Intro to Quantum Computing

Part 4: D-Wave applications and quantum machine learning

Olivia Di Matteo

Quantum Information Science Associate, TRIUMF

Questions from last time

Q: Why is the initial $B(s)$ in the annealing schedule non-zero?

A: Has to do with how the biases are implemented in hardware. If you write out the coefficients in terms of physical parameters, you can find explicit expressions for A and B:

$$A(s) = \Delta_q(\Phi_{CCJJ}(s)) \quad (1)$$

$$B(s) = 2M_{AFM}|I_p(\Phi_{CCJJ}(s))|^2 \quad (2)$$

- M_{AFM} is the max mutual inductance generated by couplers between qubits
- Δ_q is the energy difference between the eigenstates with no applied flux
- I_p is the current flowing in the superconducting loop (and is related to Δ_q)
- $\Phi_{CCJJ}(s)$ is a flux applied to *all* qubits at point (s)

See: `https://docs.dwavesys.com/docs/latest/c_qpu_0.html#coupled-rf-squid-qubits`

Questions from last time

Q: How can we anneal on the order of μs with coherence times of ns?

A: A few points in a paper by D-Wave [1], and Scott Aaronson's blog[2] (Scott was formerly known as the "Chief D-Wave skeptic"):

- Scott explains how, on a visit to D-Wave, scientists told him that they think the system decoherences in the '*energy eigenbasis*', but not the computational basis (which is where it counts)
- The paper shows experimentally for a specific problem with small energy gap, that they achieve results compatible with a fully coherent system

[1] https://www.nature.com/articles/ncomms2920

[2] https://www.scottaaronson.com/blog/?p=954

Questions from last time

Q: What is a better way to describe tunnelling in the QA system?
A: I will try my best...

Questions from last time

Q: How do you turn factoring into a QUBO?
A: Integer encoding.

Let $N = pq$ be the number you want to factor. Then express

$$p = 1 \cdot p_0 + 2 \cdot p_1 + 4 \cdot p_2 + \cdots \quad (3)$$

$$q = 1 \cdot q_0 + 2 \cdot q_1 + 4 \cdot q_2 + \cdots \quad (4)$$

where all p_i, q_i are binary variables.
You can then write the QUBO

$$\min_{\mathbf{p},\mathbf{q}} (N - p \cdot q)^2 \quad (5)$$

You'll end up with a lot of terms with degree > 2, which you'll have to break down by adding additional variables.

See: `https://www.dwavesys.com/sites/default/files/14-1002A_B_tr_Boosting_integer_factorization_via_quantum_annealing_offsets.pdf`

Today

We will focus on more practical matters and applications:

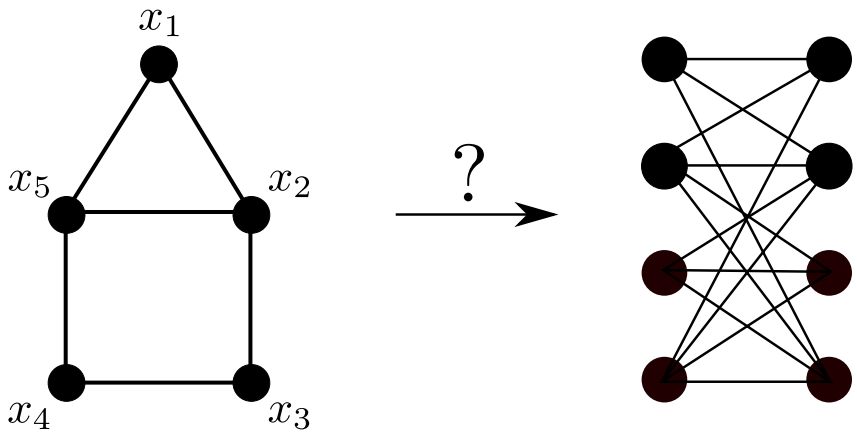
- Solving vertex cover on the QPU
- Graph minor embeddings and qbsolv
- Classifying Higgs decay signals
- Particle track reconstruction
- Quantum (with) machine learning

Solving a problem on the D-Wave

Your 'homework' from yesterday...

Recall the QUBO for vertex cover we designed earlier in the lecture.

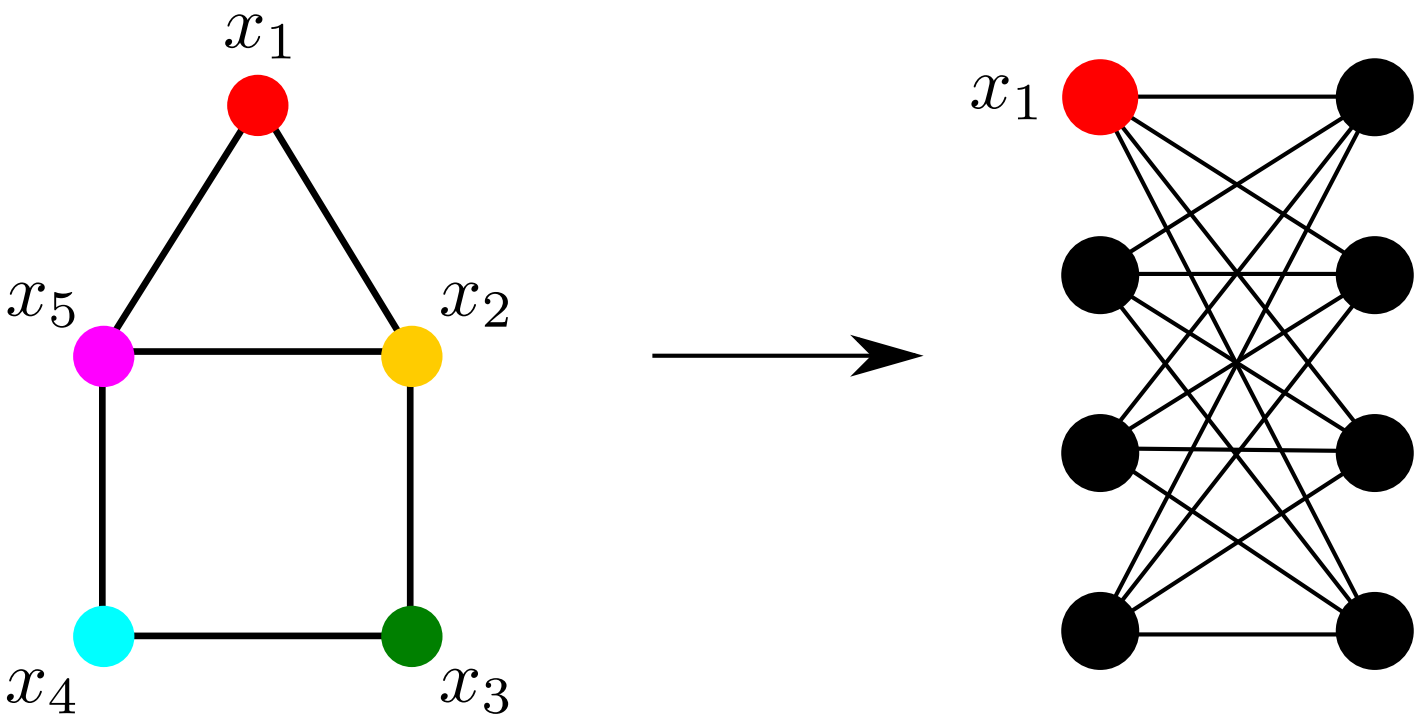
Write it out in full for the example graph. If you'd like, convert it to an Ising Hamiltonian - what are the coupling coefficients and external fields?



This graph has only 5 vertices; a D-Wave Chimera unit cell has 8. Can you assign the vertices to qubits in a way that satisfies the couplings?

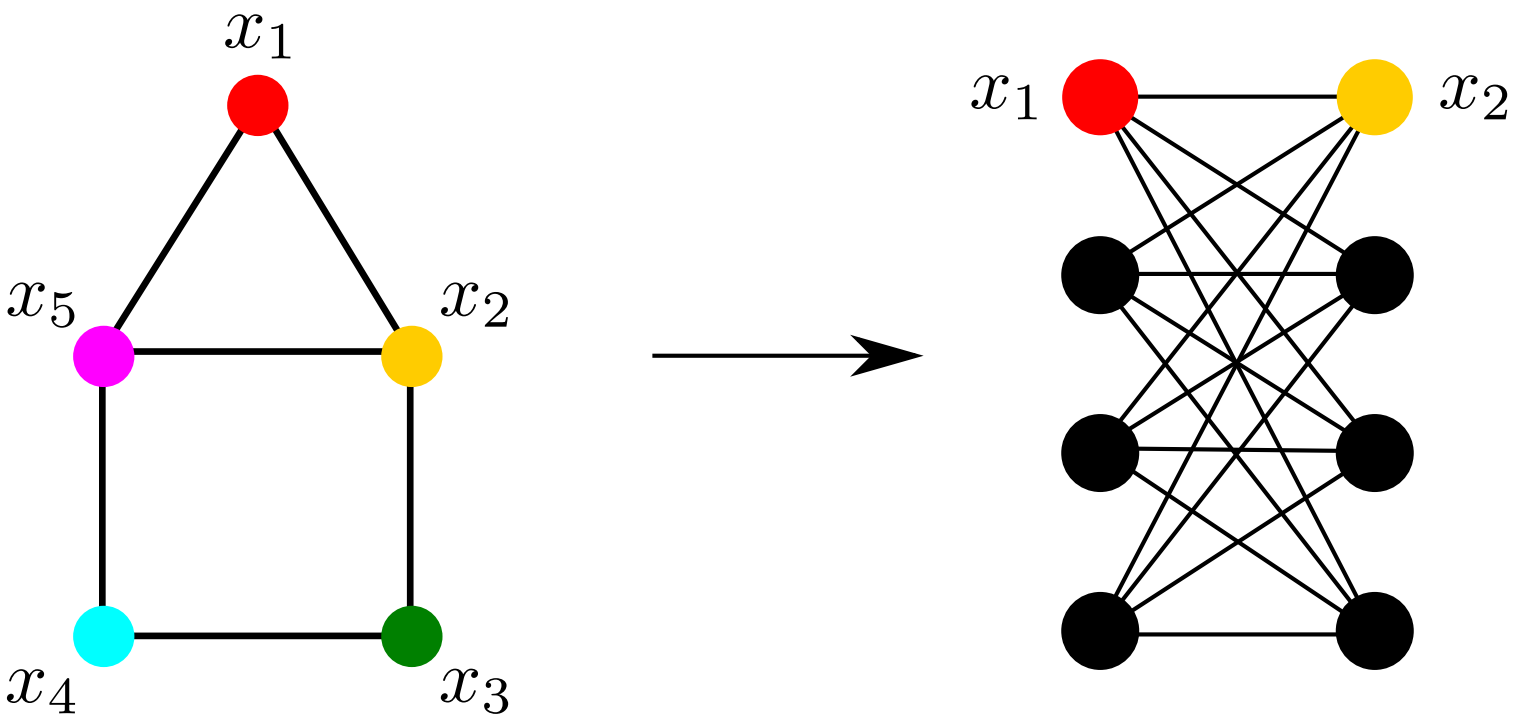
Your ‘homework’ from yesterday...

Let’s move clockwise around the graph...



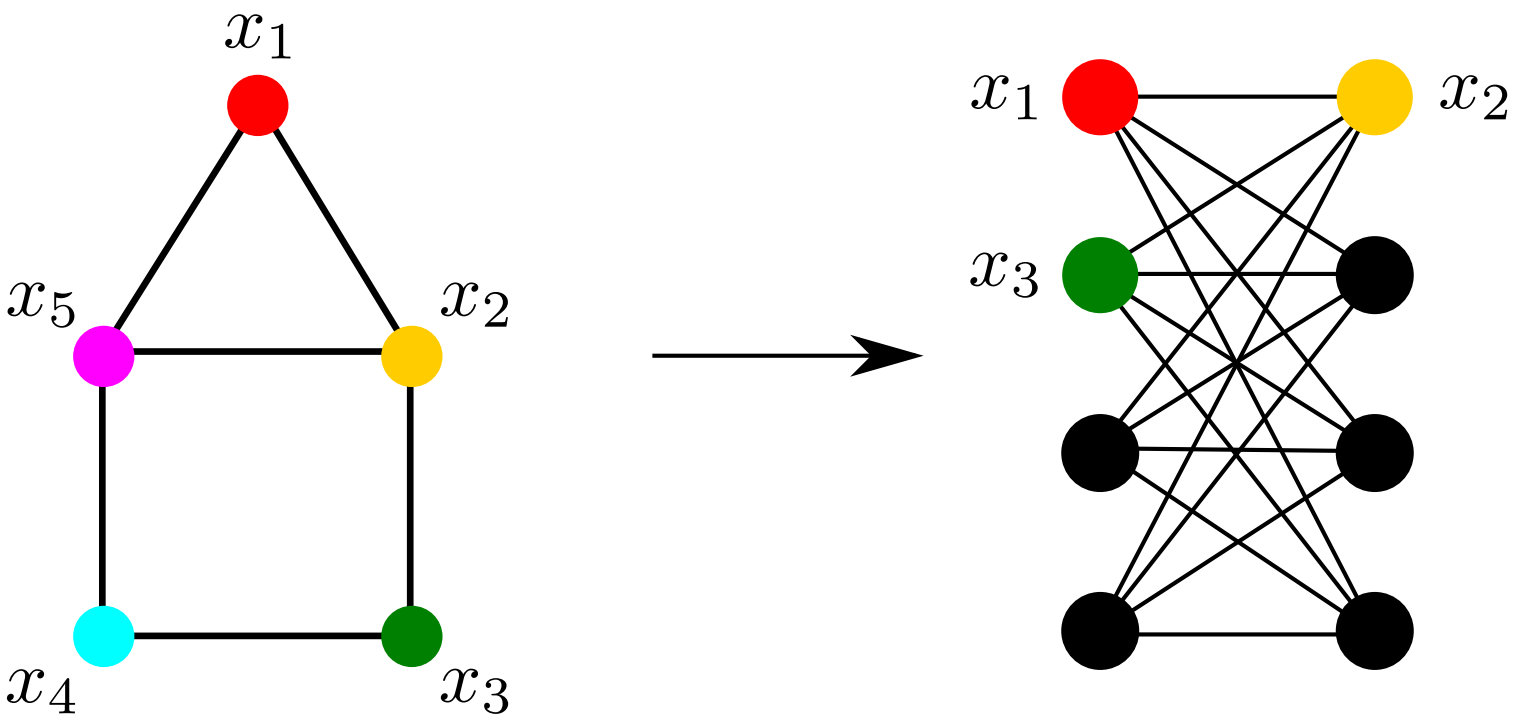
Your ‘homework’ from yesterday...

Let’s move clockwise around the graph...



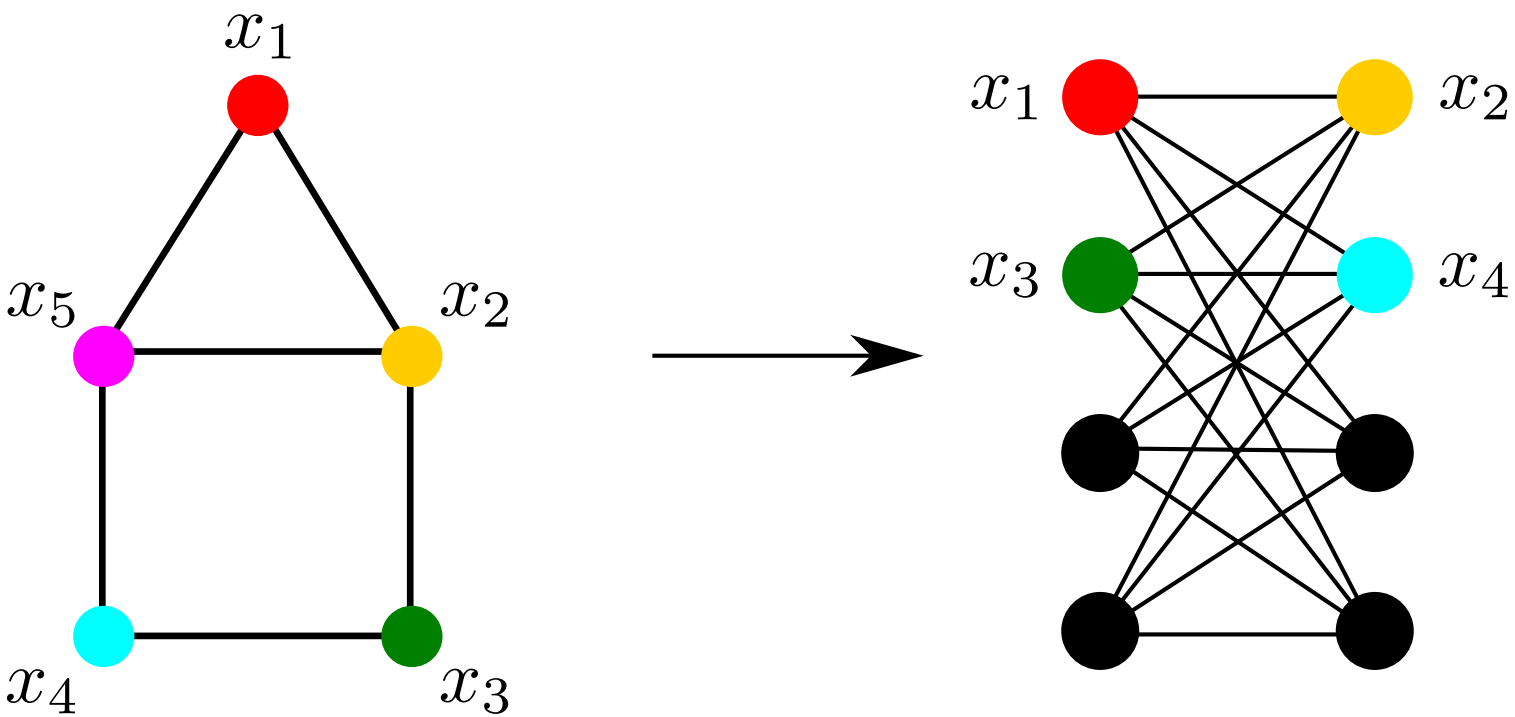
Your ‘homework’ from yesterday...

Let’s move clockwise around the graph...



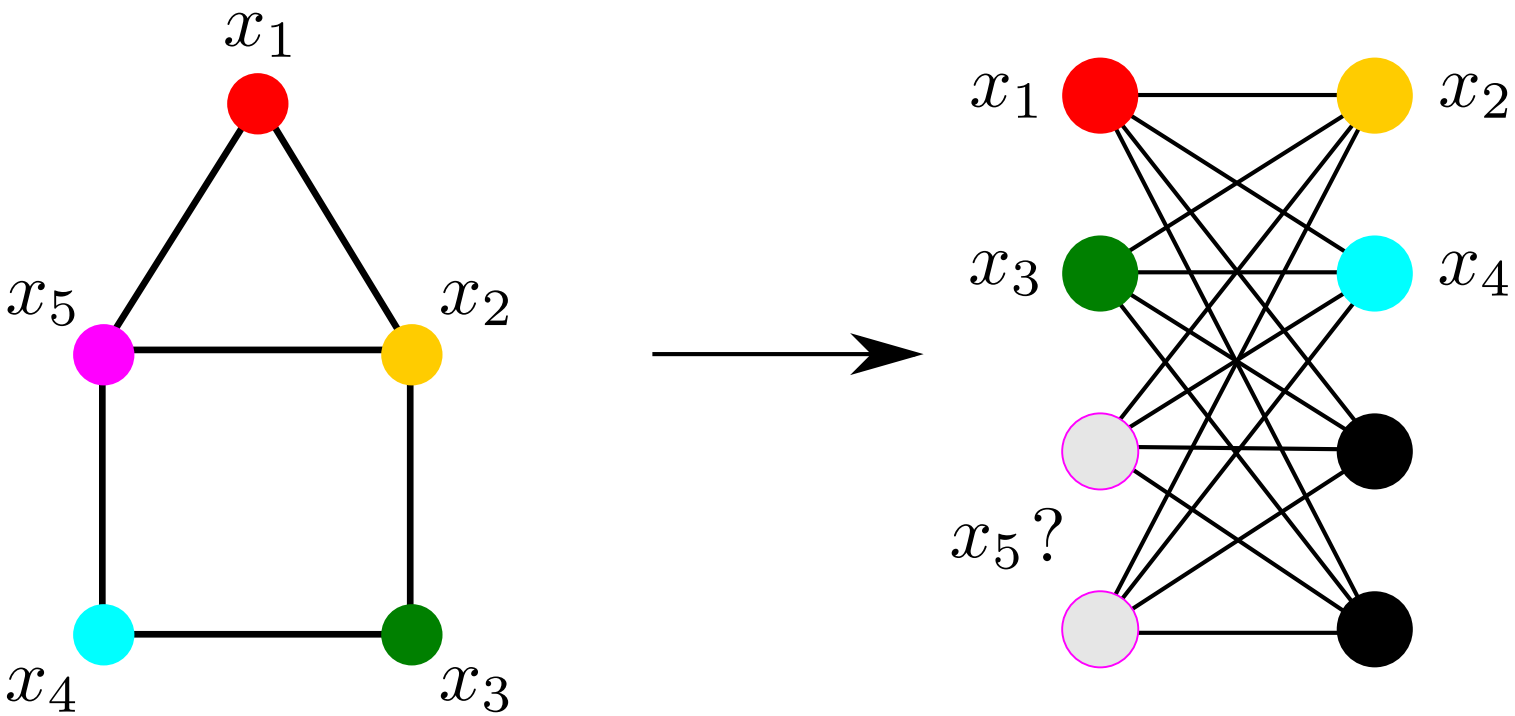
Your ‘homework’ from yesterday...

Let’s move clockwise around the graph...



Your 'homework' from yesterday...

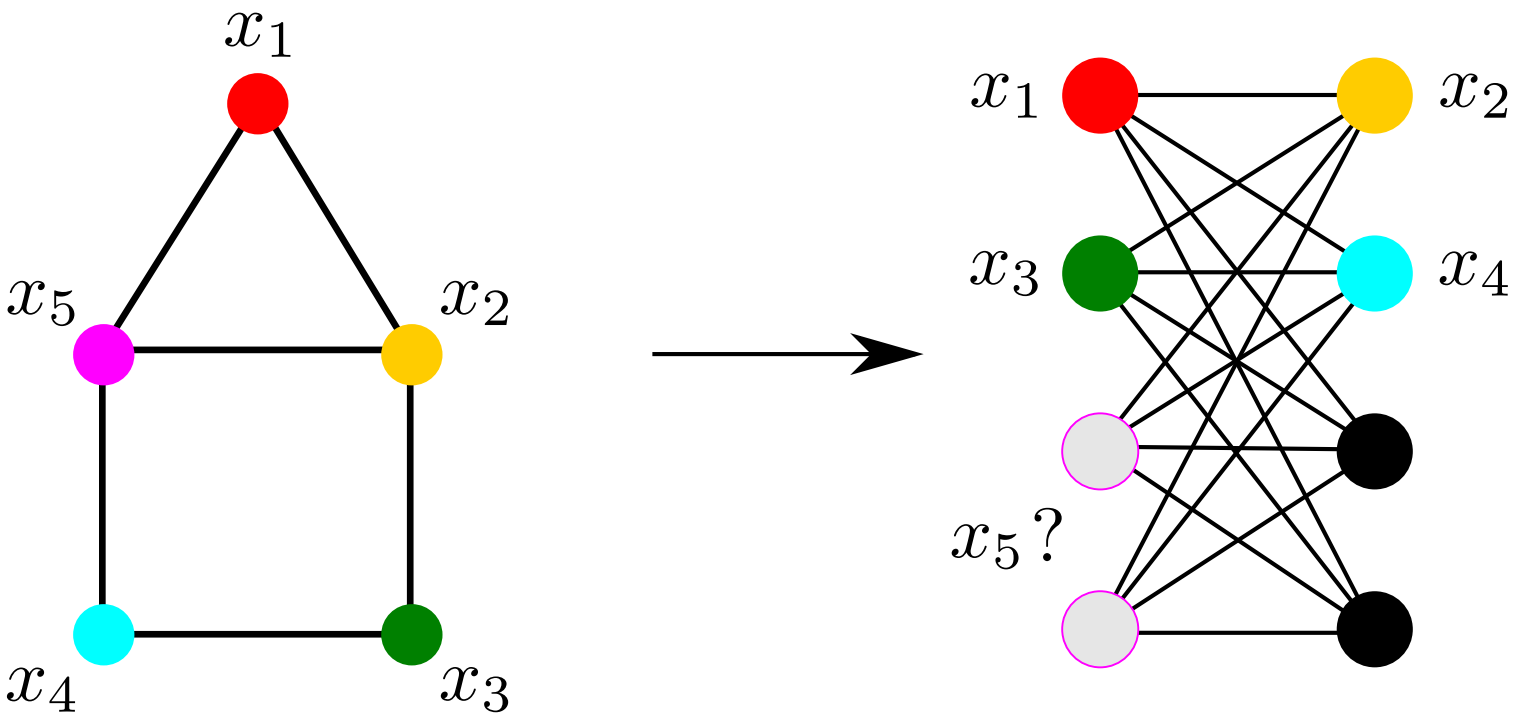
But... where do we put vertex 5? There is no remaining qubit that is attached to both qubits 1 *and* 2.



Actually, there is no qubit *at all* here that is attached to both 1 and 2. There are no triangles in a complete bipartite graph!

Your 'homework' from yesterday...

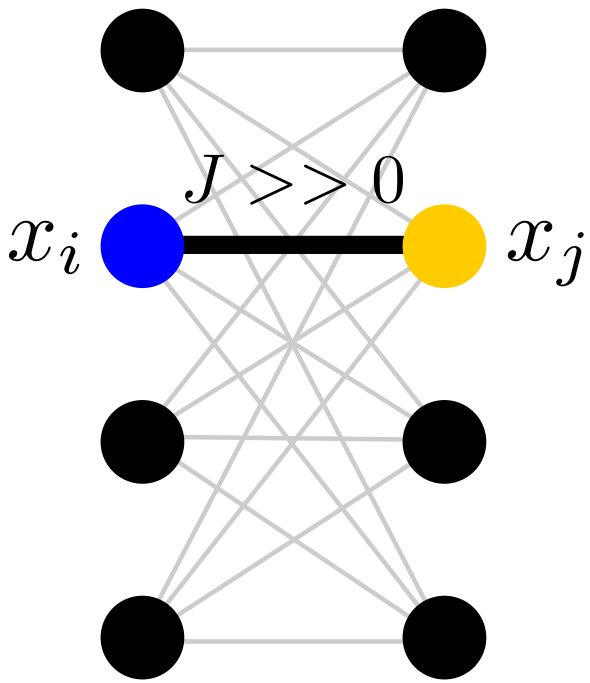
But, I said we were going to run vertex cover on the D-Wave... so how do we do this?



We need to find a *minor embedding* of the vertex cover problem graph onto the Chimera hardware graph.

Graph minor embeddings

When we need to couple together non-connected qubits, we can apply strong coupling coefficients to link many qubits together to make *chains*.



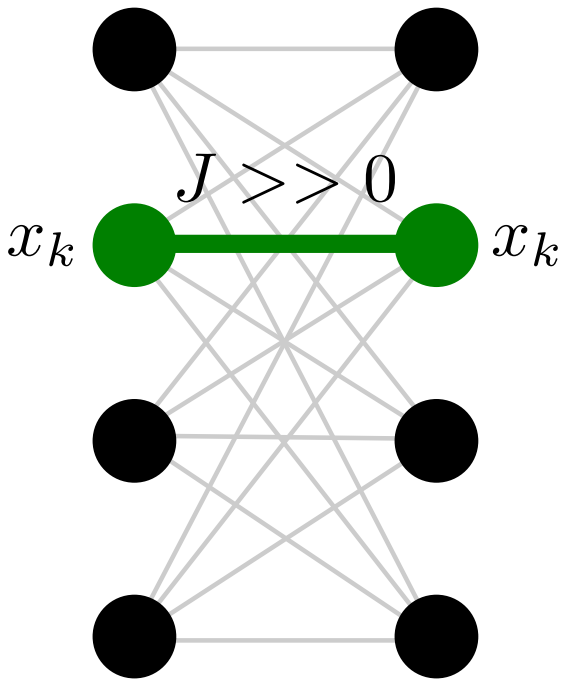
Suppose J_{ij} is very large.

If the spin at x_i is 1, then the spin at x_j will *also* want to be 1, because this is the configuration that minimizes the energy term

$$-J_{ij}x_ix_j$$

(Similarly if we had -1 for x_i.)

Graph minor embeddings



When we crank the couplings like this, we can force the two qubits to behave 'as one'.

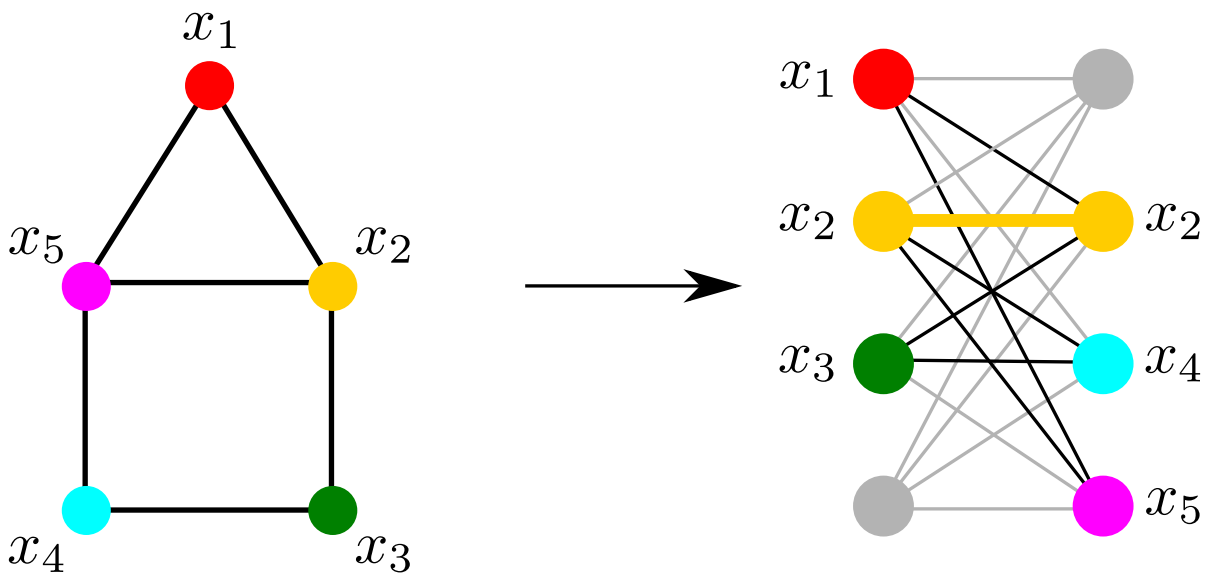
This enables us to have additional connections!

Such a configuration of qubit chains is called a *graph minor embedding*.

Note: we aren't just limited to chains of two qubits!

Graph minor embeddings

In order to implement vertex cover on the D-Wave, we will need to find such a minor embedding:



In general heuristic techniques are used to find these embeddings; thankfully D-Wave's Ocean SDK has one, `minorminer`, already implemented for us[3].

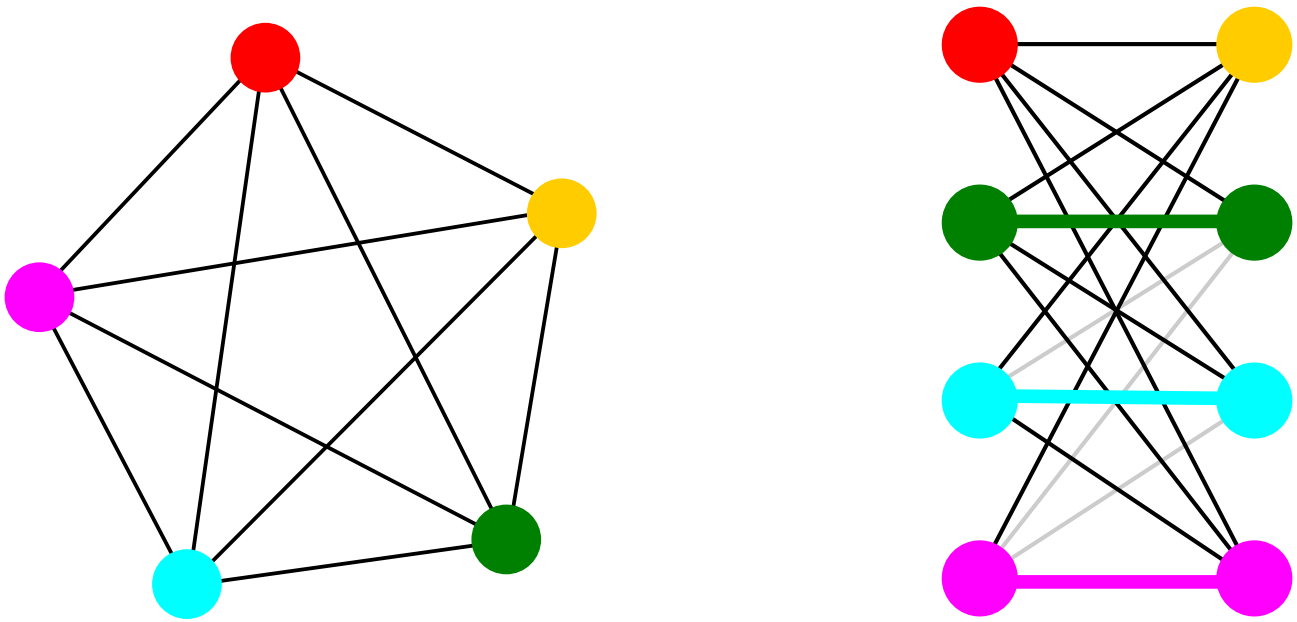
[3] For the algorithm, see `https://arxiv.org/abs/1406.2741`

Solving vertex cover on the D-Wave

Let's switch over to a Jupyter notebook and set up the vertex cover problem on the QPU!

Problem size restrictions

How large of a *fully connected* problem can we solve on a single Chimera unit cell?



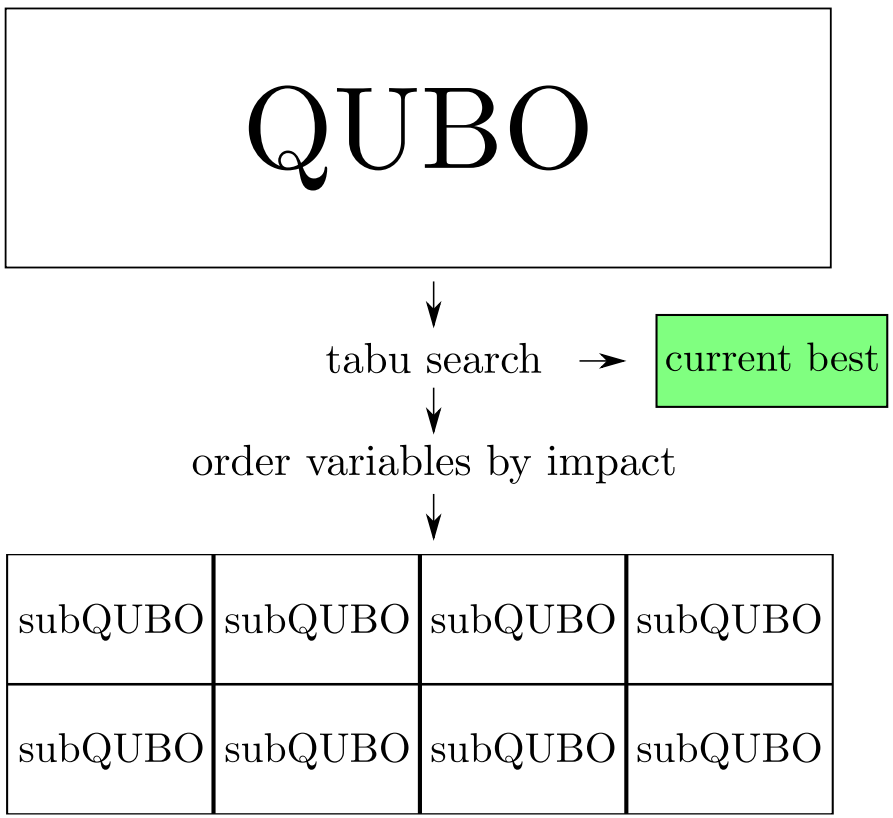
How about on the whole processor?

$$\text{Num. fully connected variables } = \sqrt{2n} + 1 \tag{6}$$

where n is the number of qubits; for the D-Wave 2000Q with 2048 qubits, we get 65 variables.

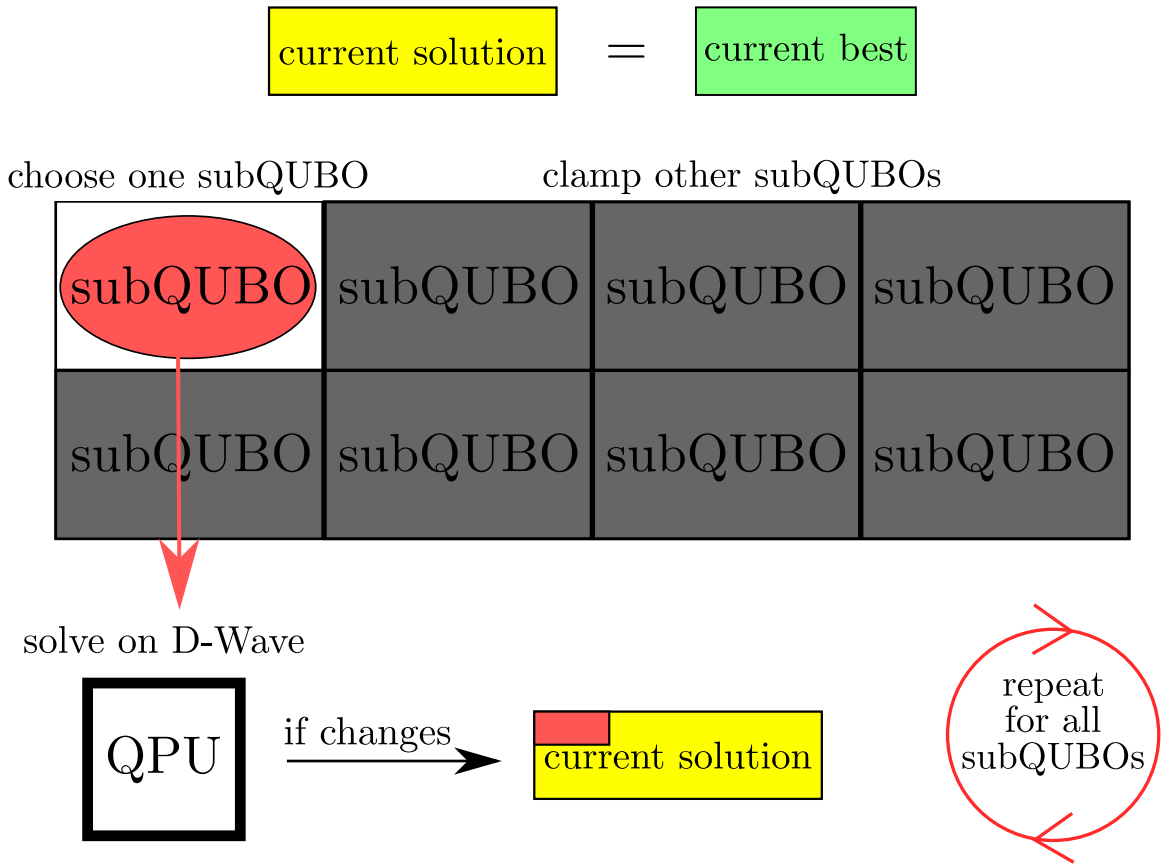
Overcoming problem size restrictions with qbsolv

What if our problem has more variables than this? Need to decompose the problem into subproblems!



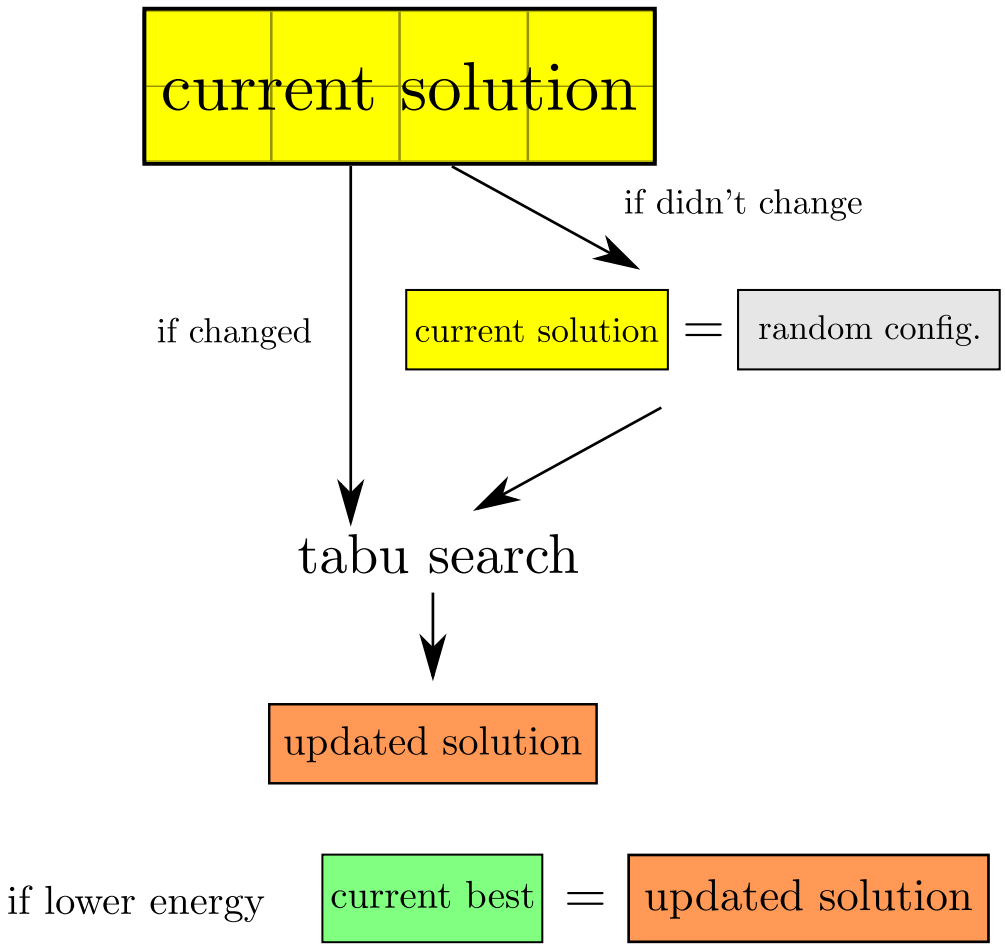
Overcoming problem size restrictions with qbsolv

What if our problem has more variables than this? Need to decompose the problem into subproblems!



Overcoming problem size restrictions with qbsolv

What if our problem has more variables than this? Need to decompose the problem into subproblems!



Overcoming problem size restrictions with qbsolv

The procedure described is iterated over many times.

Annealing on the D-Wave works well for searching *globally*, because we can tunnel to far away solutions.

tabu search works well for searching *locally*.

Next generation D-Wave hardware graph: Pegasus

Significant improvements in hardware connectivity.

Goes from max of 6 connections in Chimera to max of 15 connections.

Rumoured: first Pegasus machines in 2020 with around 5000 qubits.

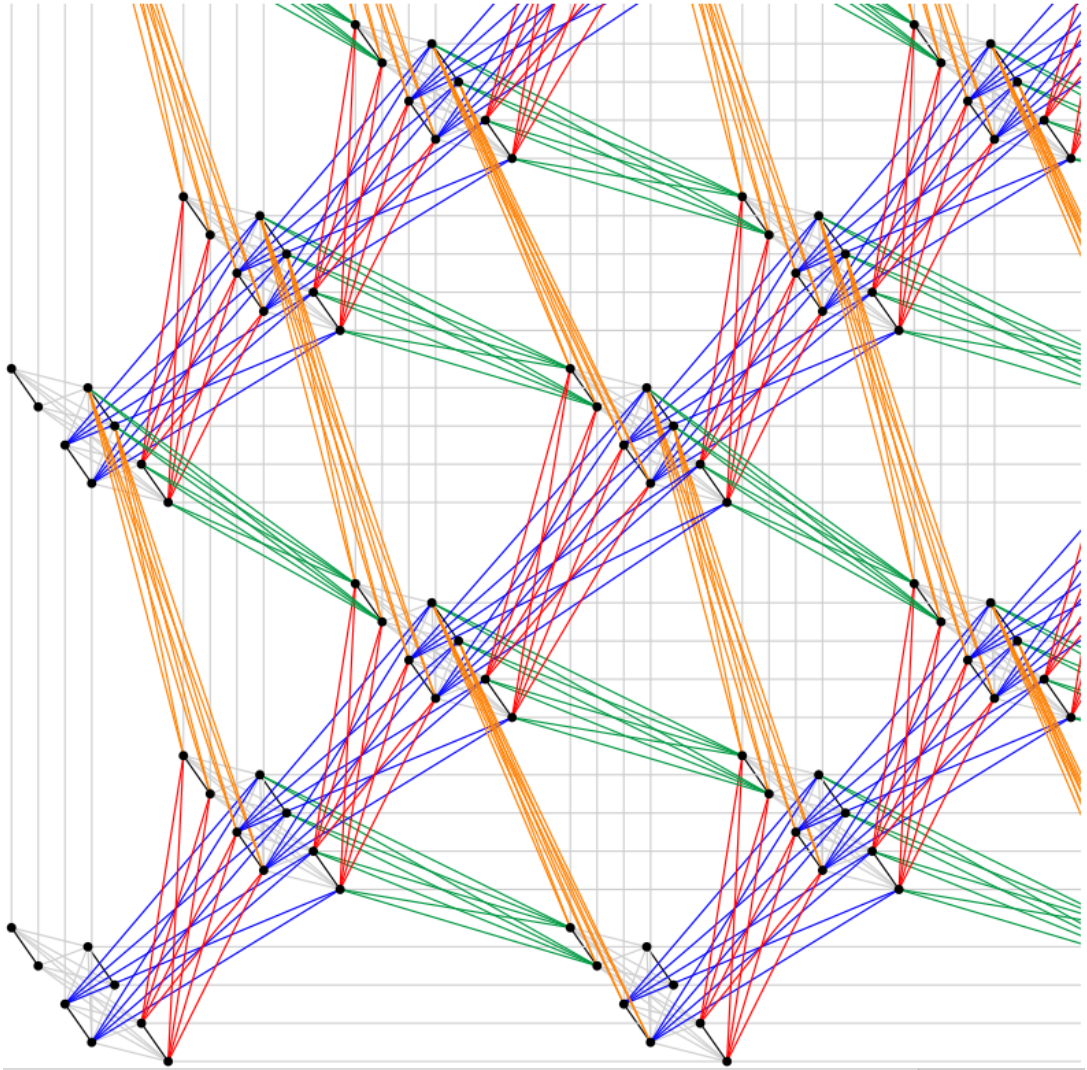


Image credit: `http://arxiv.org/abs/1901.07636`; see the paper for lots of different representations!

Applications of D-Wave hardware

What have people solved on it?

We've previously alluded to problems in graph theory and finance, but the D-Wave has been used to solve a wide array of optimization problems across different domains.

- Traffic flow optimization in Beijing (Volkswagen, 2017)
- Predicting outcome of elections (QxBranch, 2015)
- Finding grounds states of molecules (Volkswagen, 2019)

Let's look at some more domain-specific applications in HEP.

Formulating HEP problems as QUBO

As we've seen, coming up with a QUBO to represent a problem is not always intuitive.

I will showcase today two examples in HEP data analysis that have made very different use of the machines:

- Classification of Higgs signal/background
- Particle track reconstruction

Classification to identify Higgs decay on the D-Wave

nature
International journal of science

Letter | Published: 18 October 2017

Solving a Higgs optimization problem with quantum annealing for machine learning

Alex Mott, Joshua Job, Jean-Roch Vlimant, Daniel Lidar & Maria Spiropulu

Nature **550**, 375–379 (19 October 2017) | Download Citation

Main idea:

1. Train a number of *weak classifiers* on physically relevant parameters of the data
2. Use quantum annealing to choose which classifiers to include in the final model
3. Combine them together to form a *strong classifier*.

Higgs decay mechanism

The team analyzed a number of simulated events where the Higgs decays into two photons.

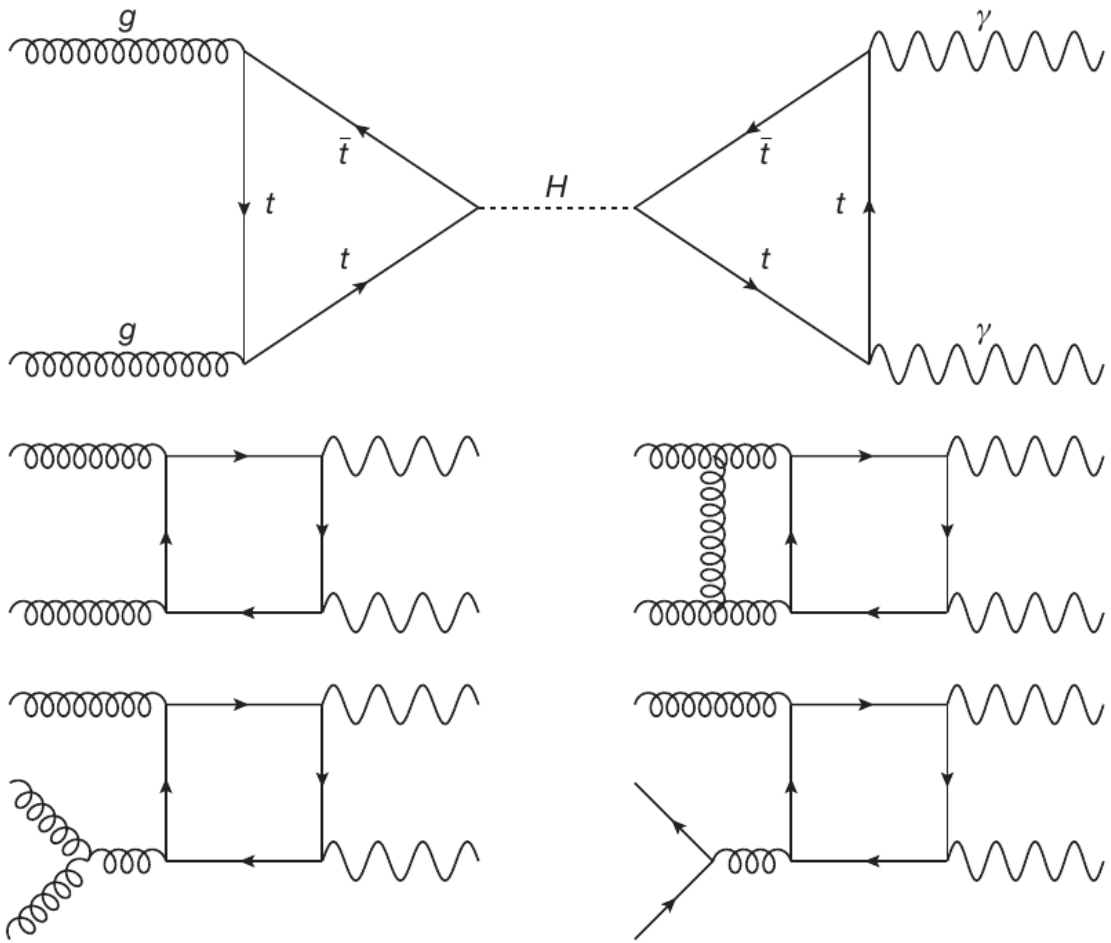


Image credit: `https://www.nature.com/articles/nature24047`

Physical parameters of Higgs decay

Each simulated event produces 8 different physical parameters.

Table 1 | The kinematic variables used to construct weak classifiers

Variable	Description
$p_{\mathrm{T}}^{1}/m_{\gamma\gamma}$	Transverse momentum (p_{T}) of the photon with the larger p_{T} (photon '1'), divided by the invariant mass of the diphoton pair ($m_{\gamma\gamma}$)
$p_{\mathrm{T}}^{2}/m_{\gamma\gamma}$	Transverse momentum (p_{T}) of the photon with the smaller p_{T} (photon '2'), divided by the invariant mass of the diphoton pair ($m_{\gamma\gamma}$)
$(p_{\mathrm{T}}^{1}+p_{\mathrm{T}}^{2})/m_{\gamma\gamma}$	Sum of the transverse momenta of the two photons, divided by their invariant mass
$(p_{\mathrm{T}}^{1}-p_{\mathrm{T}}^{2})/m_{\gamma\gamma}$	Difference of the transverse momenta of the two photons, divided by their invariant mass
$p_{\mathrm{T}}^{\gamma\gamma}/m_{\gamma\gamma}$	Transverse momentum of the diphoton system, divided by its invariant mass
$\Delta\eta$	Difference between the pseudorapidity $\eta=-\log[\tan(\theta/2)]$ of the two photons, where θ is the angle with the beam axis
ΔR	Sum in quadrature of the separation in pseudorapidity η and azimuthal angle ϕ of the two photons ($\sqrt{\Delta\eta^{2}+\Delta\phi^{2}}$)
$\lvert\eta^{\gamma\gamma}\rvert$	Pseudorapidity of the diphoton system

Image credit: `https://www.nature.com/articles/nature24047`

Physical parameters of Higgs decay

The probability distribution of each physical quantity was visibly different for the signal/background cases.

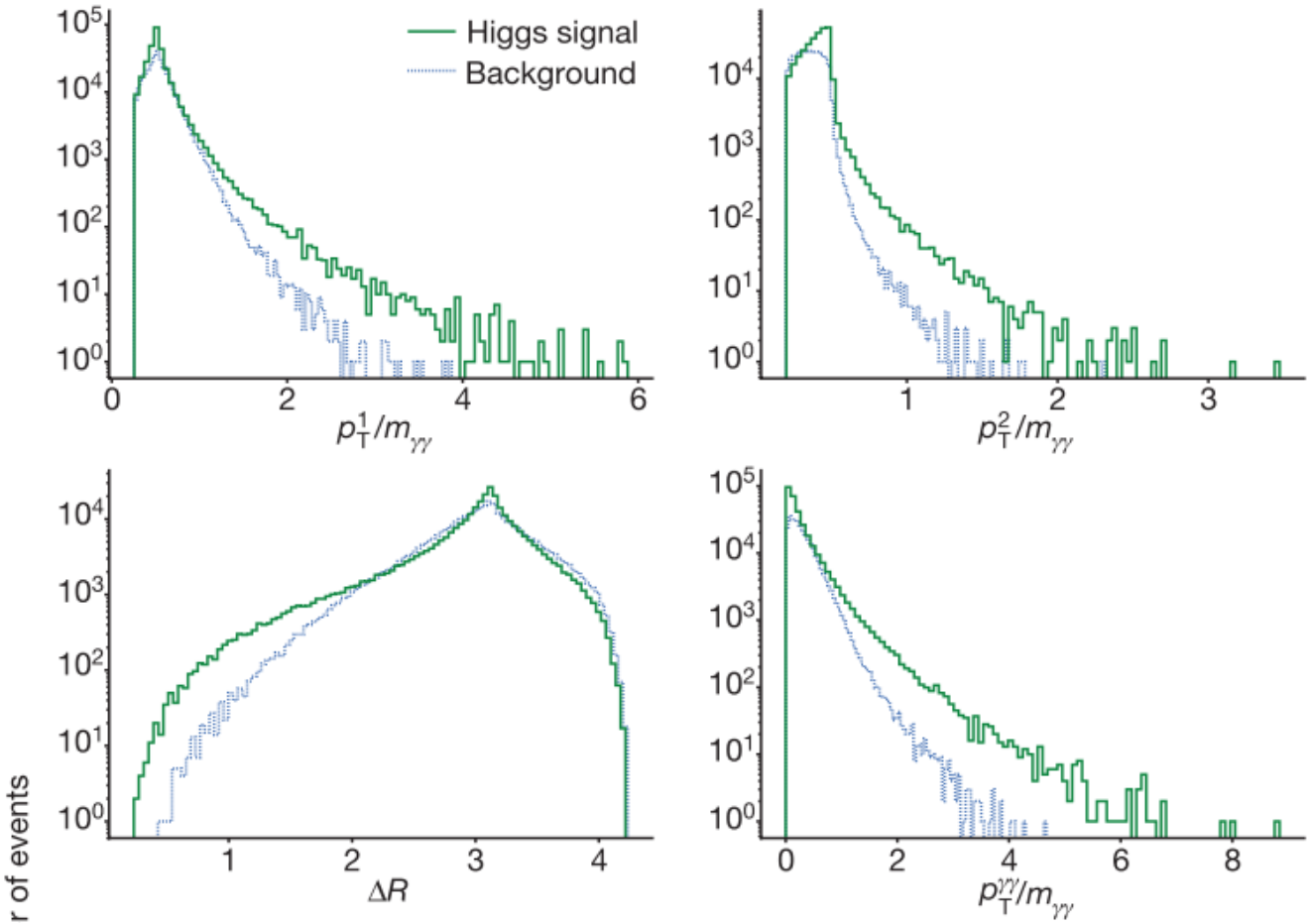


Image credit: `https://www.nature.com/articles/nature24047`

Higgs decay training data

Various sums and products of the 8 quantities were taken to obtain a total of 36 parameters.

An item in the training set is denoted as

$$\mathcal{I} = \{\mathbf{x}_\tau, y_\tau\}$$

where $\mathbf{x}_\tau$ is a 36-dimensional vector for each event, and $y_\tau = +1$ for decay signal observed, and -1 for background.

Weak classifiers

For every physical parameter, we can define a *weak classifier* that decides whether or not an observed value is signal, or background.

Let v be a physical parameter. Use the simulated values of v to create 'boundaries' around the possible values it can take if it is signal vs. background.

For example, let

- v_{+1} be the value of v *above* which we are very confident it is signal
- v_{-1} be the value of v *below* which we are very confident it is background

Weak classifiers

They then used the following form for the weak classifiers[4]:

$$c(v) = \begin{cases} +1 & \text{if } v_{+1} < v \\ \frac{v}{v_{+1}} & \text{if } 0 < v < v_{+1} \\ \frac{v}{|v_{-1}|} & \text{if } v_{-1} < v < 0 \\ -1 & \text{if } v < v_{-1} \end{cases} \tag{7}$$

[4] I've simplified the notation here and moving forward, and assumed that the values are renormalized to be centered around 0.

From weak classifiers to strong classifiers

The weak classifiers can't individually classify the values with certainty; the parameters take on a range of values. Instead, we will combine them together to form a *strong* classifer.

Let $\mathbf{x}$ be a vector of all 36 parameters. Then our strong classifier will decide if it's signal or background by summing up some combination of the weak classifiers:

$$R(\mathbf{x}_\tau) = \frac{1}{36} \sum_{i=1}^{36} c_i(\mathbf{x}_\tau) \quad \in [-1, +1] \tag{8}$$

If $R(\mathbf{x}_\tau) > 0$ we'll mark it as signal; if $R(\mathbf{x}_\tau) < 0$, we'll mark it as background.

Quantum annealing for weak classifier selection

But which classifiers (i.e. physical parameters) should we include in this model? How do we know which ones are the most important, or if any of them are redundant? Let's write a new strong classifier

$$R(\mathbf{x}_\tau) = \frac{1}{36} \sum_{i=1}^{36} b_i c_i(\mathbf{x}_\tau) \quad \in [-1, +1] \tag{9}$$

where the b_i will indicate whether or not we should include classifier i.

We will use the quantum annealer to help us choose which classifiers to use - whether or not a classifier is included is denoted by a binary variable, so we should be able to formulate some sort of QUBO.

Ising model conversion

Let b_i be a binary variable that represents whether or not we include classifier i in the model.

For a given data point $\mathbf{x}_\tau$, it is good if classifiers agree with each other, but we also don't want to include redundant information. We will use this to design a coupling term that quantifies their *agreement* over the whole data set:

$$C_{ij} = \sum_\tau c_i(\mathbf{x}_\tau) c_j(\mathbf{x}_\tau),$$

Classifiers also have to agree with the training labels! So let's add a linear term that quantifies the *quality* of a given classifier:

$$C_i = \sum_\tau c_i(\mathbf{x}_\tau) y_\tau$$

Ising model conversion

So our QUBO is starting to look like this...

$$\min_{\mathbf{b}} \sum_{ij} \left(\sum_{\tau} c_i(\mathbf{x}_\tau) c_j(\mathbf{x}_\tau) \right) b_i b_j - \sum_{i} \left(\sum_{\tau} c_i(\mathbf{x}_\tau) y_\tau \right) b_i$$

Or, filling in the coefficients,

$$\min_{\mathbf{b}} \sum_{ij} C_{ij} b_i b_j - \sum_{i} C_i b_i$$

But recall we also don't want to include too many classifiers, or any unnecessary ones. So let's add a penalty term too.

Ising model conversion

We'll get something like this:

$$\min_{\mathbf{b}} \sum_{ij} C_{ij} b_i b_j - \sum_i C_i b_i + \gamma \sum_i b_i$$

We can make the change of variables $b_i = \frac{1}{2}(s_i + 1)$ to the spin domain, and write

$$\min_{\mathbf{s}} \sum_{ij} J_{ij} s_i s_j + \sum_i h_i s_i$$

where, rearranging the coefficients,

$$J_{ij} = C_{ij}/4, \qquad h_i = \frac{1}{2}\left(\lambda - C_i + \sum_j C_{ij}\right)$$

Ising model conversion

Now that we have the Hamiltonian, we can send it to the quantum annealer for minimization.

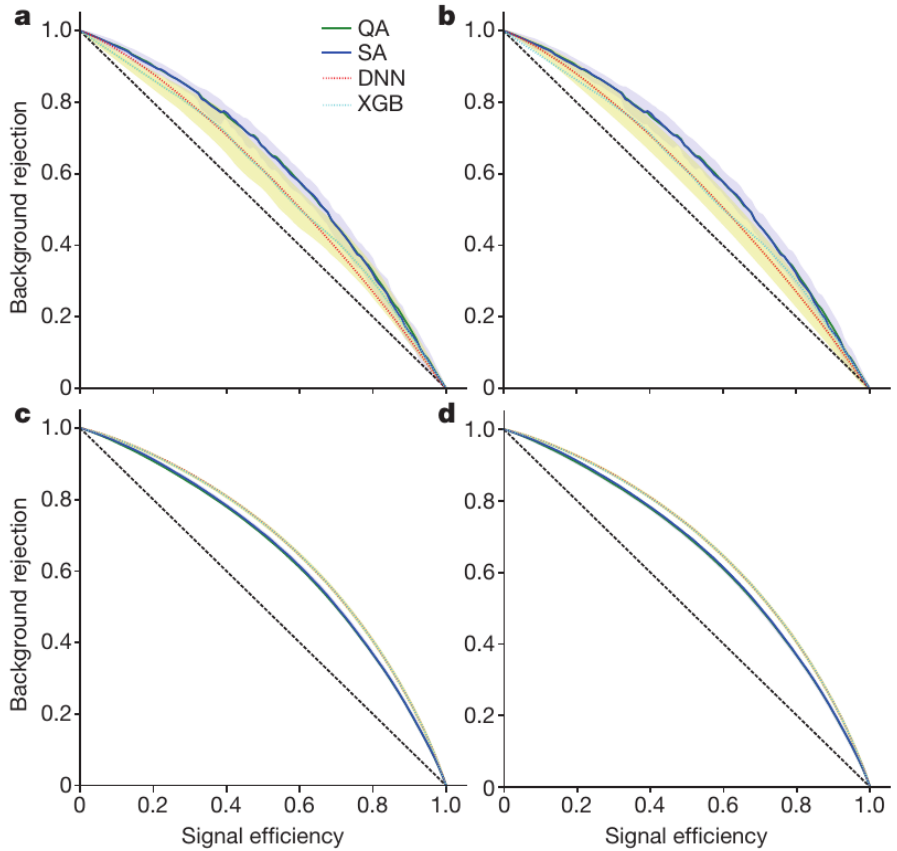
We can use the output $\mathbf{s}$, which we hope is the ground state, to construct our strong classifier:

$$R(\mathbf{x}_\tau) = \frac{1}{36} \sum_{i=1}^{36} s_i c_i(\mathbf{x}_\tau) \quad \in [-1, +1] \tag{10}$$

We can then apply this classifier to *testing data*, and see how many events we can classify correctly.

Results

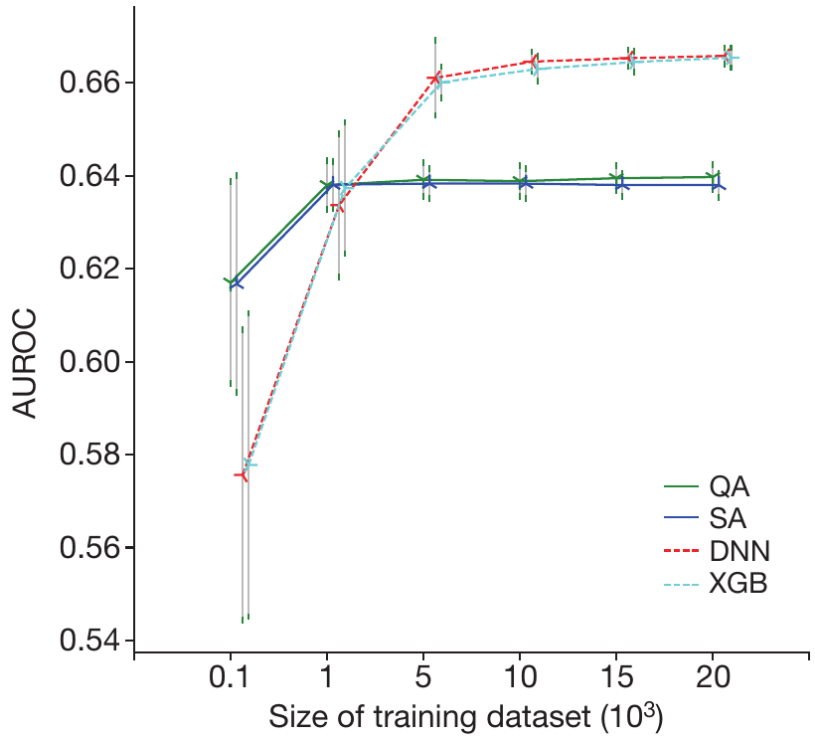
Trained using 20 sets of 100 (top) or 20000 (bottom) simulated events using 4 different methods: quantum annealing, simulated annealing, deep neural networks, and XGBoost.



All methods are better than the random classifier (black line).

Results

Interesting point: QA/SA seem to be more effective for smaller training sets.



Hard to draw definitive conclusions though because QA machines are noisy, and uncertainties are large.

Results

Interesting point: weak classifiers appeared in the solutions with varying frequency.

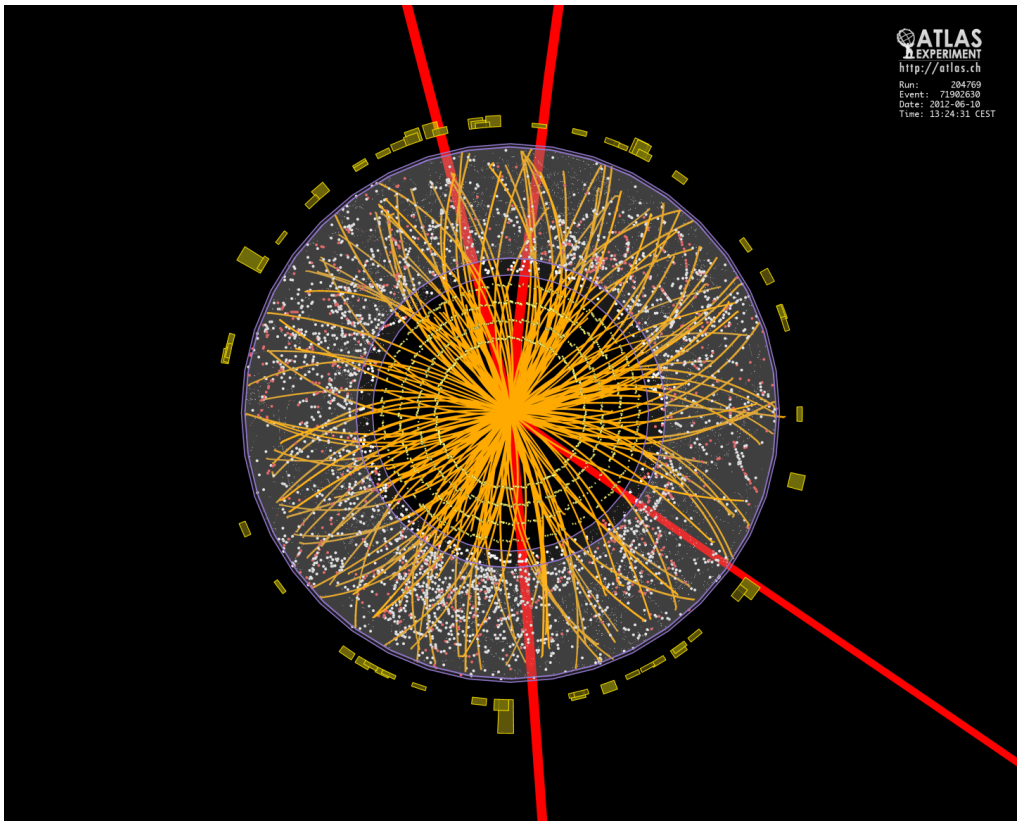
For a test run with $\lambda = 0.8$ (i.e. heavy penalty on the number of classifiers),

- Two of the original 8 variables, p_T^1 and $|\eta^{\gamma\gamma}|$, are never included in the model
- Three classifiers, $p_T^2/m_{\gamma\gamma}, (\Delta R p_T^{\gamma\gamma})^{-1}, p_T^2/p_T^{\gamma\gamma}$ are included in all of the models for all repeated trials at this λ

The authors interpret the (non)-importance of the variables using the expected kinematics of the system (e.g. Higgs events tend to have lower $p_T^{\gamma\gamma}$ than a spurious background process that produces two photons)

Particle track reconstruction

Many of you at TRIUMF are quite familiar with images like this...



Given a set of detector hits, determine which hits belong together to form *particle tracks*.

Image credit: `https://cds.cern.ch/record/1459496`

Particle track reconstruction and machine learning

The amount of data produced by the LHC is already large (CERN site: 25 GB/s).

This is going to increase even more in 2026 when it starts the high-luminosity runs.

People already look to conventional machine learning techniques to deal with this amount of data.

Image credit: `https://www.kaggle.com/c/trackml-particle-identification`

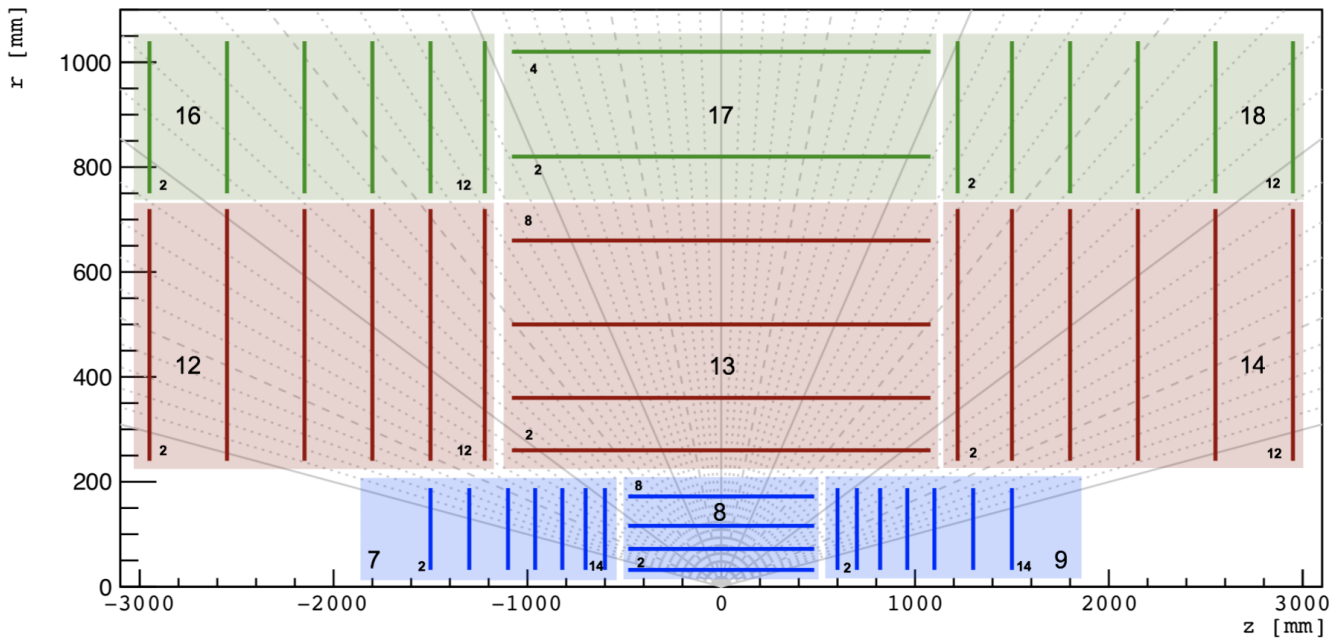
Particle track reconstruction using quantum annealing

People are also beginning to think about how we can use quantum hardware or quantum machine learning techniques to deal with the massive amount of data.

How do we transform the study of detector data points into a QUBO problem? What are we even optimizing here?

Particle track reconstruction data

Dataset (TrackML) consists of a bunch of particle 'hits'. Each hit contains information about coordinates in the detector, momentum, etc.



We want to group together hits to form the tracks of individual particles traveling through the detector.

Image credit: `https://www.kaggle.com/c/trackml-particle-identification`

Particle track reconstruction data

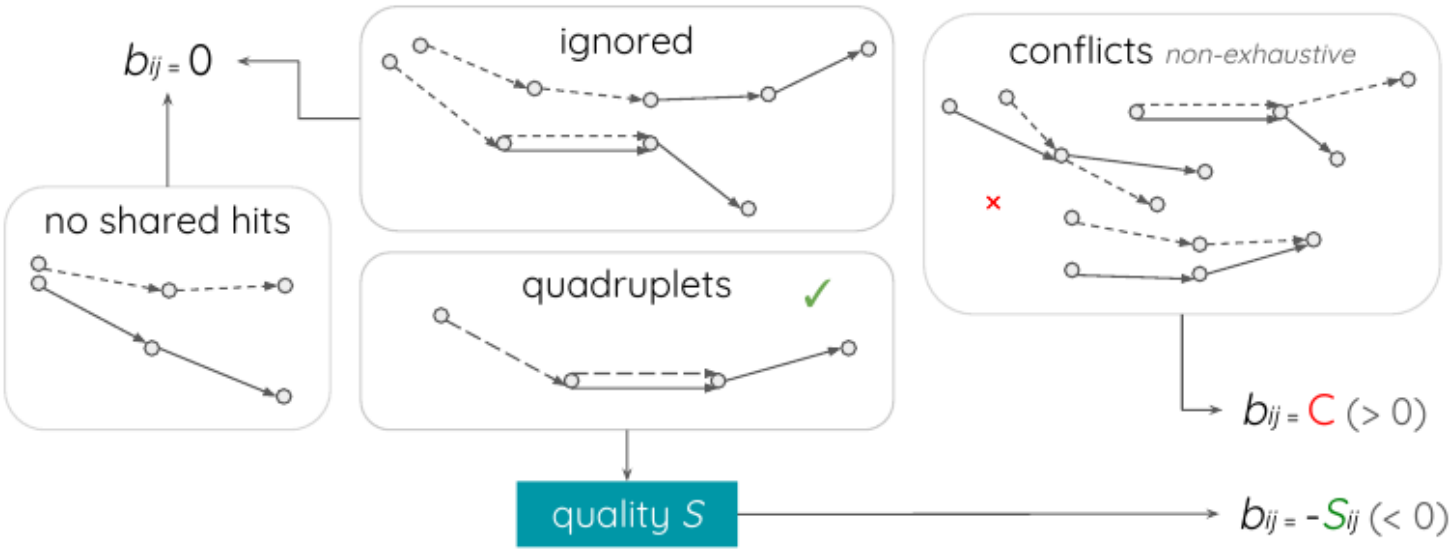
How do we determine which points make up which tracks?

First, make couple assumptions are made about track structure:

- Points on travel in a helix moving 'forward' in the detector
- Hits on the same track are registered in consecutive layers of the detector with very few holes
- Tracks with fewer than 5 hits are invalid
- Any given hit can belong to at most one track

Particle track reconstruction

Key idea: group points into doublets, triplets, and quadruplets. Determine which such *groups* should belong to a particle track.



(Earlier work done using only doublets; using triplets or quadruplets reduces the size of the space)

Image credit: `http://arxiv.org/abs/1902.08324`

QUBO construction

Assign binary variables to the triplets, and give them coupling strengths that reflect their quality.

$$O(a, b, T) = \sum_{i=1}^{N} a_i T_i + \sum_{i}^{N} \sum_{j<i}^{N} b_{ij} T_i T_j \tag{11}$$

For every pair of triplets (T_i, T_j)

- Set $b_{ij} = 0$ if there is no overlap between points
- Set $b_{ij} < 0$ with value based on compatibility of physical parameters (angles between doublets, direction)
- Set $b_{ij} > 0$ if there is a conflict

(a reflects the probability of any given triplet being part of a track, and all are set to the same initial value)

Annealing pipeline

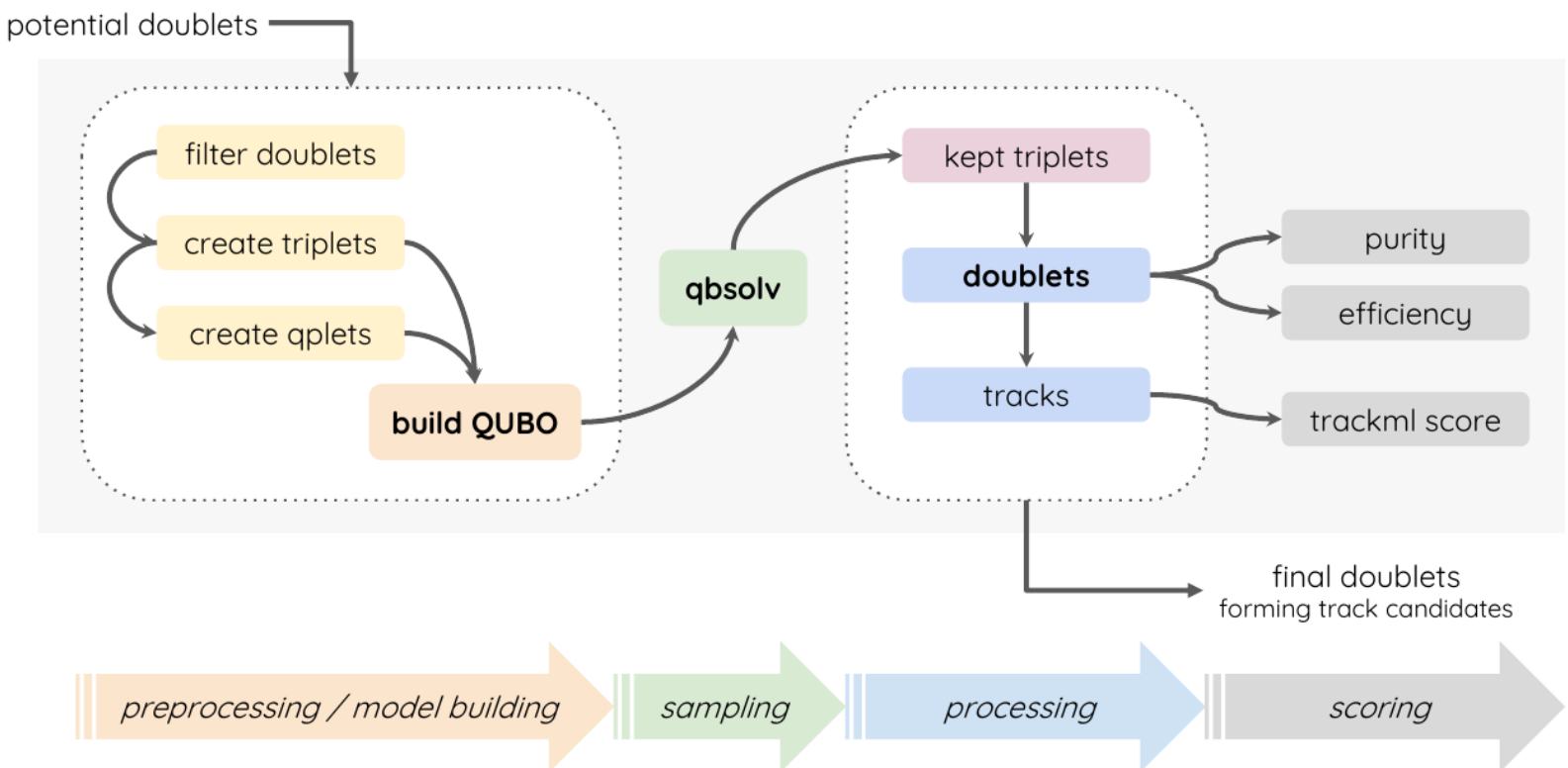


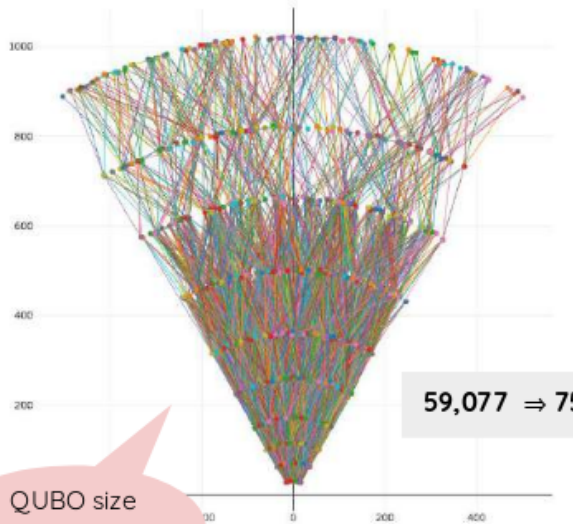
Image credit: `http://arxiv.org/abs/1902.08324`

Results

It works!

186 particles in a phi slice of π/3

precision (%): 98.5, recall (%): 98.4,
trackml score (%): **98.35**



QUBO size
68,043

59,077 ⇒ 752 doublets

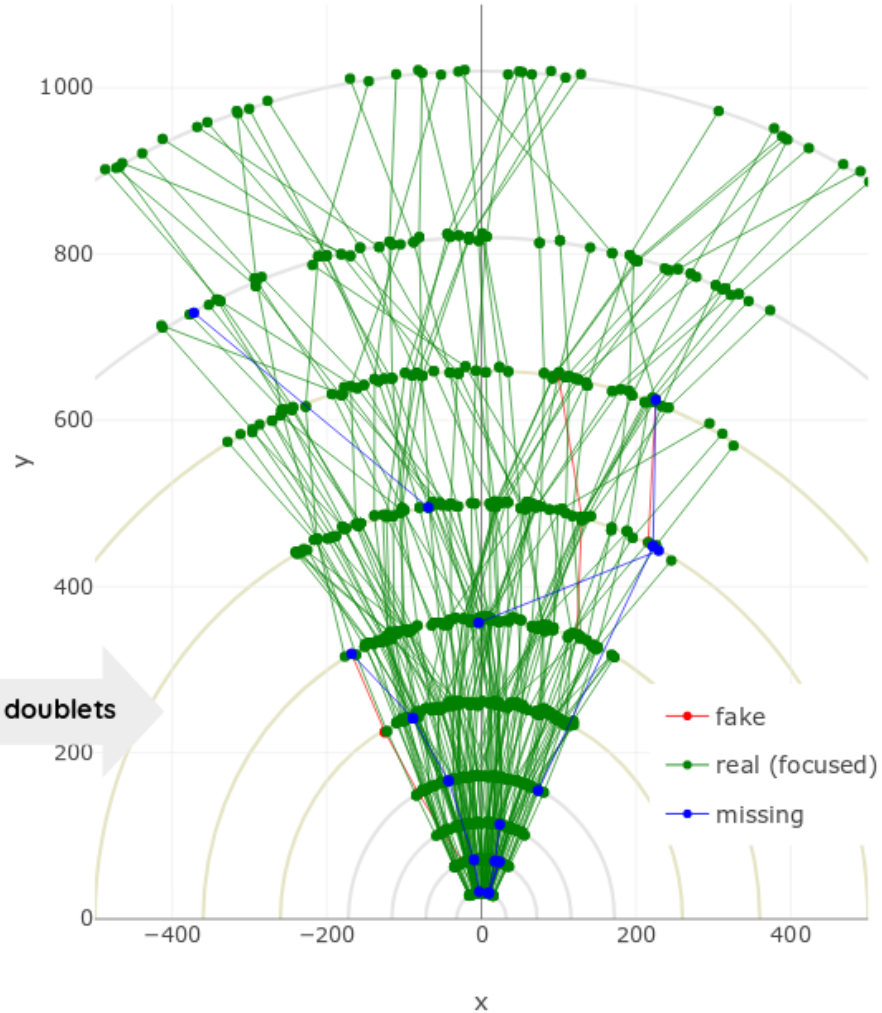
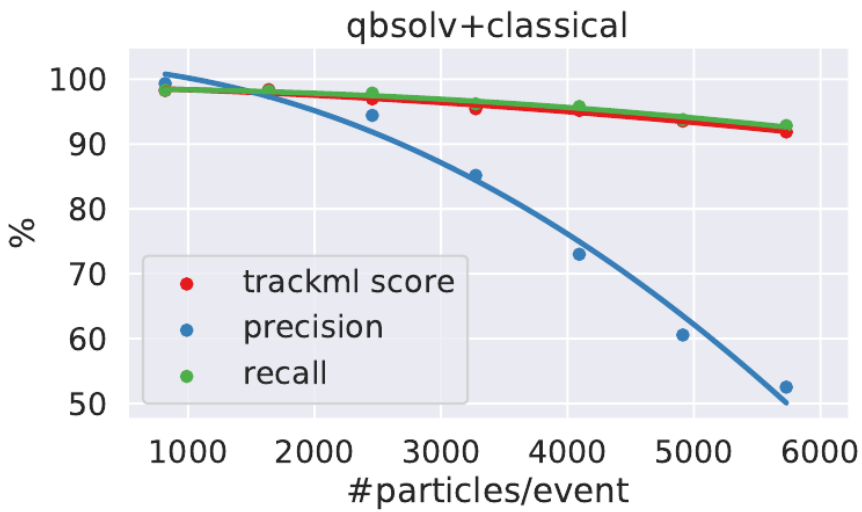
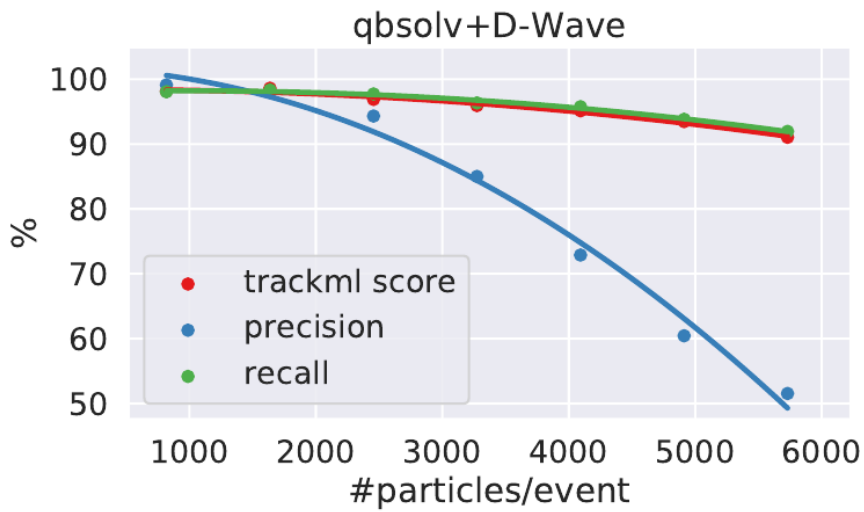


Image credit: MSc thesis of Lucy Linder `https://github.com/derlin/hepqpr-qallse/tree/master/doc`

Results





Metrics compare reconstructed doublets with 'true' doublets.

- precision - how many of the identified doublets are true?
- recall - how many of the true doublets did we find?

Did it do better?

At least for the current generation of annealers, there was no advantage observed for using the D-Wave:

- largest classical runs took $\sim$2h (wall time) to completion
- largest D-Wave runs took over 5h; overhead due to communication latency, need to perform minor embedding performed on a shared remote machine

In some cases the energy returned by the annealer was in fact *lower* than the known optimum, but this was attributed to finding 'fake' tracks.

Quantum machine learning

Quantum machine learning (QML)

QML is an emerging area of research.

As suggested by the two HEP applications above, there are a number of ways in which machine learning and quantum computing can intersect.

"QML" can refer to a number of different ideas, which are experiencing varying degrees of success...

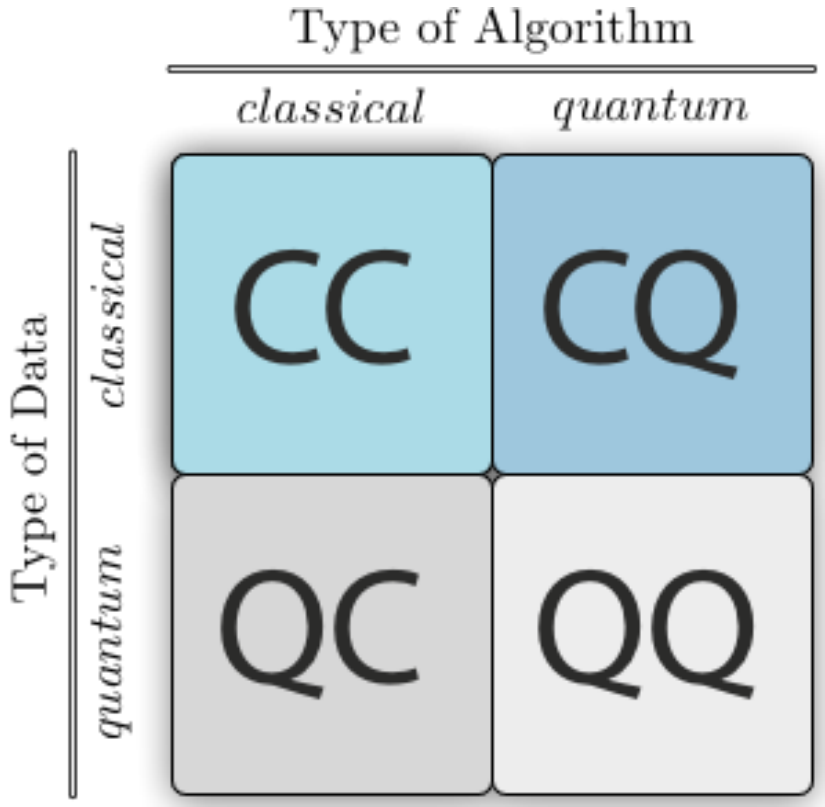


Image credit: By Maria Schuld - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=55676381

v1: machine learning to solve quantum physics problems

Very successful!

Classical machine learning algorithms (running on classical computers) have been applied to a number of different problems in quantum computing:

- Quantum tomography; reconstruction of quantum states or processes using experimental data[5]
- Designing new quantum optics experiment setups [6]
- Identifying phase transitions

[5] I do research in this area, come chat if you're interested!

[6] `https://arxiv.org/abs/1509.02749`

v2: using quantum hardware to do ML on classical data

This seems like a great idea - in classical machine learning, we need to work with a lot of data. Can't we just put all that data in superposition and use quantum computing?

[7] https://arxiv.org/abs/0811.3171

v2: using quantum hardware to do ML on classical data

This seems like a great idea - in classical machine learning, we need to work with a lot of data. Can't we just put all that data in superposition and use quantum computing?

In principle, yes, and it might even make things faster.

Many classical ML algorithms depend heavily on linear algebra subroutines; there exists the *HHL algorithm*[7] for solving sparse linear systems, that runs in time logarithmic in the matrix size (rather than polynomial).

[7] `https://arxiv.org/abs/0811.3171`

v2: using quantum hardware to do ML on classical data

In practice, you run into a number of problems.

You need to build a **qRAM** - some means of loading the classical data into the quantum computer. You need to design (and implement in hardware) some unitary matrix that does:

$$\mathbf{x} = (x_1, x_2, \ldots x_n) \rightarrow \sum_i x_i |i\rangle \tag{12}$$

i.e. read in the data as amplitudes, or

$$\sum_i |i\rangle|0\rangle \rightarrow \sum_i |i\rangle|b_i\rangle, \quad b_i \in \{0, 1\} \tag{13}$$

i.e. read in the data as bits.

v2: using quantum hardware to do ML on classical data

If the number of queries you need to make to the qRAM is polynomial, there exist qRAM constructions using a polynomial number of gates for which you may not need to do full fault-tolerant quantum computing.

However if you need to do an exponential number of queries, you need to make your error rates exponentially small, and so you need full fault-tolerance - this leads to a *huge* overhead in the number of additional qubits required to query qRAMs of substantial size. [8]

[8] I have done some of the resource estimates for this - the numbers are sad. See `https://arxiv.org/abs/1902.01329`

v2: using quantum hardware to do ML on classical data

Furthermore, at the end of the algorithm, you need to get the information out!

A classical machine learning algorithm returns a classical vector.

A quantum algorithm returns a quantum state, which we will need to produce many copies of and then measure to get the output.

v3: doing an entire ML process on quantum hardware

Is it possible to design a neural network where *all the components are quantum*? e.g.

- Some quantum system models the neurons[9]
- Training the network happens via some quantum algorithm[10]

These ideas are not yet mature, but there are some specific types of neural network where there is progress, and this is where current-generation quantum hardware might shine.

[9] See `https://arxiv.org/abs/1412.3635`)

[10] See `https://github.com/ml-lab/Quantum-Backpropagation`

v4: using quantum hardware as a sampler

Suppose we are designing a classifier - there are two ways we can accomplish this. Given some input training data $\{\mathbf{x}_i\}_i$ and associated labels $\{y_i\}$,

1. Discriminative - learn a *conditional* probability distribution that predicts the labels, i.e. $P(y|\mathbf{x})$
2. Generative - learn a *joint* probability distribution that encompasses both the data and the labels, i.e. $P(\mathbf{x}, y)$

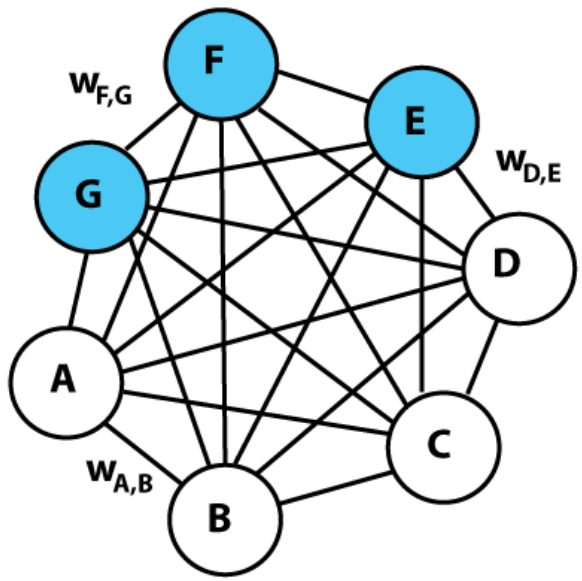
v4: using quantum hardware as a sampler

Training of some generative models requires evaluation of hard-to-compute functions of the probability distribution - instead of doing this analytically, people *sample* instead.

Given that the output of a quantum computation is probabilistic, we may be able to use this to our advantage in training generative models.

Boltzmann machines

One particular type of generative model is a *Boltzmann machine*. Boltzmann machines are a network of two types of nodes (hidden and visible). Nodes can be either on, or off.



The nodes are related to each other with weights $\{w_{ij}\}$, and can have their own bias terms $\{b_i\}$ as well.

Image credit: By Vera D - Created by Vera D, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=55007260`

We can assign an *energy* to a given configuration of nodes:

$$E(\mathbf{x}) = \sum_{ij} w_{ij} x_i x_j + \sum_i b_i x_i \tag{14}$$

The probability of the system being in a configuration is given by the Boltzmann factor

$$P(\mathbf{x}) = \frac{1}{Z} e^{\frac{-E(\mathbf{x})}{k_B T}}, \quad Z = \sum_{\mathbf{x}} e^{-\frac{E(\mathbf{x})}{k_B T}} \tag{15}$$

where T is an artificial temperature parameter.

Boltzmann machines

Given a set of training data, training a Boltzmann machine involves finding $\{w_{ij}\}$ and $\{b_i\}$ such that the probability distribution over the network matches closely with the probability distribution of the observed data.

Let θ represent the set of parameters of the network.

This is typically done by maximizing a log likelihood (or minimizing a negative log likelihood) in terms of the parameters:

$$\mathcal{L} = \sum_{d=1}^{D} \log(P(\mathbf{x}_d), \theta) \tag{16}$$

Boltzmann machines

The maximum likelihood estimation is done using gradient descent.

However performing gradient descent will involve computing the derivative of the probabilities with respect to the weights θ.

Recall that the probabilities are given by

$$P(\mathbf{x}) = \frac{1}{Z} e^{\frac{-E(\mathbf{x})}{k_B T}}, \quad Z = \sum_{\mathbf{x}} e^{-\frac{E(\mathbf{x})}{k_B T}} \tag{17}$$

If we don't know the true weights, how do we know the value of the partition function? It's not feasible to evaluate it for every possible configuration of $\mathbf{x}$!

Instead, we can sample from the distribution given by θ.

Where does the quantum hardware come in?

Recall that after running quantum annealing many times, we obtain a *distribution* of final energy states.

Given that annealing occurs at finite temperature, we might suppose that the distribution of those output states follows a Boltzmann distribution.

Then instead of sampling using a classical technique, we can feed an annealer the relevant parameters and perform the sampling physically!

Note: this is a VERY hand-wavy description of what's going on. Studies of the D-Wave have found that the distribution after finishing annealing is not quite a Boltzmann distribution, but at intermediate times during the anneal, it may be.
See: `https://www.dwavesys.com/sites/default/files/14-1013A-A_WP_Performance_Advantage_in_Quantum_Boltzmann_Sampling.pdf`,
`https://arxiv.org/abs/1503.04216`

Summary

- Problems that are too large for a D-Wave can be broken into smaller problems, mapped to hardware using minor embeddings, and then be put back together
- Thus far, there has been no concrete speedup demonstrated for quantum annealing in HEP applications
- Quantum machine learning (in its various forms) is an area to keep an eye on in the near future

Concluding remarks

If you think any of your projects might benefit from quantum computing, feel free to come and talk with me!

Please help me improve these lectures for the next iteration - a link to a feedback form will be e-mailed to everyone who signed up.

Finally, thank you to David Morrissey and Oliver Stelzer-Chilton for helping to organize these lectures, and Lukas Chrostowski and UBC's Quantum Computing Research Cluster for providing the free lunch.