Quantum Annealing

PLDI Tutorial – June 15, 2020

NC STATE UNIVERSITY
Electrical & Computer Engineering
Computer Science
@NCStateECE
@cscncsu

Quantum Annealer

Designed to solve optimization problems

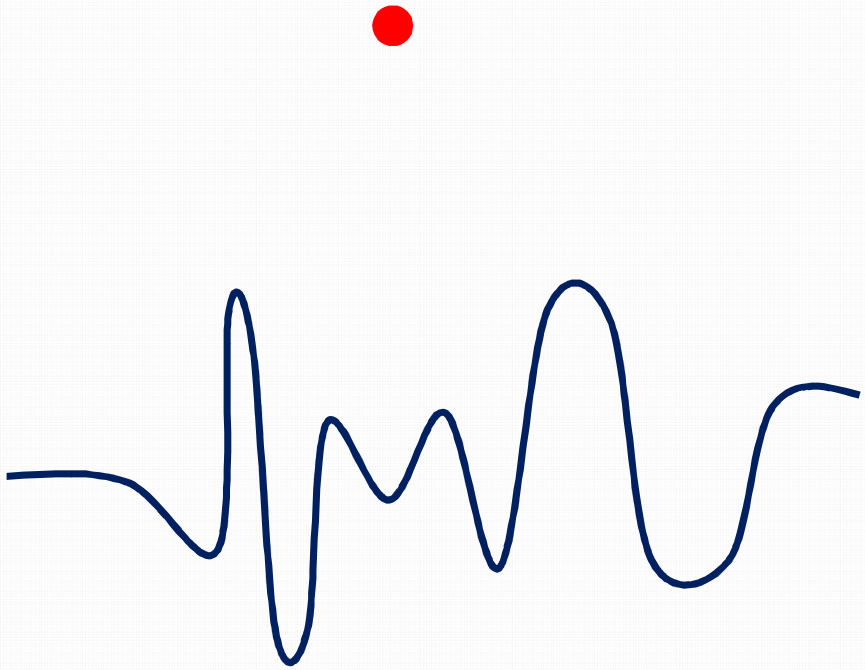
Not a general-purpose quantum computer,
but related to adiabatic quantum computing.

Photo by Capturing the human heart. on Unsplash

Classical: Simulated Annealing

Find location of minimum value in an energy landscape

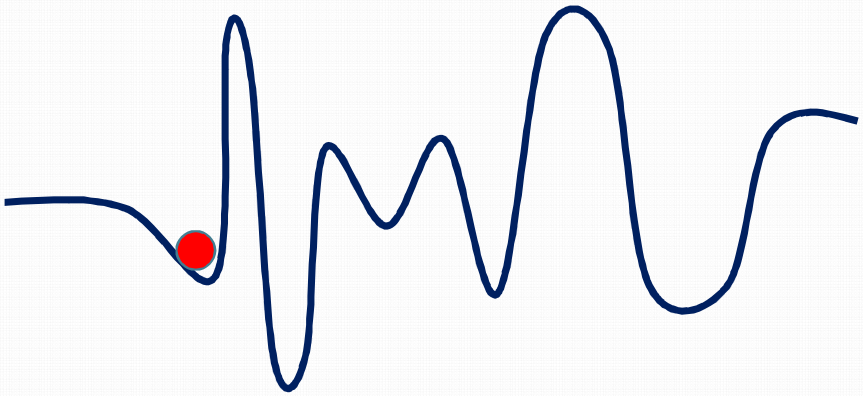
Roll down hills, but randomly allow bounce to escape local minimum.



Quantum Annealing

Quantum Annealing uses a transverse field to induce quantum tunneling through energy barriers

Depends on width of barrier, as well as height -- may be more likely to find global minimum



Quantum Annealing

Start in a state where all qubits are in superposition.

This is known to be a minimum-energy state.

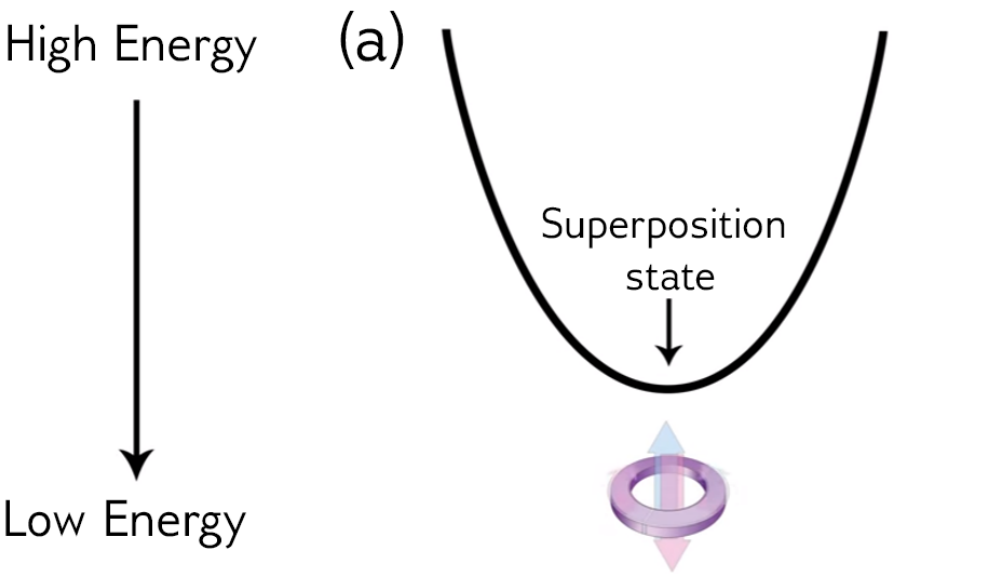


Figure from https://docs.dwavesys.com/docs/latest/c_gs_2.html

Quantum Annealing

Annealing process raises energy barrier. In this case, there are two minimum energy levels. Qubit becomes 0 (50%) or 1 (50%).

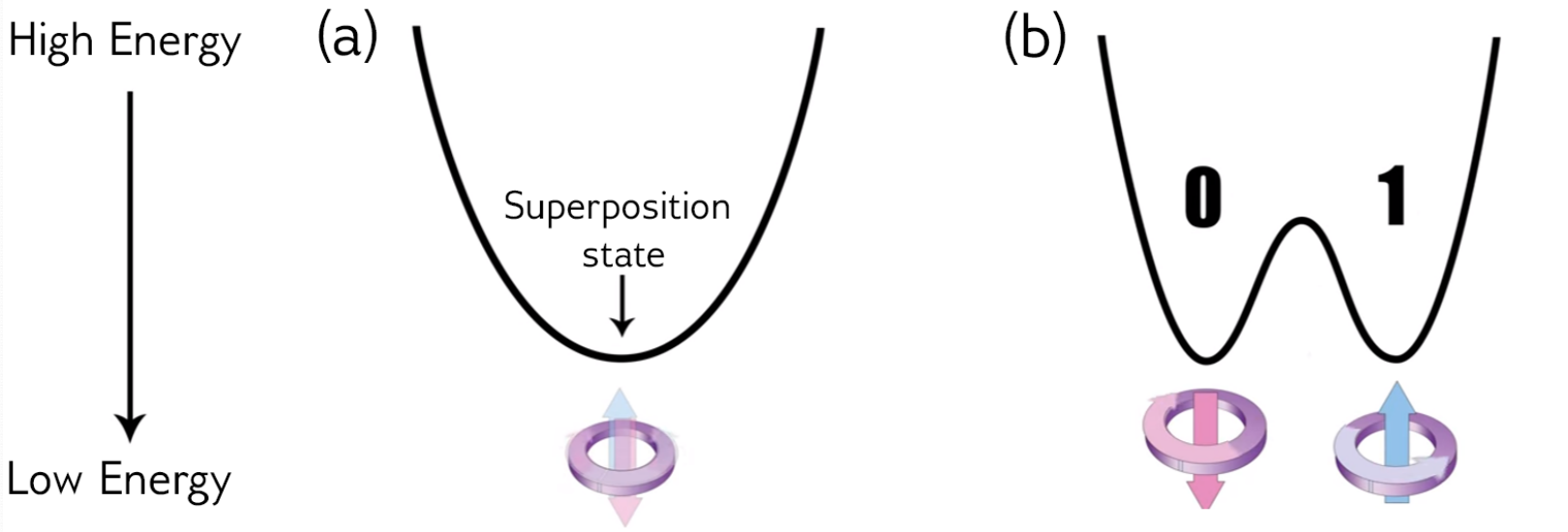


Figure from https://docs.dwavesys.com/docs/latest/c_gs_2.html

Quantum Annealing: Bias

Can influence energy levels by applying magnetic field.
Changes probability of 0 and 1.

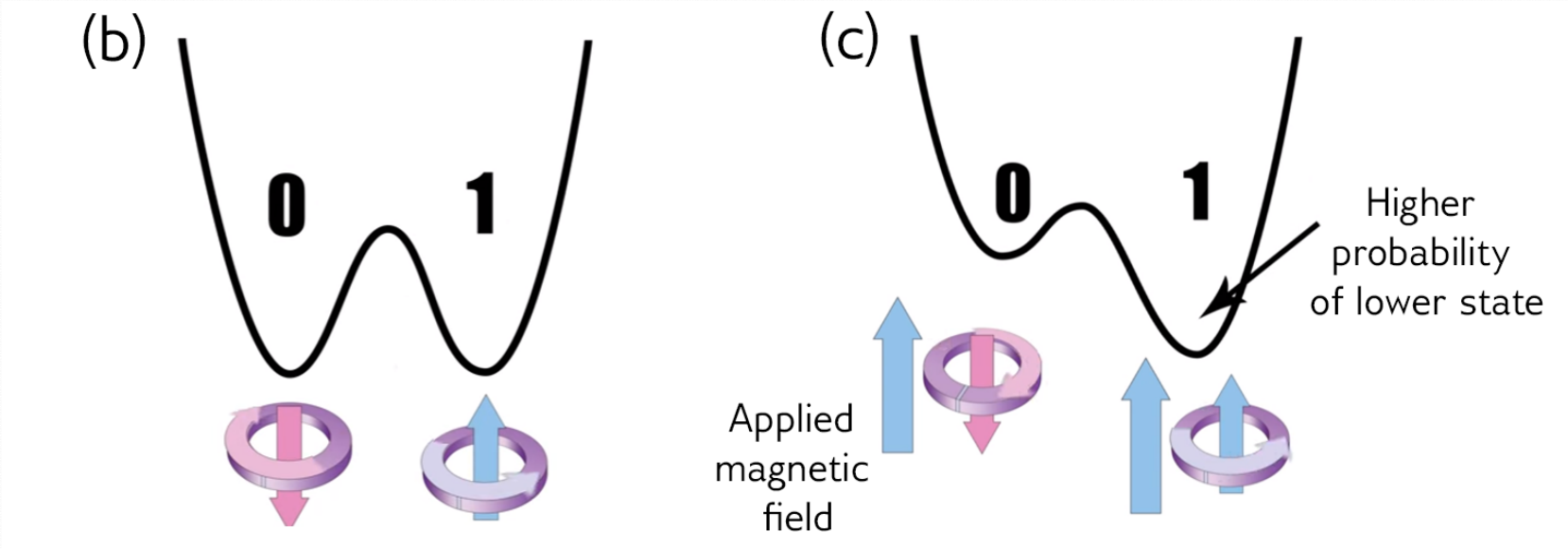


Figure from https://docs.dwavesys.com/docs/latest/c_gs_2.html

Quantum Annealing: Coupler

In addition to bias, couplers between qubits allow them to influence each other. Coupler increases probability that two qubits will end in the same (or different) states.

Together, couplers and bias generate an energy landscape, where minimum energy represents solution to the desired problem.

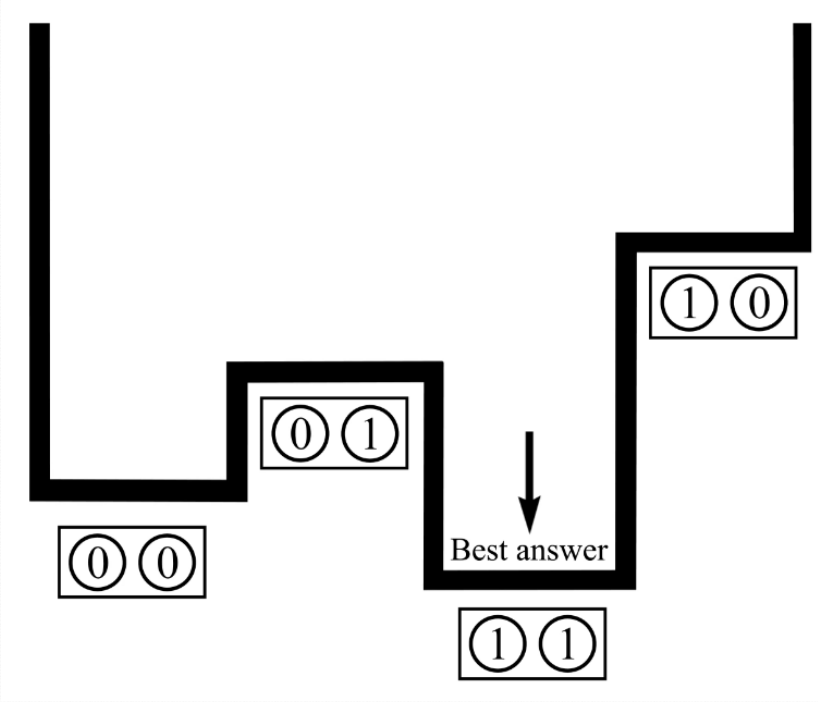


Figure from https://docs.dwavesys.com/docs/latest/c_gs_2.html

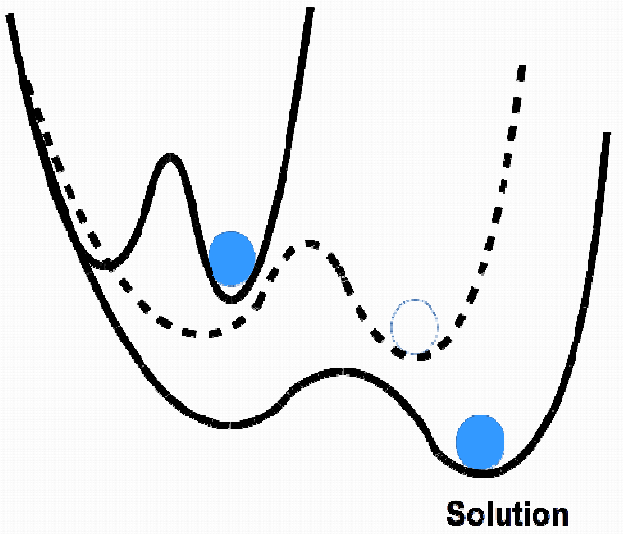
Adiabatic Evolution of State

Start in a known minimum state.

Change energy landscape (bias and couplers) to match problem to be solved.

If change is slow enough, system should remain in minimum state throughout.

Tunneling allows movement as barriers and minima change.



Adiabatic evolution

Figure from https://medium.com/@quantum_wa/quantum-annealing-cdb129e96601

Mathematical Description

Hamiltonian is a function that maps eigenstates to energy.

$$H = -\frac{A(s)}{2}\left(\sum_i \sigma_i^x\right) + \frac{B(s)}{2}\left(\sum_i h_i \sigma_i^z + \sum_{i>j} J_{i,j}\,\sigma_i^z \sigma_j^z\right)$$

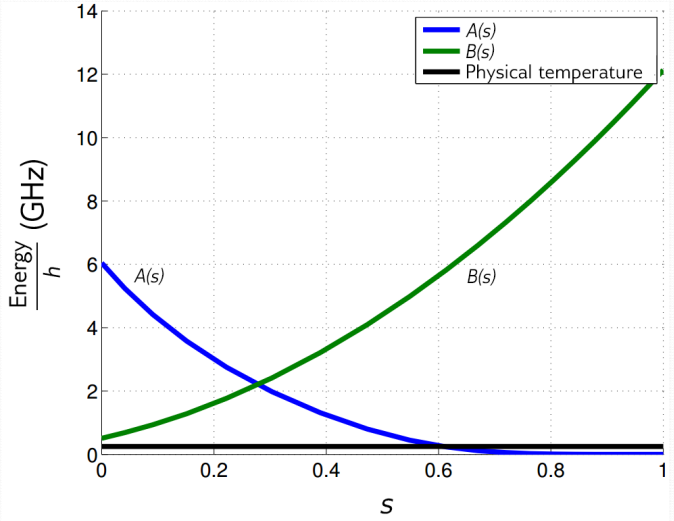
qubit = {+1, -1}

bias

coupler

Initial (tunneling) Hamiltonian

Final (problem) Hamiltonian



Over time, increase decrease A(s) to 0, increase B(s).

Adiabatic Quantum Computation

Annealing is a form of adiabatic quantum optimization.

A more general formulation is adiabatic quantum computation.

$$H_{ZZXX} = \left(\sum_i h_i \, \sigma_i^z + \sum_{i>j} J_{i,j} \, \sigma_i^z \sigma_j^z \right) + \left(\sum_i \Delta_i \, \sigma_i^x + \sum_{i>j} K_{i,j} \, \sigma_i^x \sigma_j^x \right)$$

Shown to be equivalent to gate-model quantum computation.

But this is not what the D-Wave system implements.

Problem Formulation

Ising model (2-local)

$$E_{\text{ising}} = \sum_i h_i s_i + \sum_i \sum_{j>i} J_{i,j} s_i s_j$$

s is spin up (↑ = +1) or
spin down (↓ = -1)

QUBO

(Quadratic Unconstrained Binary Opt)

$$f(x) = \sum_i Q_{i,i} x_i + \sum_i \sum_{j>i} Q_{i,j} x_i x_j$$

x is binary vector

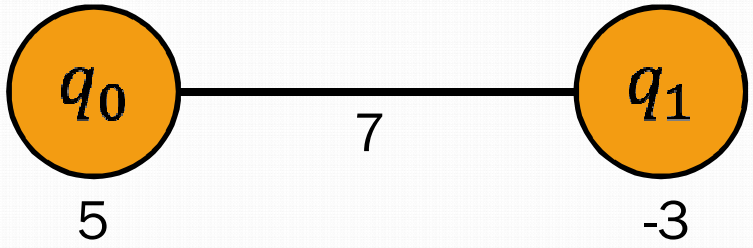
$x^T Q x$, minimize for $x \in \{0,1\}^n$

q is qubit (0 or 1)

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

Problem Formulation

Quadratic problem can be viewed as a graph, with qubits as nodes and couplers as edges



$$H = 5q_0 + 7q_0q_1 - 3q_1$$

Problem Formulation

Ising model (2-local)

$$E_{\text{ising}} = \sum_i h_i s_i + \sum_i \sum_{j>i} J_{i,j} s_i s_j$$

QUBO
(Quadratic Unconstrained Binary Opt)

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

Program = set of biases and couplings
Outcome = vector of qubit values that minimize the objective function

D-Wave 2000Q QPU

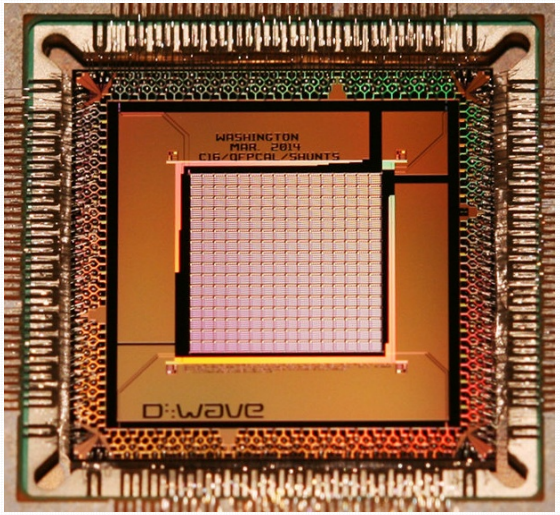
Up to 2048 qubits, 6016 couplers

Over 128,000 Josephson junctions on a single chip

Range of values: $h \in [-1, +1], J \in [-2, +2]$

Limited connectivity: Chimera graph

5000-qubit system announced in 2019



qubit = {+1, -1}

bias

coupler

$$H = -\frac{A(s)}{2}\left(\sum_i \sigma_i^x\right) + \frac{B(s)}{2}\left(\sum_i h_i \sigma_i^z + \sum_{i>j} J_{i,j} \sigma_i^z \sigma_j^z\right)$$

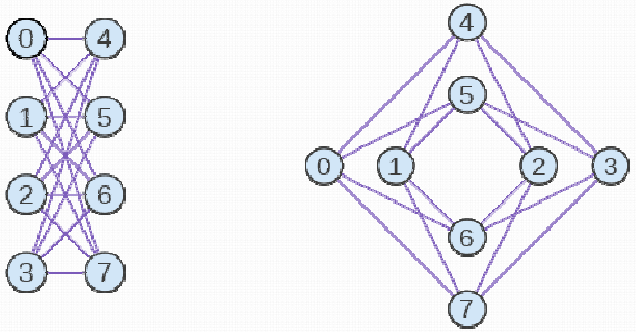
Initial (tunneling) Hamiltonian

Final (problem) Hamiltonian

Building Block: Unit Cell

Logical Topology

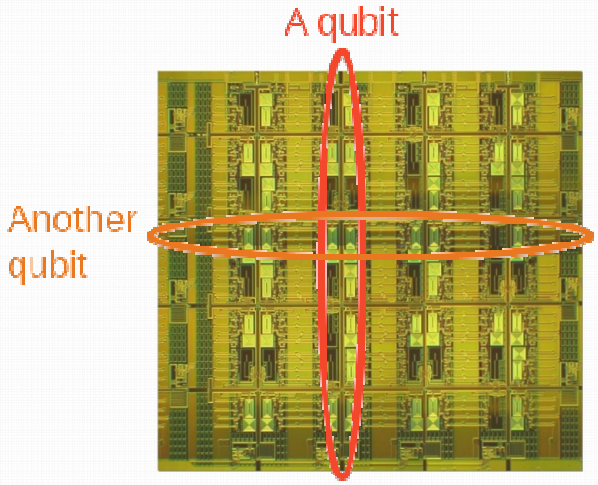
8 qubits connected in bipartite graph



Physical Implementation

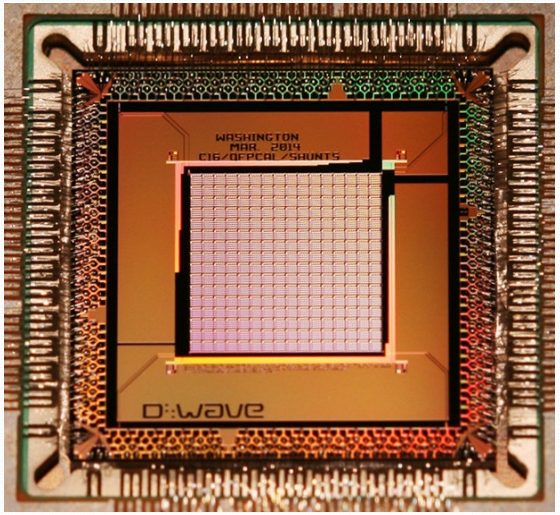
RF SQUIDs

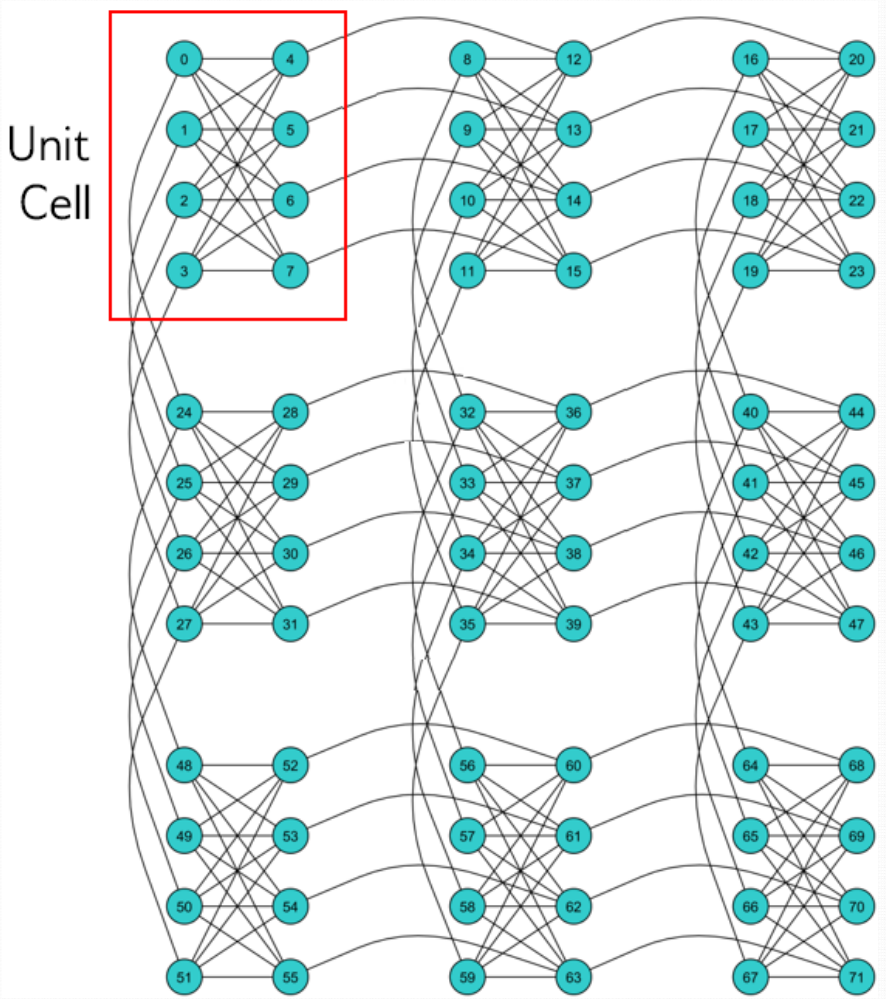
Coupling happens at intersection of qubit circuits



Chimera Graph

16x16 network of unit cells
4 connect east-west
4 connect north-south





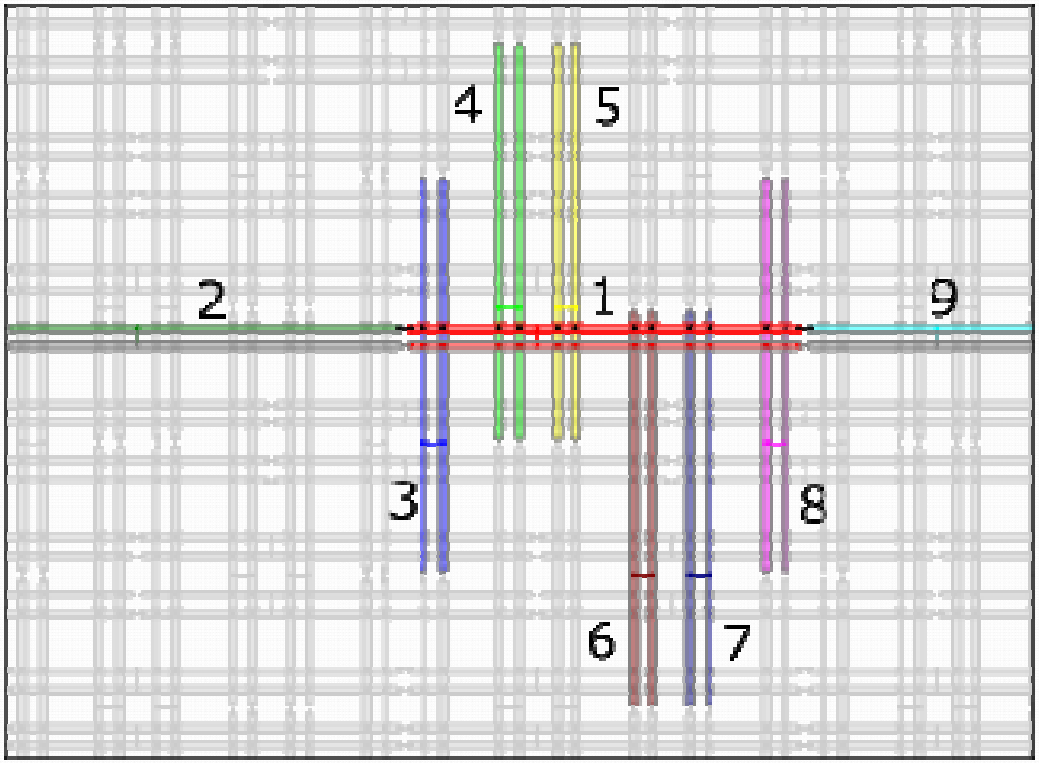
https://docs.dwavesys.com/docs/latest/c_gs_4.html

Pegasus Graph

Next-generation interconnection

Each qubit has 12 internal couplers
(compared to 4 in Chimera) plus
2 or 3 additional couplers.

Degree = 15
(compared to 5 in Chimera)



https://docs.ocean.dwavesys.com/projects/dwave-networkx/en/latest/intro.html

Errors

Interaction with environment

- RF shielded, dilution refrigerator -- but not perfectly isolated.

Annealing schedule

- If too fast, or energy levels too “close,” can jump to a non-minimal state.
- Latest systems provide some user control of annealing schedule.

Run multiple shots (500-1000) and analyze results

Runtime

Setup / read time dominate annealing time

- *Before:* reset + programming + post-programming thermalization
- *After:* readout
- Larger by orders of magnitude

Advantage?

- Google reported 108x speed for D-Wave-friendly problem
- (but compared to non-optimal classical algorithm on single CPU)

D-Wave Programming

https://docs.dwavesys.com/docs/latest/c_gs_1.html

Tools

To work on the exercises in this section:

- Download Quantum Apprentice spreadsheet from tutorial web site: https://sites.google.com/ncsu.edu/qc-tutorial/info/
- Go to https://cloud.dwavesys.com/leap/resources/learning-docs/
  - Will need to login to Leap, or sign up for an account
- Click on “Interactive Examples,” and then “Go to Jupyter Explorer View”
- Open `leap/demos/sandbox/01-sandbox.ipynb`
  - Will be pasting code from tutorial Python files into this notebook

Weight: Qubit bias

The weight (bias) influences the final state of each qubit.

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

$a_0 = -5$

q_0	$a_0 q_0$
0	0
1	-5

Negative weight:
"I want qubit to be 1."

$a_0 = 0$

q_0	$a_0 q_0$
0	0
1	0

Zero weight:
"I don't care."

$a_0 = 5$

q_0	$a_0 q_0$
0	0
1	5

Positive weight:
"I want qubit to be 0."

Strength: Qubit pair coupling

The strength (coupling) influences the qubits as a pair.

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

$b_{0,1} = -5$

q_0	q_1	$b_{0,1}q_0q_1$
0	0	0
0	1	0
1	0	0
1	1	-5

Negative strength:
"I want both qubits to be true."

$b_{0,1} = 0$

q_0	q_1	$b_{0,1}q_0q_1$
0	0	0
0	1	0
1	0	0
1	1	0

Zero strength:
"I don't care."

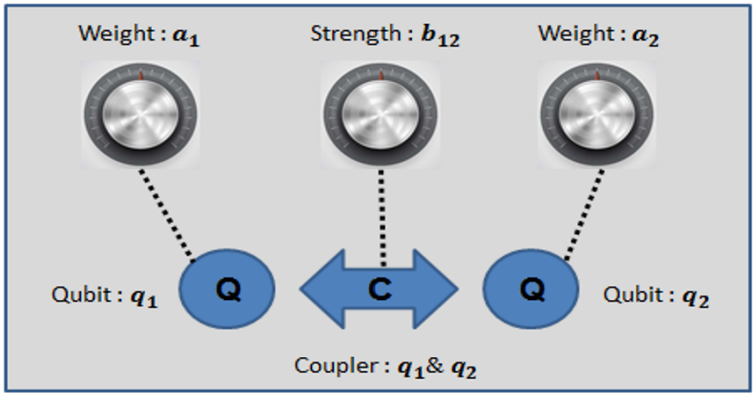
$b_{0,1} = 5$

q_0	q_1	$b_{0,1}q_0q_1$
0	0	0
0	1	0
1	0	0
1	1	5

Positive strength:
"I want neither qubit to be true."

Quantum Apprentice

Programming Model: Two qubits



Objective : $O(a_1, a_2, b_{12}; q_1, q_2) = a_1 q_1 + a_2 q_2 + b_{12} q_1 q_2$

QMI (Quantum Machine Instruction) :

a_1	a_2	b_{12}

q1	*q2*	Objective
0	0	0
0	1	0
1	0	-1
1	1	**-2**

a_1	a_2	b_{12}
-1	**0**	**-1**

QA Exercises: Two Qubits

Use QA to find weights and strengths to implement the following relationships

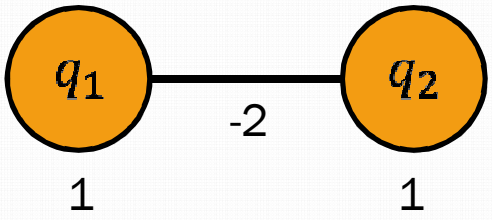
- equal: q_1 and q_2 are the same
- not-equal: q_1 and q_2 are different
- implies: if q_1 is true, q_2 is true

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

Solutions

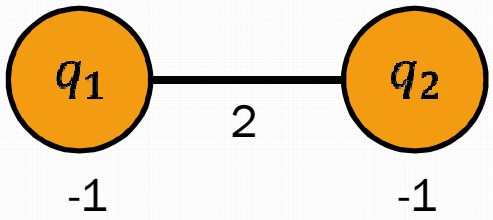
q_1	q_2	$q_1 = q_2$
0	0	0
0	1	1
1	0	1
1	1	0

a_1	a_2	$b_{1,2}$
1	1	-2



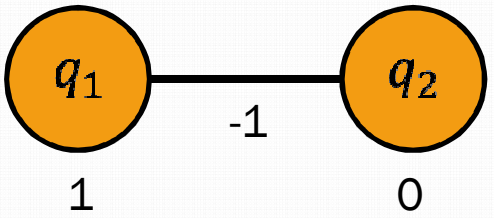
q_1	q_2	$q_1 \neq q_2$
0	0	0
0	1	-1
1	0	-1
1	1	0

a_1	a_2	$b_{1,2}$
-1	-1	2



q_1	q_2	$q_1 \Rightarrow q_2$
0	0	0
0	1	0
1	0	1
1	1	0

a_1	a_2	$b_{1,2}$
1	0	-1



Running on a Simulator

```
from dimod import ExactSolver

sampler = ExactSolver()

Q = {('q1', 'q1'): 1, ('q1', 'q2'): -2, ('q2', 'q2'): 1}    # q1 = q2

response = sampler.sample_qubo(Q)

for datum in response.data(['sample', 'energy']):
       print(datum.sample, "Energy: ", datum.energy)
```

```
{'q1': 0, 'q2': 0} Energy: 0.0
{'q1': 1, 'q2': 1} Energy: 0.0
{'q1': 1, 'q2': 0} Energy: 1.0
{'q1': 0, 'q2': 1} Energy: 1.0
```

dwave_2qubit_sim.py

Running on Hardware

```
from dwave.system import DWaveSampler,EmbeddingComposite

sampler = EmbeddingComposite(DWaveSampler())

Q = {('q1', 'q1'): 1, ('q1', 'q2'): -2, ('q2', 'q2'): 1}    # q1 = q2

response = sampler.sample_qubo(Q, num_reads=5000)

for datum in response.data(['sample', 'energy', 'num_occurrences']):
    print(datum.sample, "Energy: ", datum.energy, "Occurrences: ",
          datum.num_occurrences)
```

```
{'q1': 0, 'q2': 0} Energy: 0.0 Occurrences: 2593
{'q1': 1, 'q2': 1} Energy: 0.0 Occurrences: 2406
{'q1': 1, 'q2': 0} Energy: 1.0 Occurrences: 1
```

dwave_2qubit_hw.py

3-Qubit Objective: OR

$$q_1 \vee q_2 = q_3$$

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

q_1	q_2	q_3	Objective
0	0	0	0
0	0	1	a_3
0	1	0	a_2
0	1	1	$a_2 + a_3 + b_{2,3}$
1	0	0	a_1
1	0	1	$a_1 + a_3 + b_{1,3}$
1	1	0	$a_1 + a_2 + b_{1,2}$
1	1	1	$a_1 + a_2 + a_3 + b_{1,2} + b_{1,3} + b_{2,3}$

a_1	a_2	a_3

$b_{1,2}$	$b_{1,3}$	$b_{2,3}$

3-Qubit Objective: OR

$$q_1 \vee q_2 = q_3$$

$$E_{\text{qubo}} = \sum_i a_i q_i + \sum_i \sum_{j>i} b_{i,j} q_i q_j$$

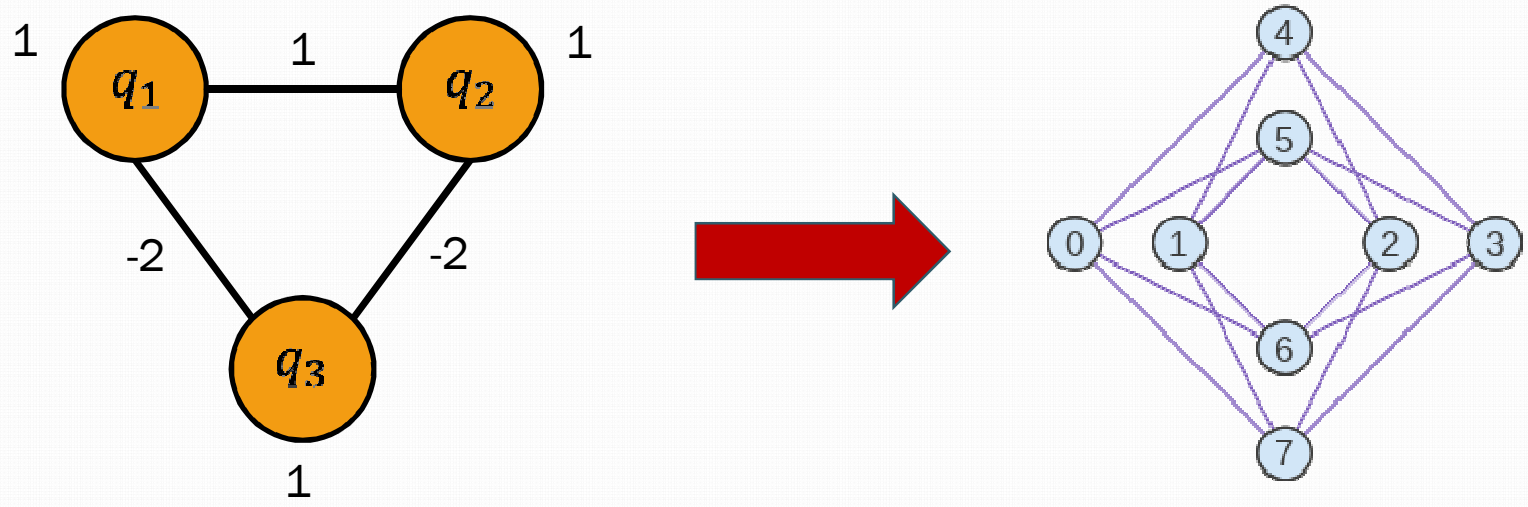
q_1	q_2	q_3	Objective	Value
0	0	0	0	0
0	0	1	a_3	1
0	1	0	a_2	1
0	1	1	$a_2 + a_3 + b_{2,3}$	0
1	0	0	a_1	1
1	0	1	$a_1 + a_3 + b_{1,3}$	0
1	1	0	$a_1 + a_2 + b_{1,2}$	3
1	1	1	$a_1 + a_2 + a_3 + b_{1,2} + b_{1,3} + b_{2,3}$	0

a_1	a_2	a_3
1	1	1

$b_{1,2}$	$b_{1,3}$	$b_{2,3}$
1	-2	-2

Minor Embedding

In order to run on the D-Wave system, must assign logical variables to physical qubits. D-Wave calls this a minor embedding.

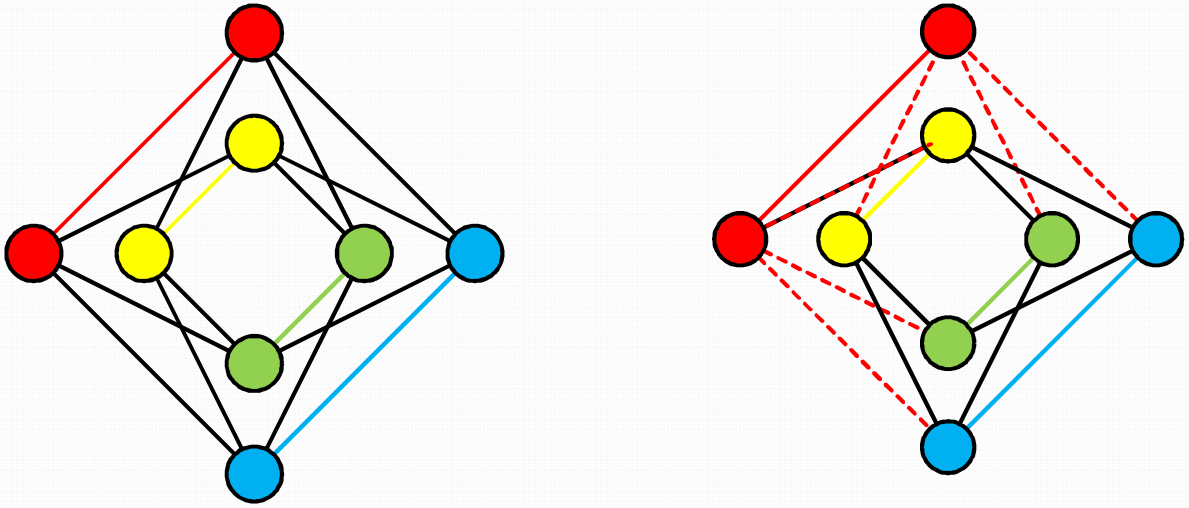


Problem: no completely-connected group of three qubits!

Chaining

Solution: Use two physical qubits to represent one logical qubit.

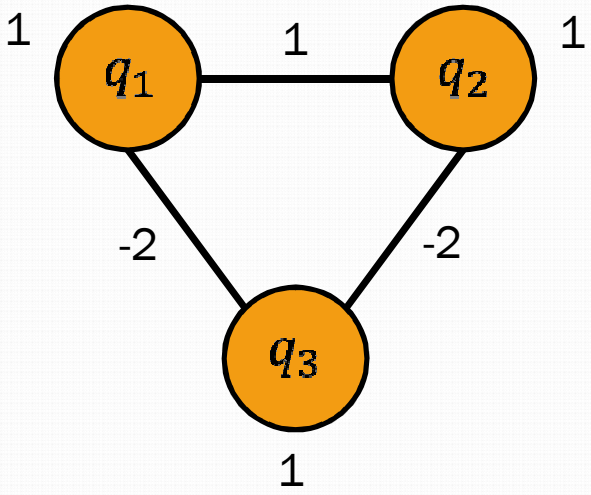
This is called chaining.



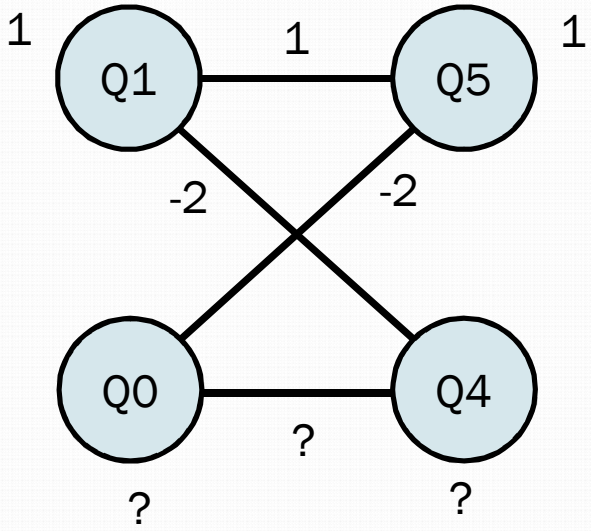
While each red qubit only has four neighbors, the pair of red qubits has six neighbors.

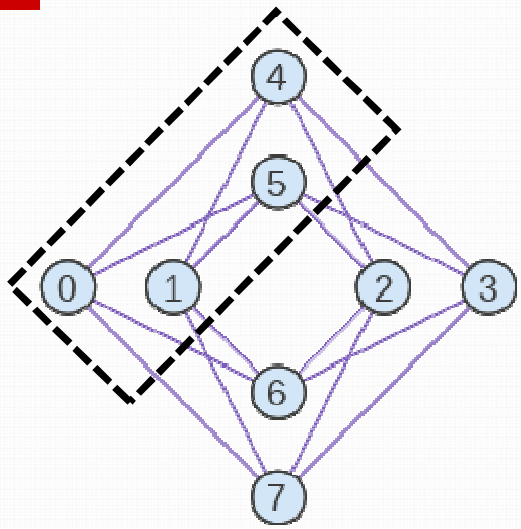
Chaining

Logical qubits



Physical qubits



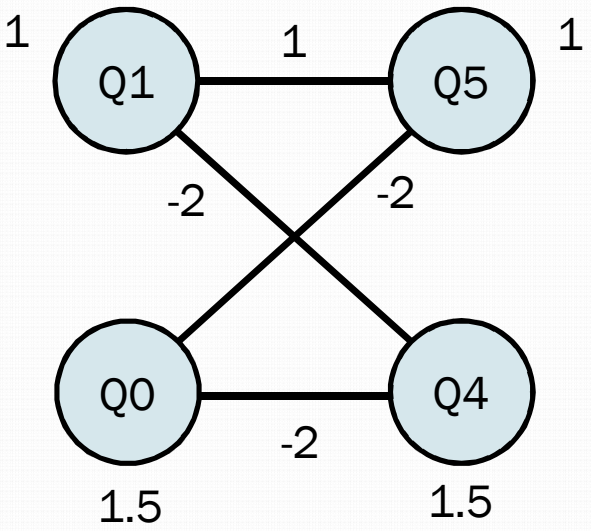


What do we do about weights and strength?

Chaining

When splitting a logical qubit into a chain of N physical qubits:

1. Evenly split the weight between the physical qubits. (Q0 = 0.5, Q4 = 0.5)
2. Add a strong negative strength to the coupling. (-2 between Q0 and Q4)
3. Negate the coupling strength and divide among the physical qubits. (Add +1 to both Q0 and Q4.)



Automatic Embedding

```
from dwave.system import DWaveSampler, EmbeddingComposite

sampler = EmbeddingComposite(DWaveSampler())

Q = {('q1', 'q1'): 1, ('q2', 'q2'): 1, ('q3', 'q3'): 1,
     ('q1', 'q2'): 1, ('q1', 'q3'): -2, ('q2','q3'): -2}   # Boolean OR gate

response = sampler.sample_qubo(Q, num_reads=500)

for datum in response.data(['sample', 'energy', 'num_occurrences']):
    print(datum.sample, "Energy: ", datum.energy,
          "Occurrences: ", datum.num_occurrences)
```

dwave_3qubit_auto.py

Manual Embedding

```
from dwave.system import DWaveSampler, FixedEmbeddingComposite

embedding = {'q1': {1}, 'q2': {5}, 'q3': {0,4}}
sampler = FixedEmbeddingComposite(DWaveSampler(), embedding)

Q = {('q1', 'q1'): 1, ('q2', 'q2'): 1, ('q3', 'q3'): 1,
     ('q1', 'q2'): 1, ('q1', 'q3'): -2, ('q2','q3'): -2}   # Boolean OR gate

response = sampler.sample_qubo(Q, num_reads=500)

for datum in response.data(['sample', 'energy', 'num_occurrences']):
    print(datum.sample, "Energy: ", datum.energy,
          "Occurrences: ", datum.num_occurrences)
```

dwave_3qubit_fixed.py

Manual Explicit Embedding

```
from dwave.system import DWaveSampler, FixedEmbeddingComposite

embedding = {'q1': {1}, 'q2': {5}, 'q3': {0}, 'qq3': {4}}
sampler = FixedEmbeddingComposite(DWaveSampler(), embedding)

Q = {('q1', 'q1'): 1, ('q2', 'q2'): 1, ('q3', 'q3'): 1.5, ('qq3', 'qq3'): 1.5,
     ('q1', 'q2'): 1, ('q1', 'qq3'): -2, ('q2','q3'): -2, ('q3', 'qq3'): -2}

response = sampler.sample_qubo(Q, num_reads=500)

for datum in response.data(['sample', 'energy', 'num_occurrences']):
    print(datum.sample, "Energy: ", datum.energy,
          "Occurrences: ", datum.num_occurrences)
```

AND

$z = xy$

Consider truth table.

Pick a value T for true entries, and make other values > T.

$xy - 2xz - 2yz + 3z = 0$

x	y	z	Obj	
0	0	0	T	
0	0	1	> T	z
0	1	0	T	
0	1	1	> T	z, yz
1	0	0	T	
1	0	1	> T	z, xz
1	1	0	> T	xy
1	1	1	T	

Observations:

- Since (0,0,0) is a solution, T = 0.
- z weight must be > 0, xy weight must be >0
- Last row: Az + Bxy + Cxz + Dyz = 0, so weights for xz and yz must be negative.

AND

dwave_AND.py

```
from dimod import ExactSolver

sampler = ExactSolver()

Q = {('z', 'z'): 3, ('x', 'y'): 1, ('x', 'z'): -2, ('y', 'z'): -2}

response = sampler.sample_qubo(Q)

for datum in response.data(['sample', 'energy']):
    print(datum.sample, "Energy: ", datum.energy)
```

```
{'x': 0, 'y': 0, 'z': 0} Energy: 0.0
{'x': 1, 'y': 0, 'z': 0} Energy: 0.0
{'x': 0, 'y': 1, 'z': 0} Energy: 0.0
{'x': 1, 'y': 1, 'z': 1} Energy: 0.0
```

```
{'x': 1, 'y': 1, 'z': 0} Energy: 1.0
{'x': 0, 'y': 1, 'z': 1} Energy: 1.0
{'x': 1, 'y': 0, 'z': 1} Energy: 1.0
{'x': 0, 'y': 0, 'z': 1} Energy: 3.0
```

Half Adder

Circuit that adds two 1-bit values (x, y) and produces sum (s) and carry-out (c).

$$x + y = s + 2c$$
$$x + y - s - 2c = 0$$

Square it, to make a minimization problem.

$$(x + y - s - 2c)^2 = 0$$

$$x^2 + xy - xs - 2xc + yx + y^2 - ys - 2yc - sx - sy + s^2 + 2sc - 2cx - 2cy + 2cs + 4c^2 = 0$$

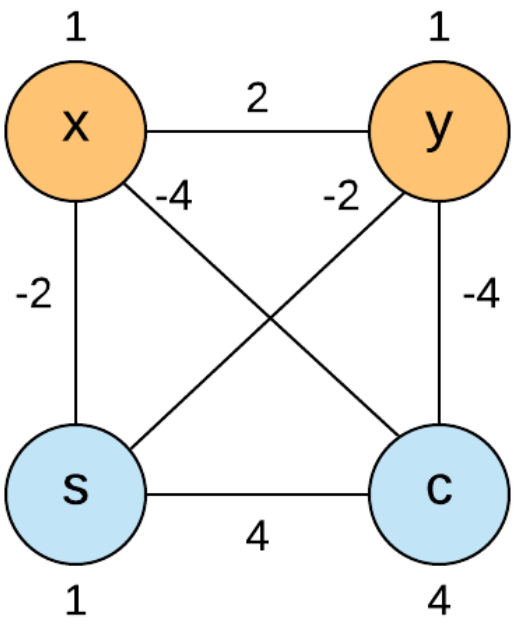
$$x^2 + y^2 + s^2 + 4c^2 + 2xy - 2xs - 4xc - 2ys - 4yc + 4sc = 0$$

NOTE: For Boolean variables, $x^2 = x$

x	y	s	c	Obj
0	0	0	0	T
0	0	0	1	> T
0	0	1	0	> T
0	0	1	1	> T
0	1	0	0	> T
0	1	0	1	> T
0	1	1	0	T
0	1	1	1	> T
1	0	0	0	> T
1	0	0	1	> T
1	0	1	0	T
1	0	1	1	> T
1	1	0	0	> T
1	1	0	1	T
1	1	1	0	> T
1	1	1	1	> T

Half Adder

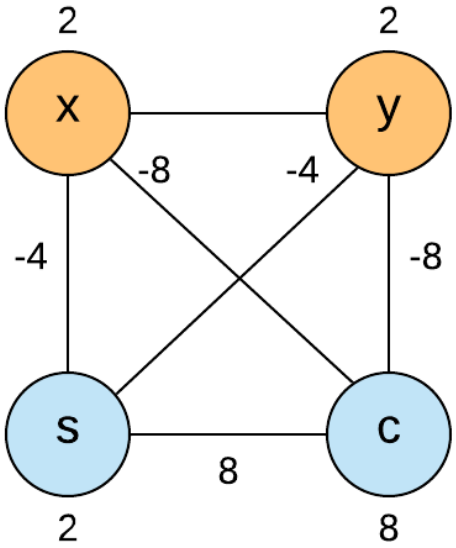
$$x + y + s + 4c + 2xy - 2xs - 4xc - 2ys - 4yc + 4sc = 0$$

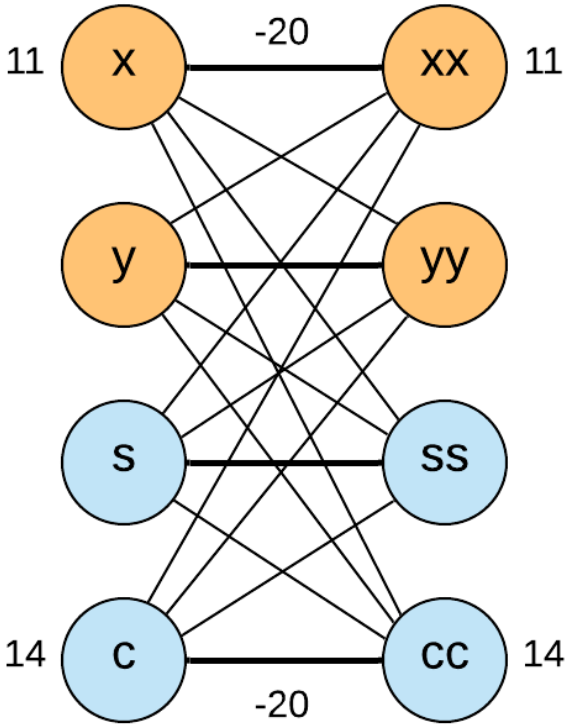


Fully-connected graph -- does not map directly to bipartite unit cell.

x	y	s	c	Obj
0	0	0	0	0
0	0	0	1	44
0	0	1	0	1
0	0	1	1	9
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	4
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	4
1	1	0	0	4
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Half Adder: Minor Embedding

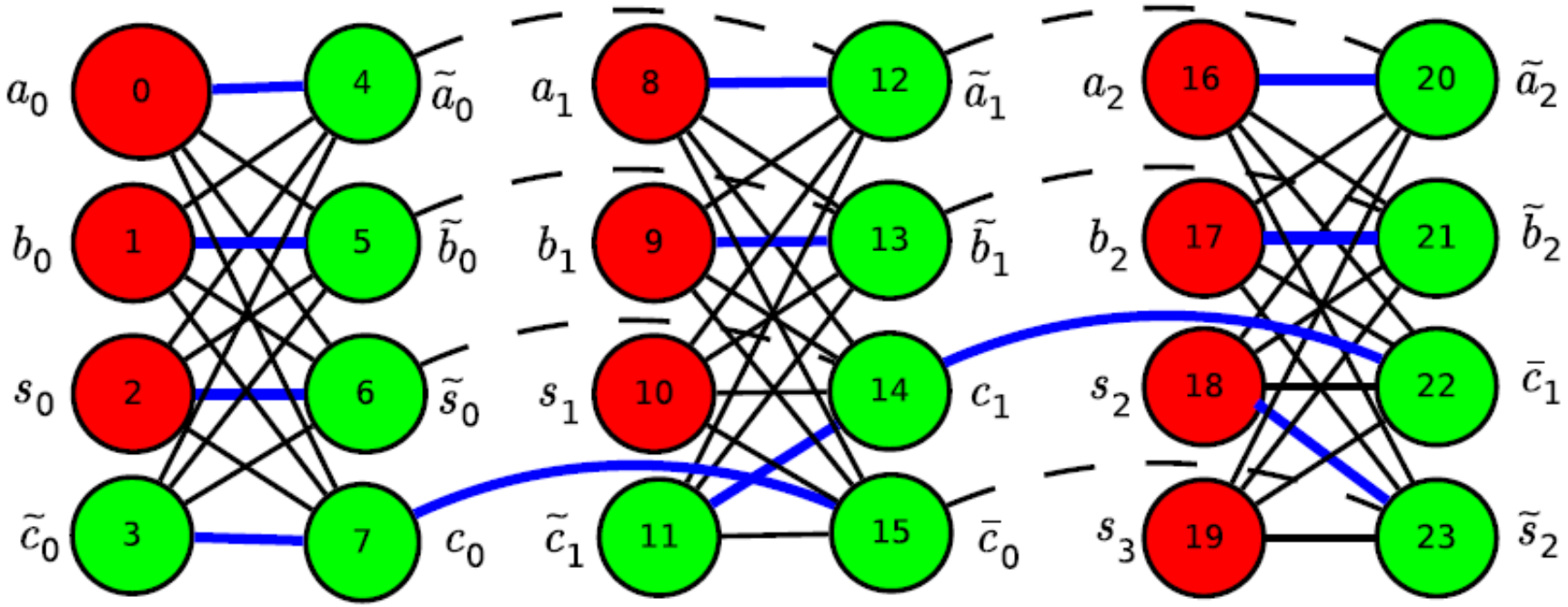




c	s	x	y	energy	num_oc.
0	**1**	**0**	**1**	**1**	**0.0**	**1170**
1	**0**	**0**	**0**	**0**	**0.0**	**1118**
2	**0**	**1**	**1**	**0**	**0.0**	**1260**
3	**0**	**1**	**0**	**1**	**0.0**	**1425**
4	0	1	0	0	1.0	6
5	0	0	0	1	1.0	4
6	1	0	1	0	1.0	1
7	1	1	1	1	1.0	3
8	1	0	0	1	1.0	6
9	0	1	1	1	1.0	4
10	0	0	1	0	1.0	3

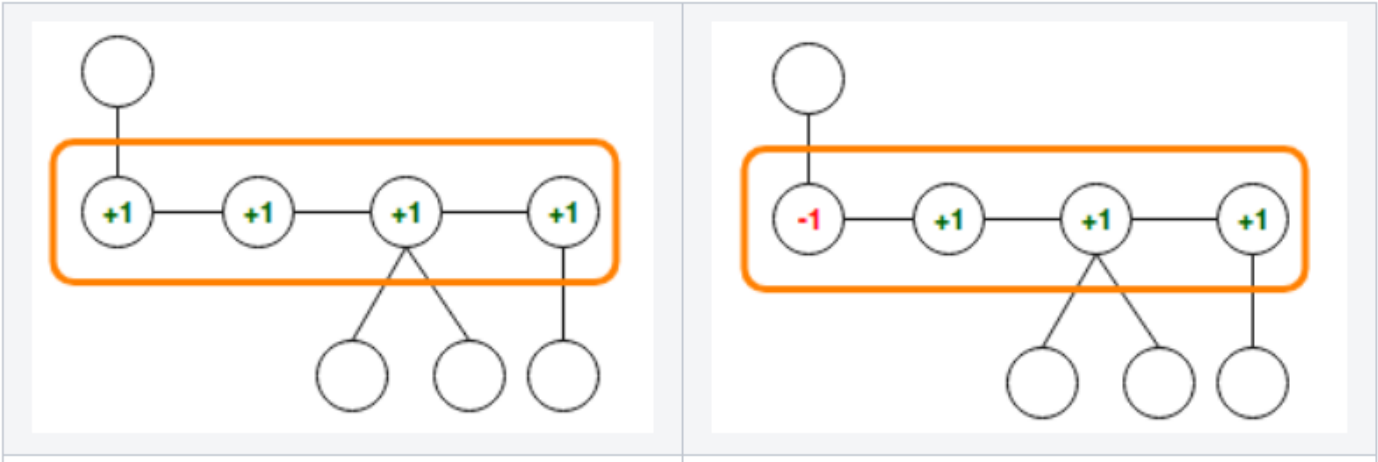
`dwave_halfadder.py`

3-bit Adder


a_0
0
4
$\widetilde{a}_0$
b_0
1
5
$\widetilde{b}_0$
s_0
2
6
$\widetilde{s}_0$
$\widetilde{c}_0$
3
7
c_0
a_1
8
12
$\widetilde{a}_1$
b_1
9
13
$\widetilde{b}_1$
s_1
10
14
c_1
$\widetilde{c}_1$
11
15
$\overline{c}_0$
a_2
16
20
$\widetilde{a}_2$
b_2
17
21
$\widetilde{b}_2$
s_2
18
22
$\overline{c}_1$
s_3
19
23
$\widetilde{s}_2$

Chain Break

When a variable is mapped to a chain, D-Wave performs an *unembedding* to report result using original variables.



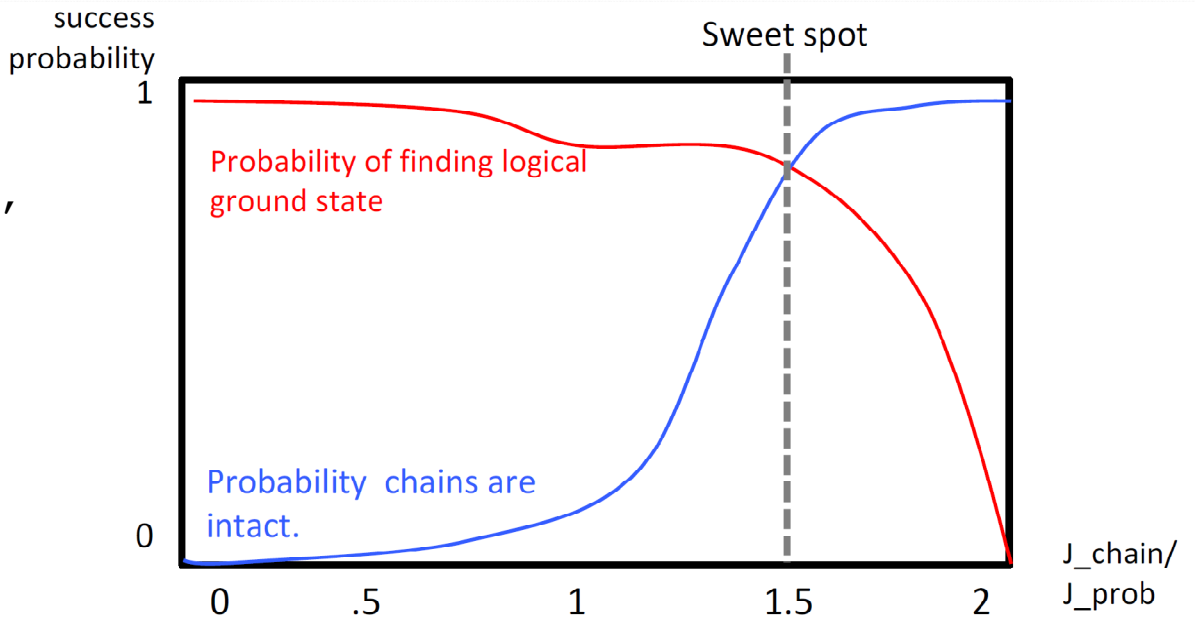
A chain break is when the qubits in a chain do not end up in the same state. The default method for resolving is majority-vote.

https://support.dwavesys.com/hc/en-us/community/posts/360016697094-What-is-a-chain-

Chain Strength

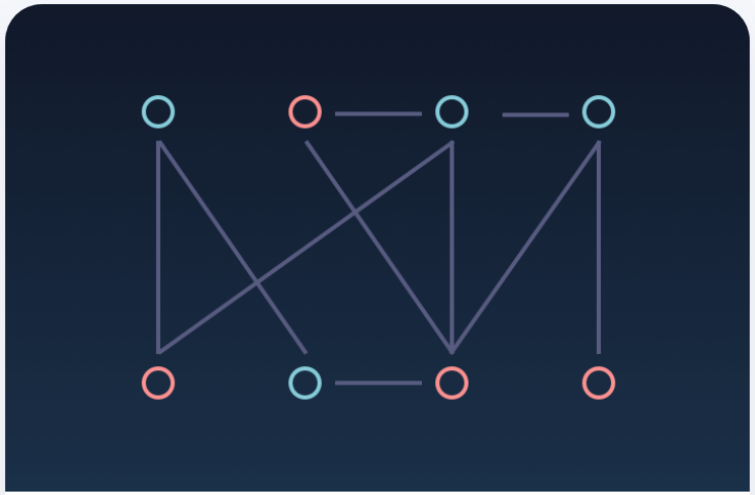
To minimize chain breaks, can adjust chain strength.

```
response =
  sampler.sample_qubo(Q, num_reads=500,
  chain_strength=5)
```



https://www.dwavesys.com/sites/default/files/2_Wed_Am_PerfTips.pdf

Social Network Analysis



DEMO 2

Social Network Analysis

Hard optimization problems can be a good fit for the D-Wave quantum computer.

https://cloud.dwavesys.com/leap/demos/socialnetwork