{C(z)|z} \in \{0{,}1\}^n\}$)
  - Encode C into an Hamiltonian:

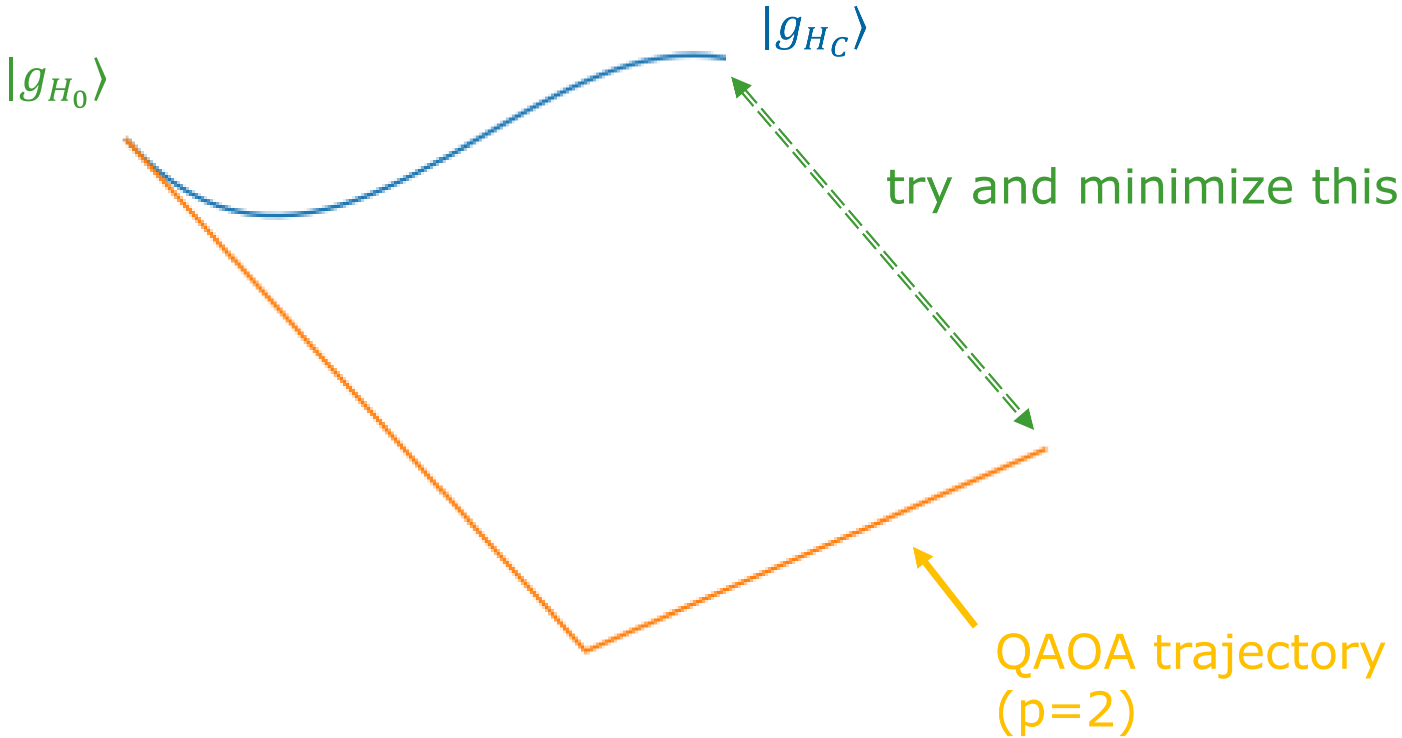
$$H_C = \sum_{z \in \{0,1\}^n} C(z)|z\rangle\langle z|$$

  - Prepare state $|+^n\rangle$
  - Apply circuit : $\prod_m U'_{m,0} U'_{m,C}$
  - Measure all the qubits
- If the number of steps is large enough, the final state should be close to $|z^*\rangle$
- In practice, we limit ourselves to a few steps:

$$|\psi(\gamma,\beta)\rangle = \left[\prod_{m=1}^{p} U_0(\beta_m) U_C(\gamma_m)\right] H^{\otimes n}\ |0^n\rangle$$

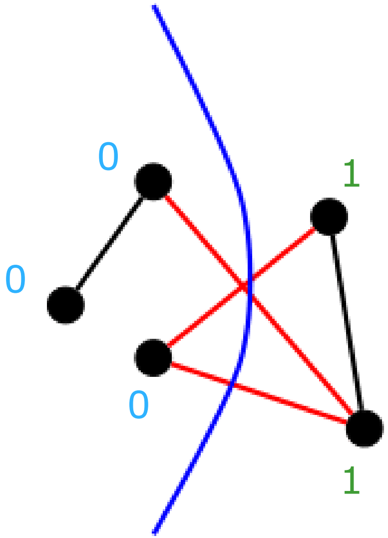
- Use a classical optimizer to maximize $\mathrm{F}(\gamma,\beta) = \langle\psi(\gamma,\beta)|H_C|\psi(\gamma,\beta)\rangle$

A discrete time algorithm for combinatorial optimization
or the so-called QAOA algorithm



Atos

Example : MaxCut



$C(00011) = 3$

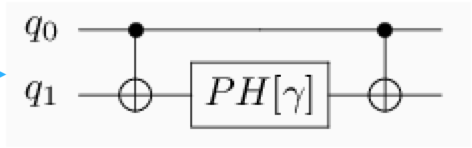
- Given a graph $G = (V, E)$ define:

$$C(z) = \sum_{(i,j)\in E} z_i \oplus z_j$$

- $\langle x_0 x_1 | \frac{1-\sigma_z \sigma_z}{2} | x_0 x_1 \rangle = (x_0 \oplus x_1)$, hence:

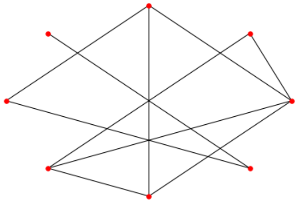
$$H_C = \sum_{(i,j)\in E} \frac{1 - \sigma_z^i \sigma_z^j}{2}$$

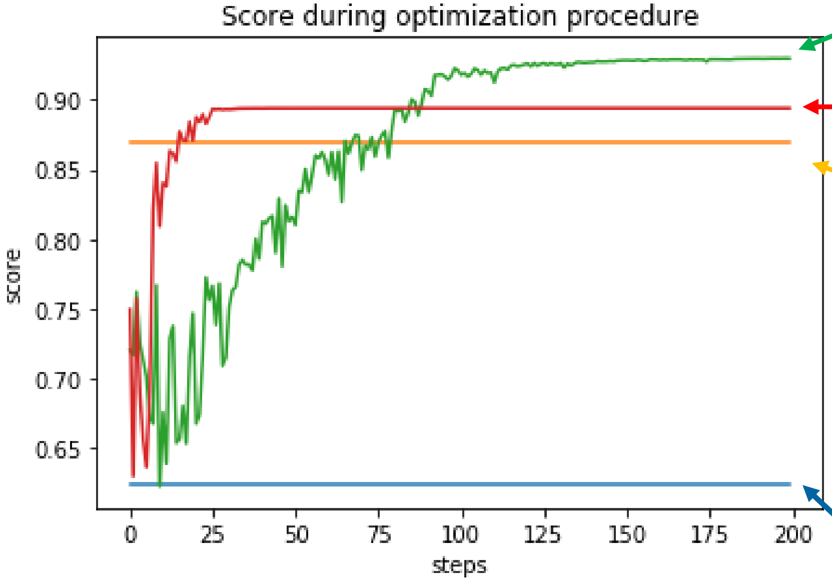
- $U_C(\gamma) = \exp(-i\gamma H_C) = \prod_{(i,j)\in E} \exp \boxed{\left(-i \frac{\gamma\left(1-\sigma_z^i \sigma_z^j\right)}{2}\right)}$



Example : MaxCut

G=





p=10

p=4

good classical approximation scheme ($\alpha \approx 0.878$)

random sampling

2 Synthesizing circuits

From a « classical » cost function C to H_C

- Usually: $C(z) = \sum_\alpha w_\alpha C_\alpha(z)$ where $C_\alpha : \{0,1\}^n \rightarrow \{0,1\}$ are « local » and $w_\alpha \in \mathbb{R}$
- Lets assume that each C_α is described using a boolean formula:

$$F \coloneqq F \wedge F \mid F \vee F \mid F \oplus F \mid \neg F \mid x_i$$

- We define H_{C_α} by induction as follows:

$$H_{x_i} := \frac{1-\sigma_z^i}{2} \qquad H_{\neg F} := 1 - H_F$$

$$H_{F_1 \wedge F_2} := H_{F_1} H_{F_2} \qquad H_{F_1 \vee F_2} \coloneqq H_{F_1} + H_{F_2} - H_{F_1} H_{F_2}$$

$$H_{F_1 \oplus F_2} \coloneqq H_{F_1} + H_{F_2} - 2 H_{F_1} H_{F_2}$$

- Ex: for MaxCut :
  - $C_\alpha(z) = z_i \oplus z_j$
  - $H_{z_i \oplus z_j} = H_{z_i} + H_{z_j} - 2\, H_{z_i} H_{z_j} = \frac{1-\sigma_z^i}{2} + \frac{1-\sigma_z^j}{2} - \frac{1}{2}(1-\sigma_z^i)(1-\sigma_z^j) = \frac{1-\sigma_z^i \sigma_z^j}{2}$

Applications

High level library

ad hoc synthesis

pyAQASM

quantum processor

Synthesizing $U_C(\gamma)$

- Usually :

« local » functions

$$C(z) = \sum_\alpha C_\alpha(z)$$

- Hence:

« local » commuting unitaries

$$U_C(\gamma) = \prod_\alpha \overbrace{\exp(-i\gamma H_{C_\alpha})}$$

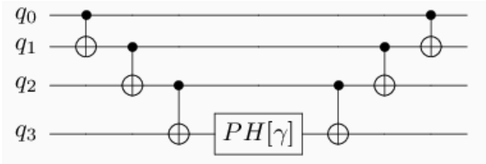
contains only σ_z

Atos

Synthesizing $U_C(\gamma)$: first strategy (graph coloration)

- Synthesize each $\exp(-i\gamma H_{C_\alpha})$ individually:

  $\exp(-i\gamma\sigma_z\sigma_z \dots \sigma_z)$ $\longrightarrow$

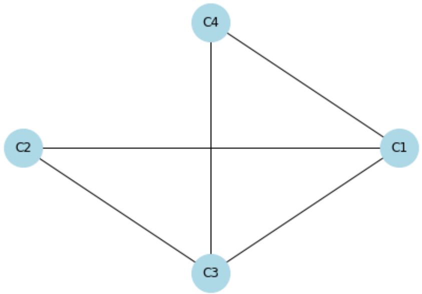


« parity phase shift »

- Find a good order to enumerate the clauses C_α:
  - G_C : intersection graph of the clauses C_α

E.g : $C(z) = \overbrace{z_0 \oplus z_1}^{C_1} + \overbrace{z_1 \wedge z_3}^{C_2} + \overbrace{z_3 \vee z_0}^{C_3} + \overbrace{z_0 \wedge z_2}^{C_4}$

  - Find a clean coloration of G_C
  - build the circuit according to the coloration

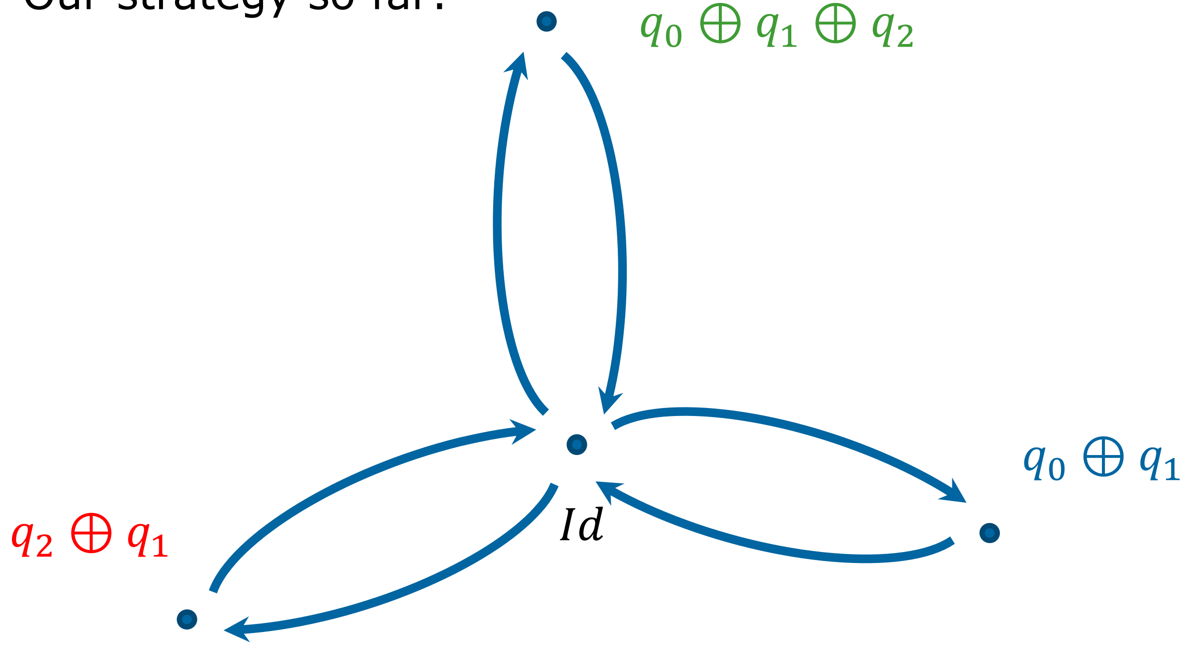


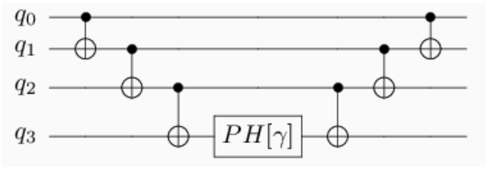
- In practice: greedy heuristic for graph coloration works fine (depth <= max degree +1).

Atos

Synthesizing $U_C(\gamma)$: another strategy (parity network synthesis)

- Our strategy so far:

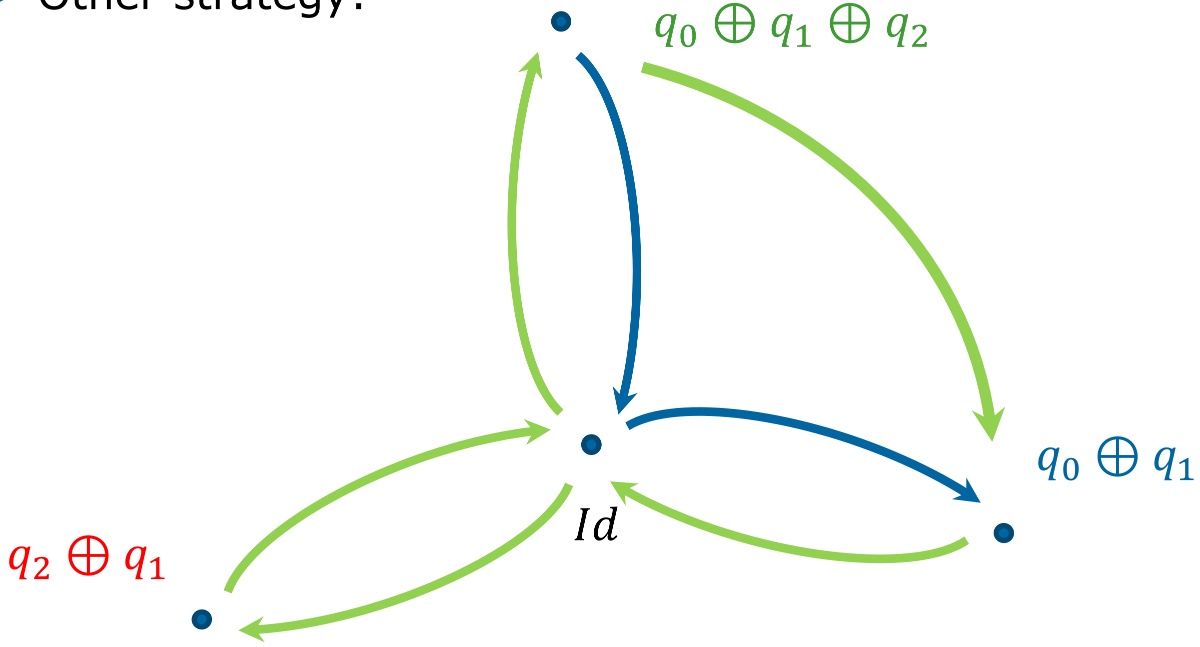


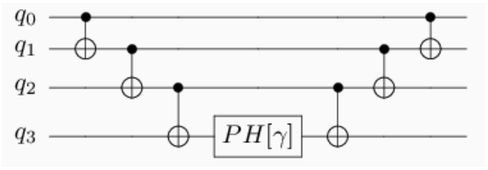


$$\sigma_Z^0 \sigma_Z^1 \sigma_Z^2 \sigma_Z^3$$

Synthesizing $U_C(\gamma)$: another strategy (parity network synthesis)

- Other strategy:

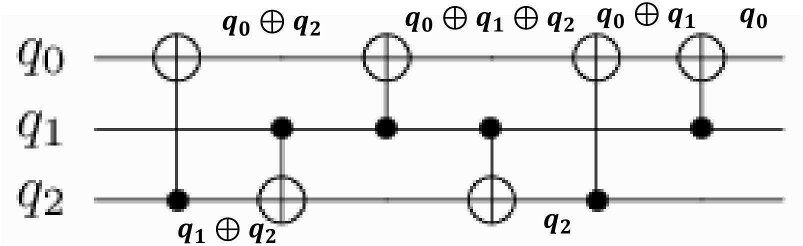




$$\sigma_Z^0 \sigma_Z^1 \sigma_Z^2 \sigma_Z^3$$

Synthesizing $U_C(\gamma)$: another strategy (parity network synthesis)

- $U_C(\gamma)$ has a very particular form: $|x\rangle \mapsto \exp\big(iP(x)\big)\,|x\rangle$ where:
  - $P: \{0,1\}^n \to \mathbb{R}$
  - $P = \sum_i \theta_i f_i$ where $f_i: \{0,1\}^n \to \{0,1\}$ is linear and $\theta_i \in \mathbb{R}$
- We consider annotated CNOT circuits:



- A parity network for a such a polynomial P is a CNOT circuit such that for all f_i in P, f_i appears in the corresponding annotated circuit.

Synthesizing $U_C(\gamma)$: another strategy (parity network synthesis)

$$C(z) = z_0 \oplus z_1 + z_1 \wedge z_3 + z_3 \vee z_0 + z_0 \wedge z_2$$

$q_0 \oplus q_1$	-0.5
q_1	-0.25
$q_1 \oplus q_3$	0.25
q_3	-0.5
$q_0 \oplus q_3$	-0.25
q_0	-0.5
$q_0 \oplus q_2$	0.25
q_2	-0.25

In [8]:
```
pb = Problem()
variables = [pb.new_var() for _ in range(4)]
pb.add_clause(variables[0] ^ variables[1])
pb.add_clause(variables[1] & variables[3])
pb.add_clause(variables[3] | variables[0])
pb.add_clause(variables[0] & variables[2])
print(pb)
```

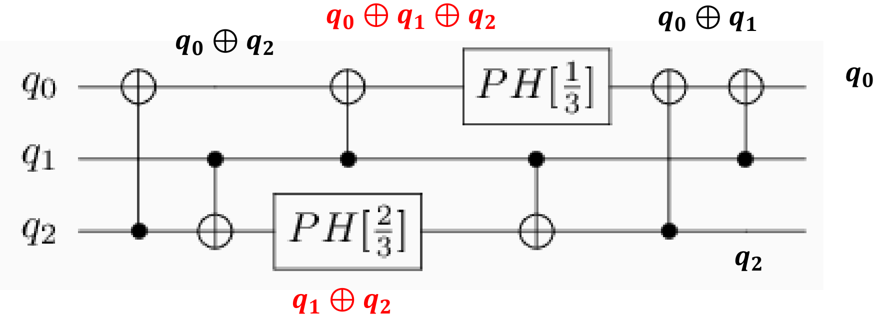
```
Problem over 4 variables:
Clause #0: V(0) ^ V(1)
Clause #1: V(1) & V(3)
Clause #2: V(3) | V(0)
Clause #3: V(0) & V(2)
```

In [7]:
```
obs = pb.generate_cost_observable()
print(obs)
```

```
1.75 * I^4 +
-0.5 * (ZZ | [0,1]) +
-0.25 * (Z | [1]) +
0.25 * (ZZ | [1,3]) +
-0.5 * (Z | [3]) +
-0.25 * (ZZ | [3,0]) +
-0.5 * (Z | [0]) +
0.25 * (ZZ | [0,2]) +
-0.25 * (Z | [2])
```

Synthesizing $U_C(\gamma)$: another strategy (parity network synthesis)

- E.g: $P(x) = \frac{2}{3}\, x_1 \oplus x_2 + \frac{1}{3}\, x_0 \oplus x_1 \oplus x_2$



Synthesizing $U_C(\gamma)$: another strategy parity network synthesis

- Finding minimal (in # of CNOTs) parity networks is hard[1]

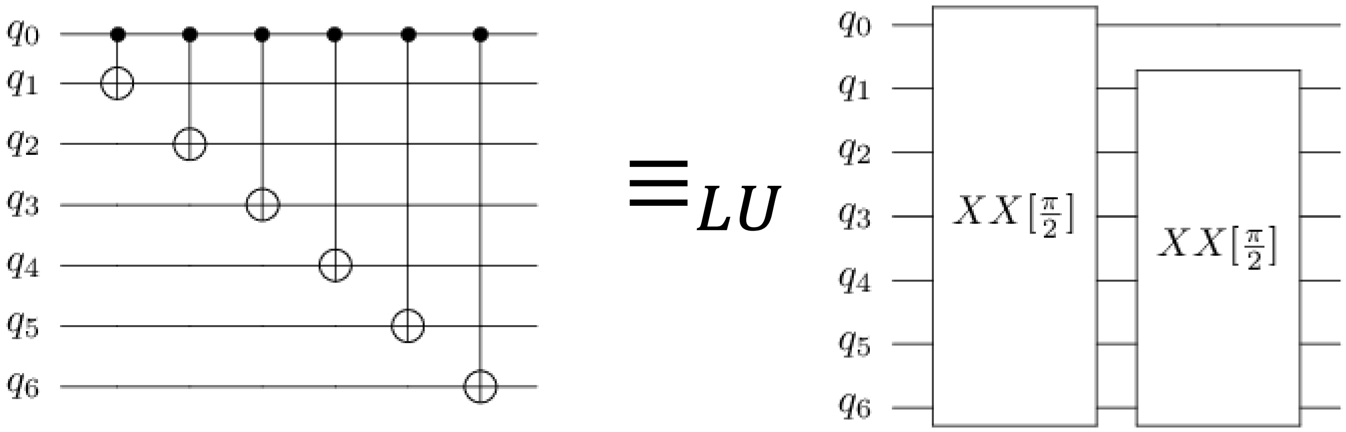
- But there exists some nice heuristics :
  - based on Gray-code enumeration
  - provably optimal when P contains all the possible f_i

- For a random cost function C over 10 bits:
  - parity network synthesis : 50% less CNOTs compared to naive approach
  - but no control on the circuit depth

[1]. On the cnot complexity of cnot-phase circuits, by Amy, Azimzadeh, Mosca

Parity network synthesis in trapped ions

- Working with fan-in/out CNOT gates:



- Synthesis is :
  - « cheap » : depends on some property of the interaction graph (for 2-parities)
  - « complicated »: this property is hard to quantify (related to smallest Vertex Cover)

Atos

3 Connectivity and embedding schemes

First of all : 2-parities are universal

- Say we have « large » parities $C\,\sigma_z^{i_1}\sigma_z^{i_2}..\sigma_z^{i_k}$ in H_C
- We can « split » it into:

some ancilla

$$\underbrace{C\,\sigma_z^{i_1}\sigma_z^{i_2}..\sigma_z^{i_k}}_{k} \equiv \pm|C|\left(\underbrace{\sigma_z^{a}\sigma_z^{i_1}\dots\sigma_z^{i_l}}_{l+1} - \mathrm{sign(C)}\underbrace{\sigma_z^{a}\sigma_z^{i_{l+1}}\dots\sigma_z^{i_k}}_{k-l+1}\right)$$

- 3-parities can also be decomposed:

$$\sigma_z^1\sigma_z^2\sigma_z^3 \equiv \sigma_z^1\sigma_z^2 + \sigma_z^2\sigma_z^3 + \sigma_z^1\sigma_z^3 + \sum\left[\sigma_z^i\sigma_z^a - \sigma_z^i\right] - 2\sigma_z^a$$

Embedding (2) Ising models

$$H = \sum h_i \sigma_z^i + \sum J_{i,j} \sigma_z^i \sigma_z^j$$

spins (±1)

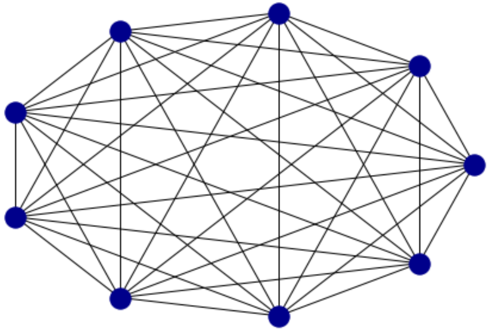
- Correspondance with the QUBO framework:

$$C(z) = \sum c_i z_i - \sum J_{i,j} z_i z_j$$

boolean variables (0,1)

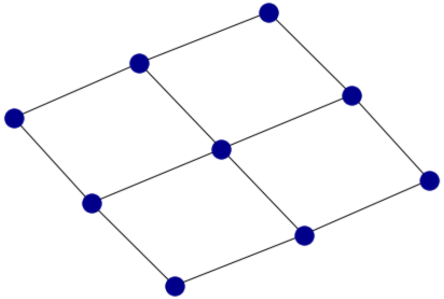
Embedding (2) Ising models

$$H = \sum h_i \sigma_z^i + \sum J_{i,j} \sigma_z^i \sigma_z^j$$

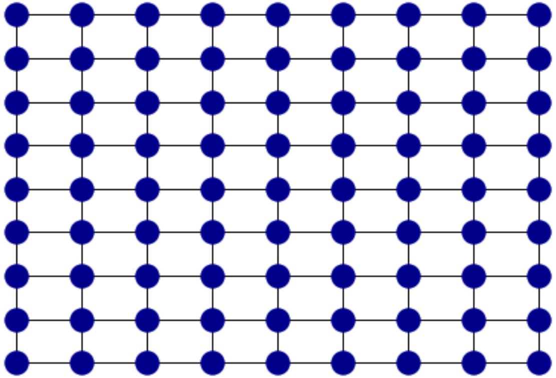


Problem connectivity

vs.



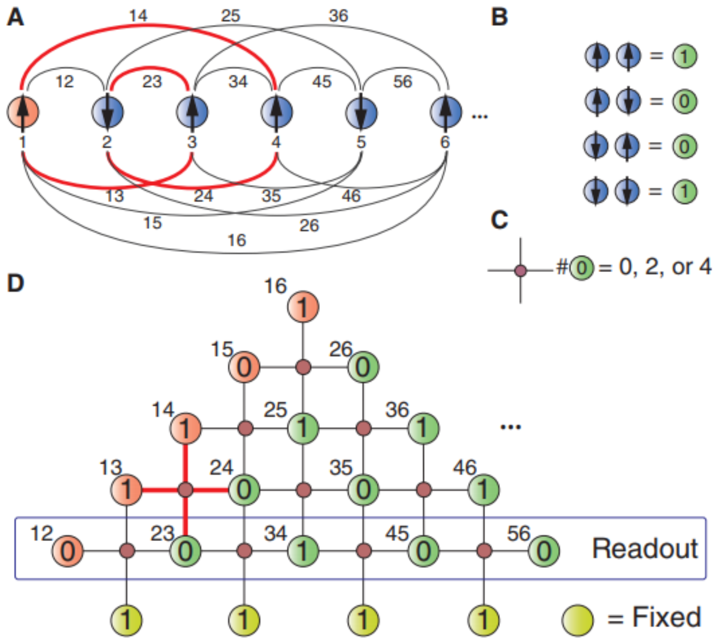
Hardwa re graph



LHZ Scheme : the physicists approach

$$\sum h_{x,i}\sigma_x^i + \sum h_{z,i}\sigma_z^i + \sum J_{i,j}\sigma_z^i\sigma_z^j$$

$$A\sum \sigma_x^i + B\sum J_i\sigma_z^i + C\sum \sigma_z^{(l,n)}\sigma_z^{(l,s)}\sigma_z^{(l,e)}\sigma_z^{(l,w)}$$



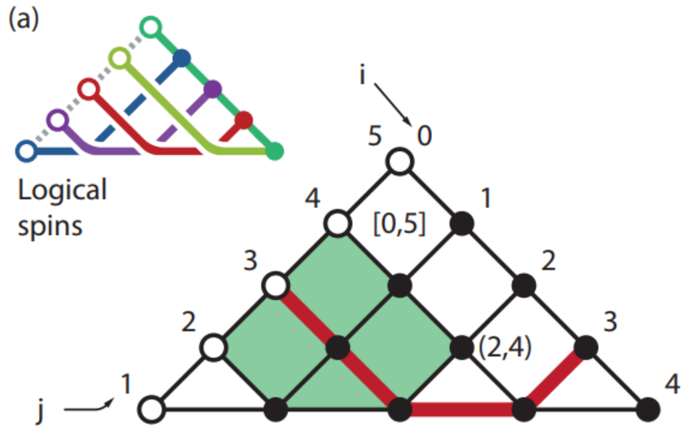
LHZ, A quantum annealing architecture with all-to-all connectivity from local interactions

LHZ Scheme : using stabilisers

- LHZ re-expressed as:
  - ➢ local drives
  - ➢ + stabilisers constraints

- Allow more general embedding schemes

- Still constrained on a grid-like structure



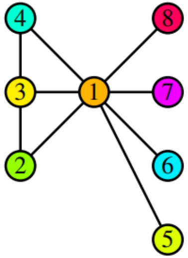
RBL, Stabilisers as a design tool for new forms of Lechnes-Hauke-Zoller Annealer

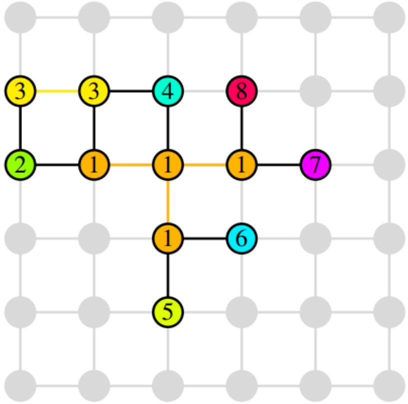
The minor embedding approach
(by D-wave)

- Embed the interaction graph as a minor of the hardware graph

- 1 spin == a subtree of the hardware graph

- Smallest hardware graph that:
  – can embed any graph of size $\leq n$
  – has a limited connectivity (grid-like)
  – has « simple » embeddings (easy to compute)

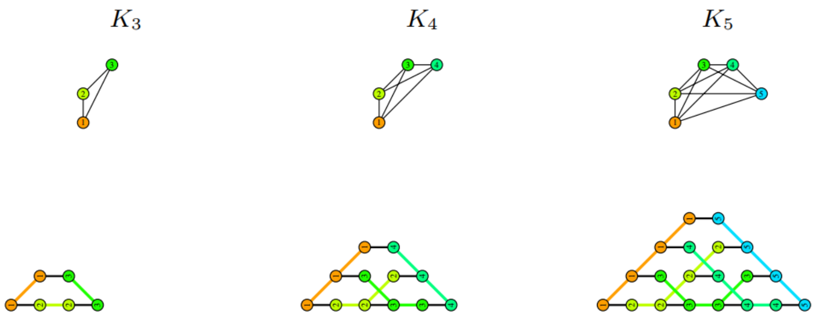




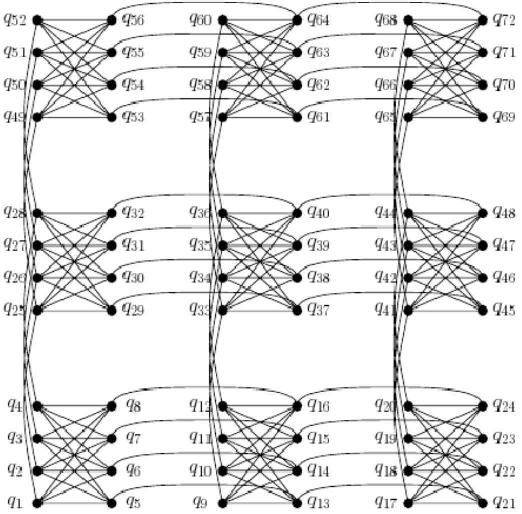
- Possible answer: a triad graph, or a chimera-graph

Vicky Choi, Minor-Embedding in Adiabatic Quantum Computation (part I & II)

The minor embedding approach
(by D-wave)



Triad graph



Chimera graph

In general : same overhead of $O(n^2)$

Vicky Choi, Minor-Embedding in Adiabatic Quantum Computation (part I & II)

4

Combinatorial optimization inside the QLM

Variational Quantum Eigensolver
for combinatorial optimization

- Applications
- High level library
- ad hoc synthesis
- pyAQASM
- quantum processor

```
my_problem = Problem()

var1 = my_problem.new_var()
var2 = my_problem.new_var()
var3 = my_problem.new_var()

my_problem.add_clause(var1 ^ var2)
my_problem.add_clause(var2 | var3 & var1)
my_problem.add_clause(~var1)
```

```
circuit = my_problem.generate_circuit([0.1, 1.6, 2.4], [1., 2., 3.],
                                      optimization_strategy="coloring")
```

```
val = my_problem.evaluate([0.1, 1.6, 2.4], [1., 2., 3.],
                          qpu=qat.linalg.get_qpu_server(),
                          optimization_strategy="parity")
```

Todo:
- finer compilation/synthesis using HW specs
- various encoding schemes
- classical optimization part?

Thank you

For more information please contact:

simon.martiel@atos.net