

Building quantum applications with D-Wave's Leap

Alexander Condello

Before we start....

- (Almost) everything I will talk about is in open source
- Documentation is online and in LEAP
- We want feedback!
 - Community
 - Issues
 - Pull requests
- Your feedback helps us prioritize

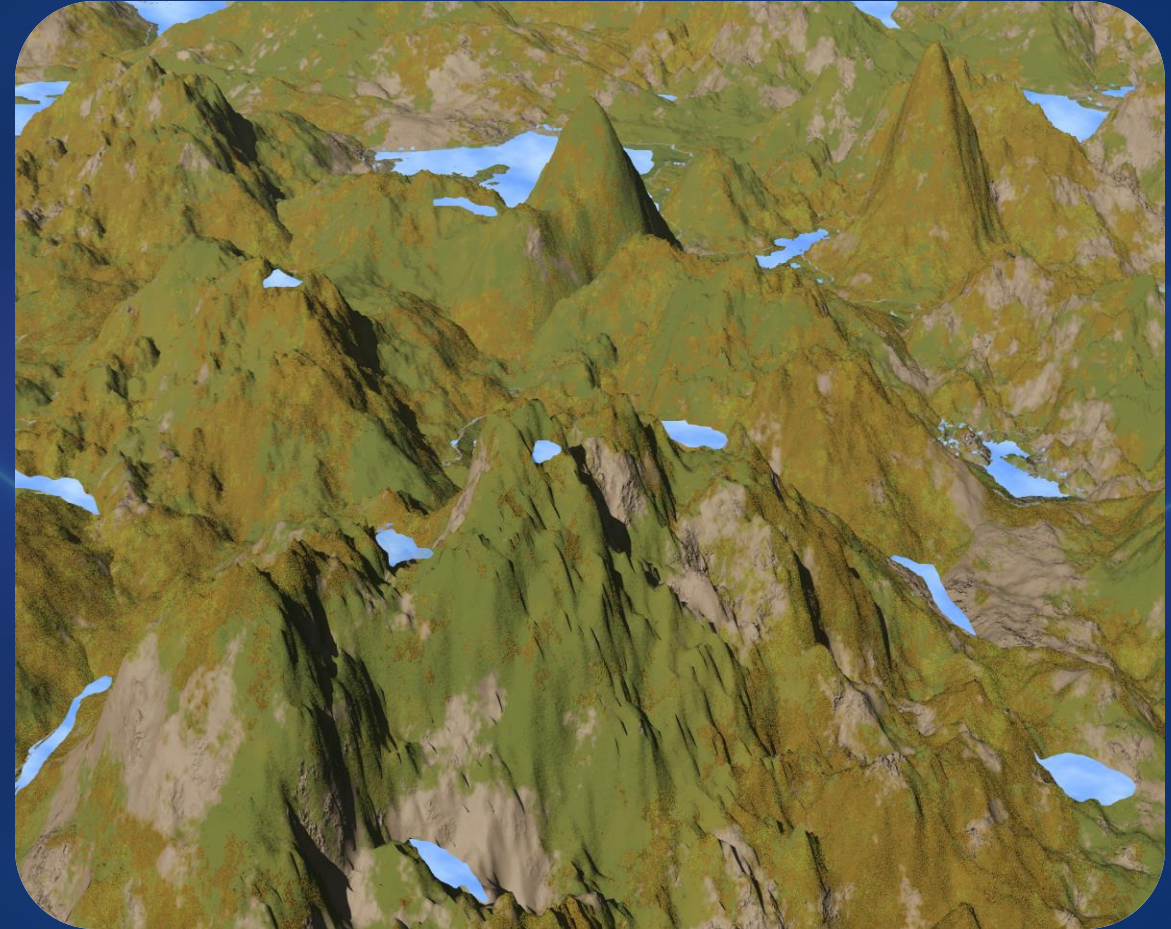


Demo

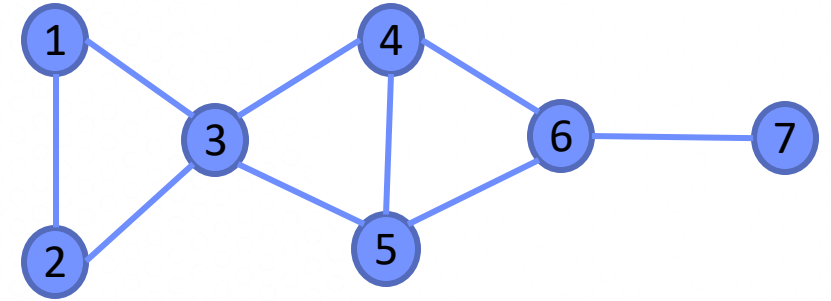
Solving Problems with BQMs

Landscape metaphor

Space of solutions defines an energy landscape and the best solution is the lowest valley

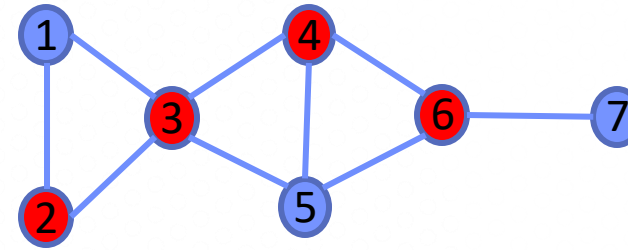
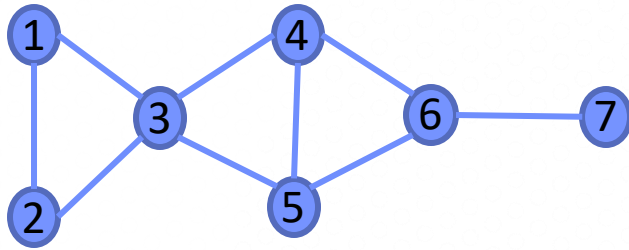


Example



- Given:
 - Network of pipelines
- What do we want:
 - A (minimum) set of junctions from which we can monitor every pipeline segment

Example



Junctions for monitoring pipes



Goal:

Nodes that *cover* every
edge

A vertex cover.

With the Ocean tools...

```
import networkx as nx

import dwave_networkx as dnx

from dwave.system import DWaveSampler, EmbeddingComposite

sampler = EmbeddingComposite(DWaveSampler())

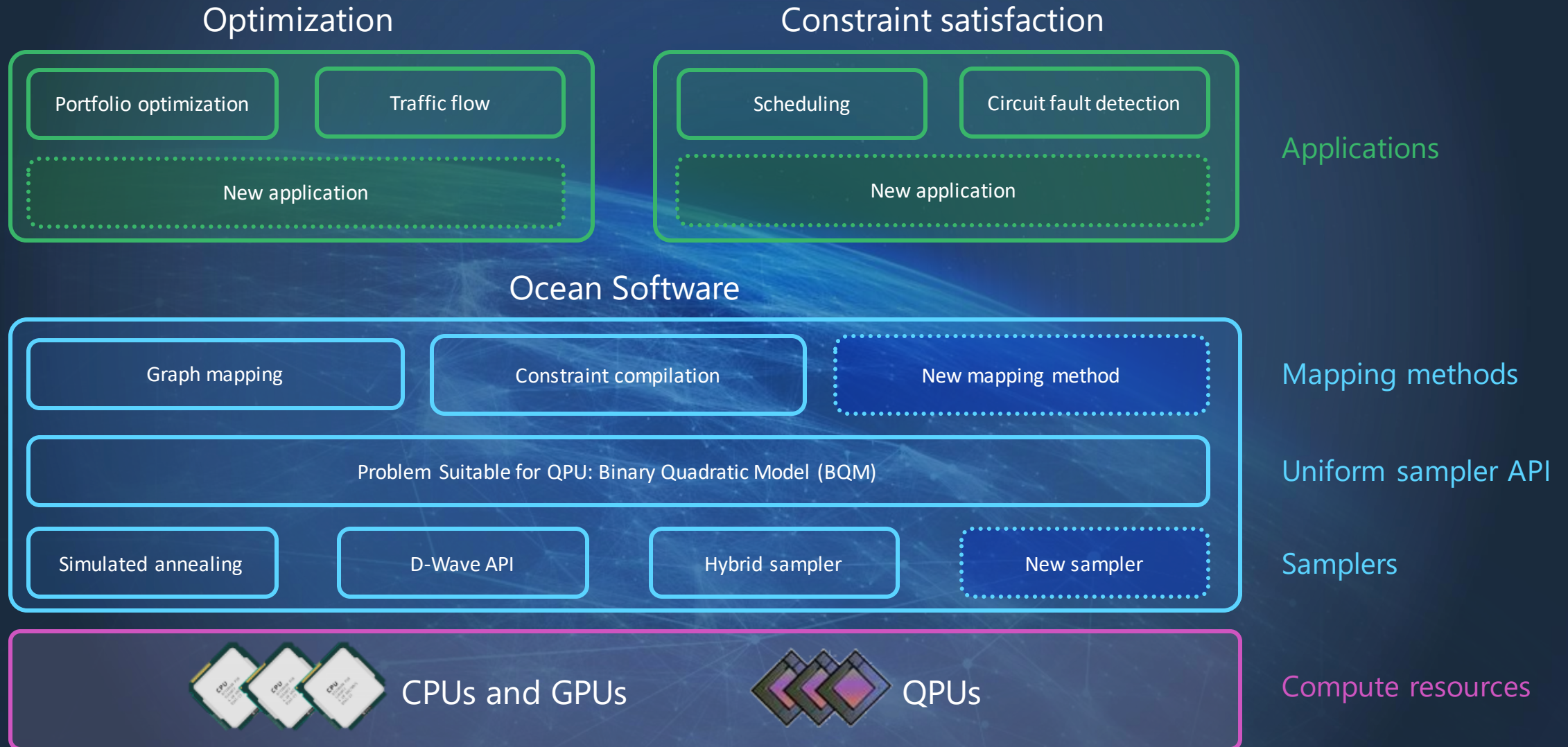
G = nx.Graph()

G.add_edges_from([(1,2),(1,3),(2,3),(3,4),(3,5),(4,5),(4,6),(5,6),(6,7)])

cover = dnx.min_vertex_cover(G, sampler=sampler)
```


Ocean™ software development kit

Suite of open-source Python tools on the D-Wave GitHub repository



With the Ocean tools...

```
import networkx as nx

import dwave_networkx as dnx

from dwave.system import DWaveSampler, EmbeddingComposite

sampler = EmbeddingComposite(DWaveSampler())

G = nx.Graph()

G.add_edges_from([(1,2),(1,3),(2,3),(3,4),(3,5),(4,5),(4,6),(5,6),(6,7)])

cover = dnx.min_vertex_cover(G, sampler=sampler)
```

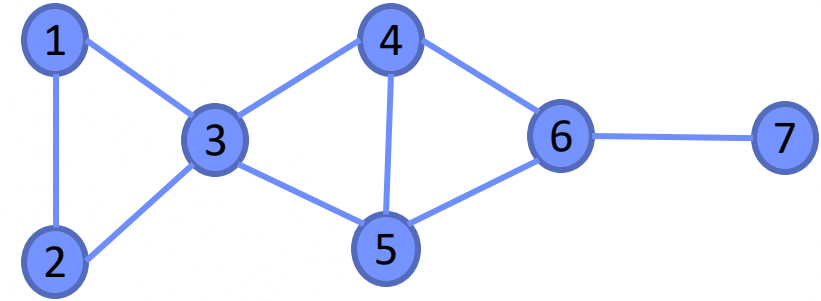

Binary Quadratic Model

$$E(v) = \sum_{i,j} v_i v_j a_{i,j} + \sum_i v_i b_i + c$$

$$a_{i,j}, b_i, c \in \mathbb{R}$$

$$v_i \in \{-1, +1\} \text{ or } v_i \in \{0, 1\}$$

Binary Quadratic Model and pipelines



Binary Variables

- Each junction either has a sensor or no

$$E(v) = \alpha \sum_{v \in V} v + \sum_{u, v \in E} (1 - u)(1 - v)$$

Pairwise interactions

- Every edge needs a sensor

Linear optimization

- Minimum number of sensors

Materials Properties

Atomic magnetometer

Solid state materials simulation

Quantum molecular dynamics

Quantum chemistry computation

200+
EARLY
APPLICATIONS

Finding Higgs Boson

Image recognition

Tree cover classifier

DNA binding

Individual cancer drugs

Machine Learning

Optimization

Radiotherapy

Multi-period portfolios

Satellite placement

Traffic flow

Internet ad placement

Formation of Terrorist Networks

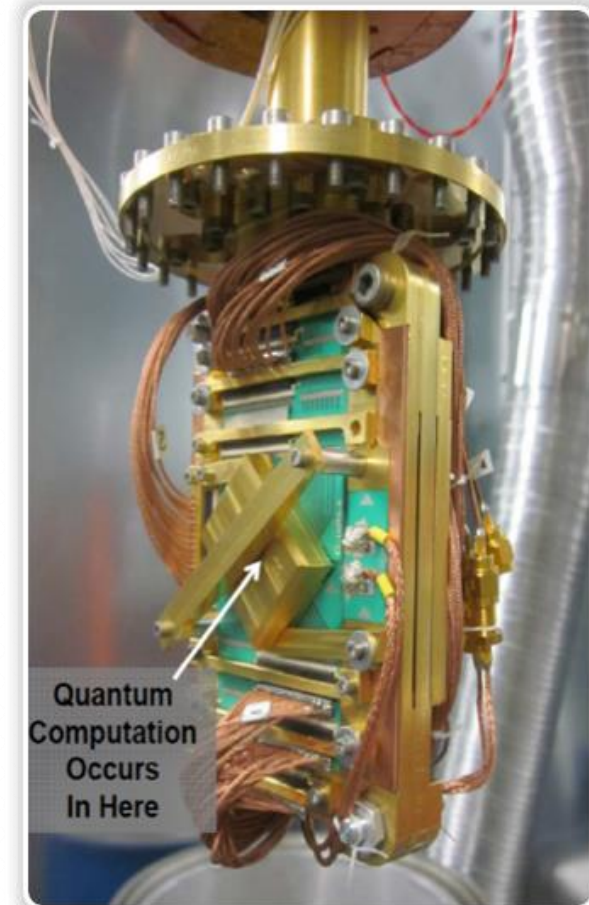
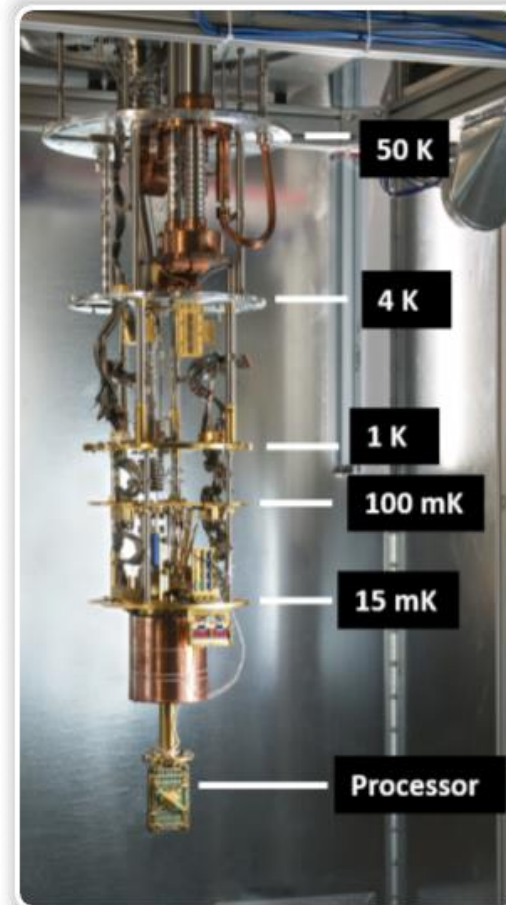
Fault detection in circuits

Facial recognition

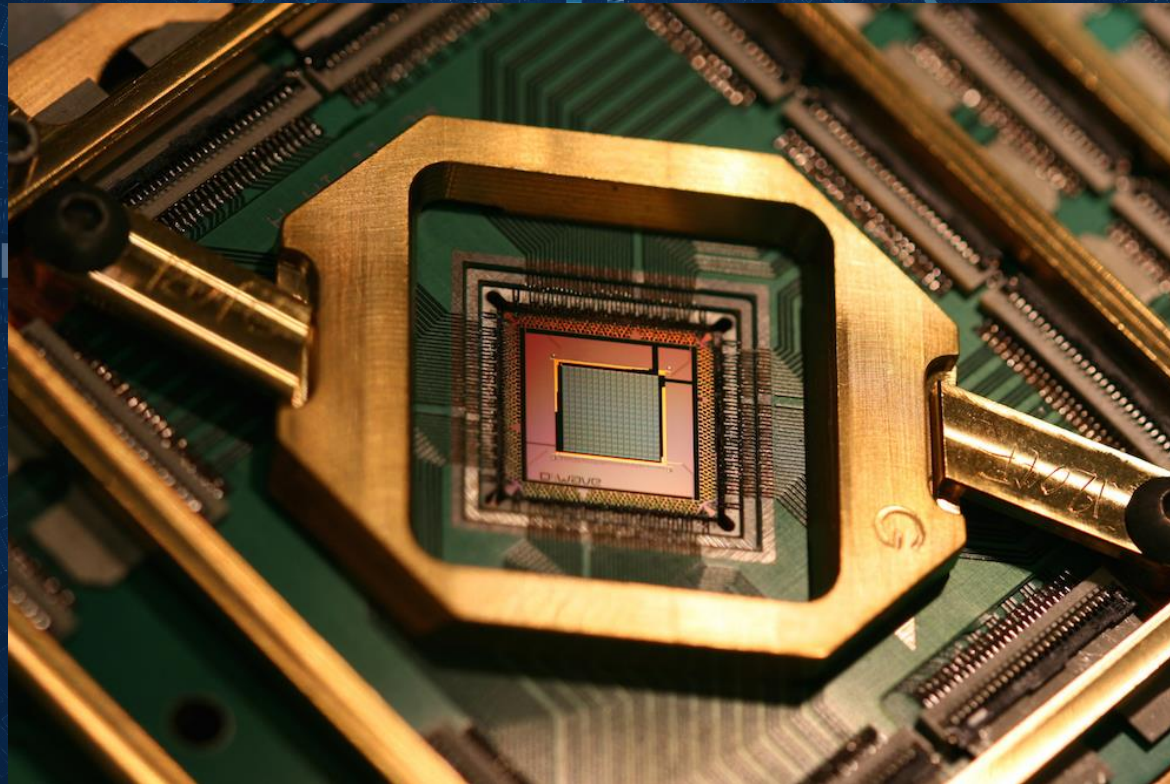
Cyber Security & Fault Detection

BQM to QMI

The D-Wave System



Quantum Processing Unit (QPU)



Ising Hamiltonian

$$H_{\text{ISING}} = -\frac{A(s)}{2} \left(\sum_i \sigma_x^{(i)} \right) + \frac{B(s)}{2} \left(\sum_i h_i \sigma_z^{(i)} + \sum_{i>j} J_{i,j} \sigma_z^{(i)} \sigma_z^{(j)} \right)$$

Quantum Machine
Instruction (QMI)

Binary Quadratic Model

$$E(v) = \sum_{i,j} v_i v_j a_{i,j} + \sum_i v_i b_i + c$$

$$a_{i,j}, b_i, c \in \mathbb{R}$$

$$v_i \in \{-1, +1\} \text{ or } v_i \in \{0, 1\}$$

Quantum Machine Instruction (QMI)

$$E(v) = \sum_{i,j} v_i v_j a_{i,j} + \sum_i v_i b_i + c$$

+ Rules

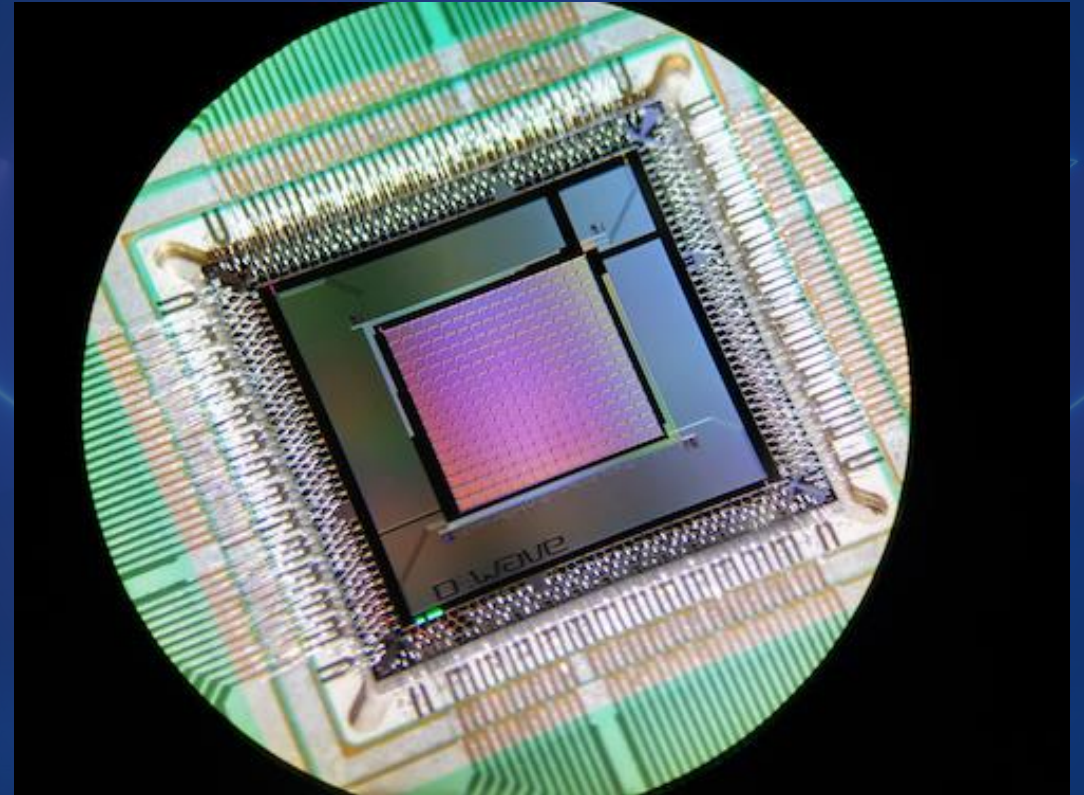
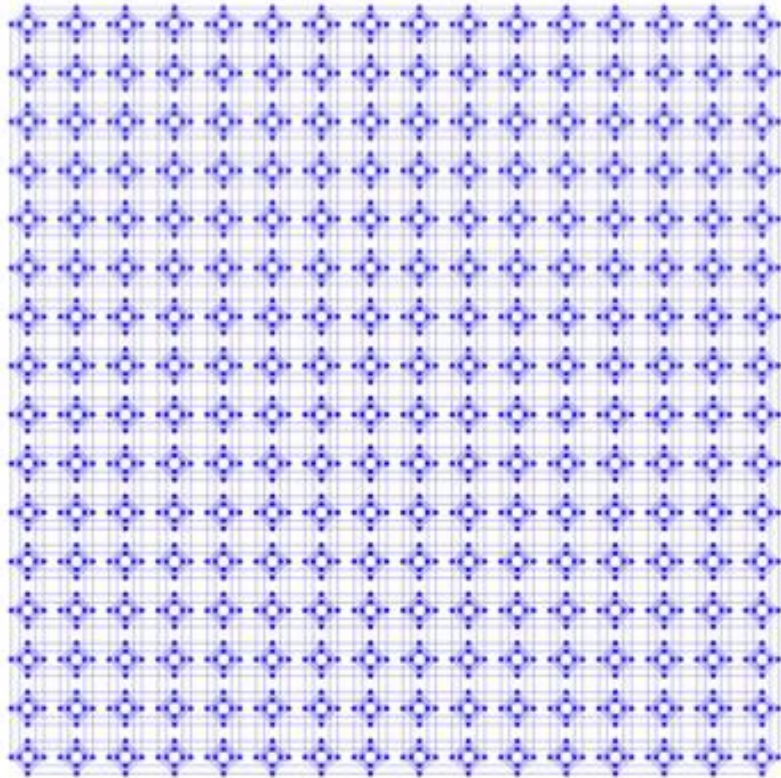
$$v_i \in \{-1, 1\}$$

$$a_{i,j} \in [-2, 1], b_i \in [-1, 1], c \in \mathbb{R}$$

Hardware structured

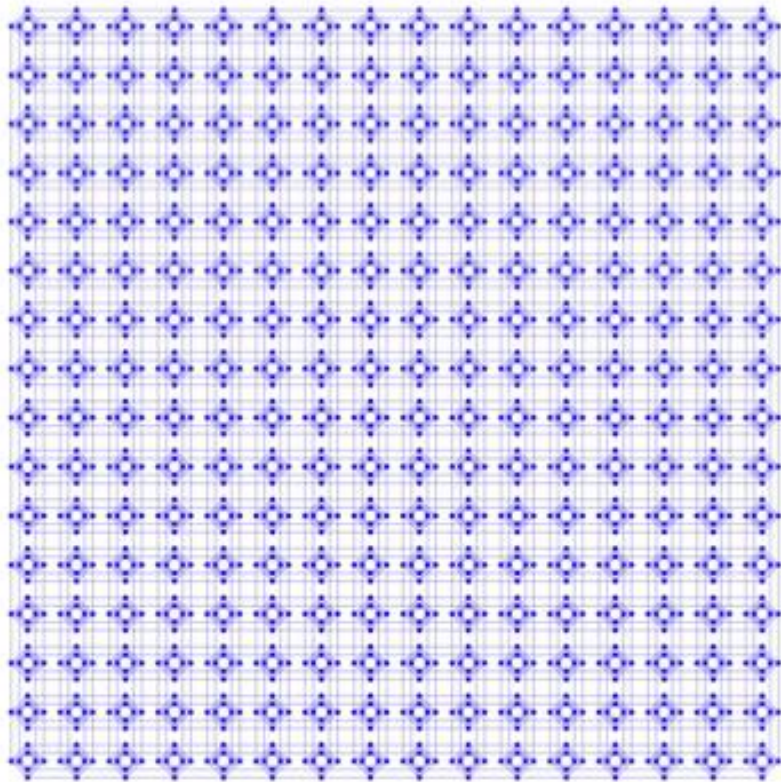
Qubit Layout

2000Q



Working Graphs

2000Q



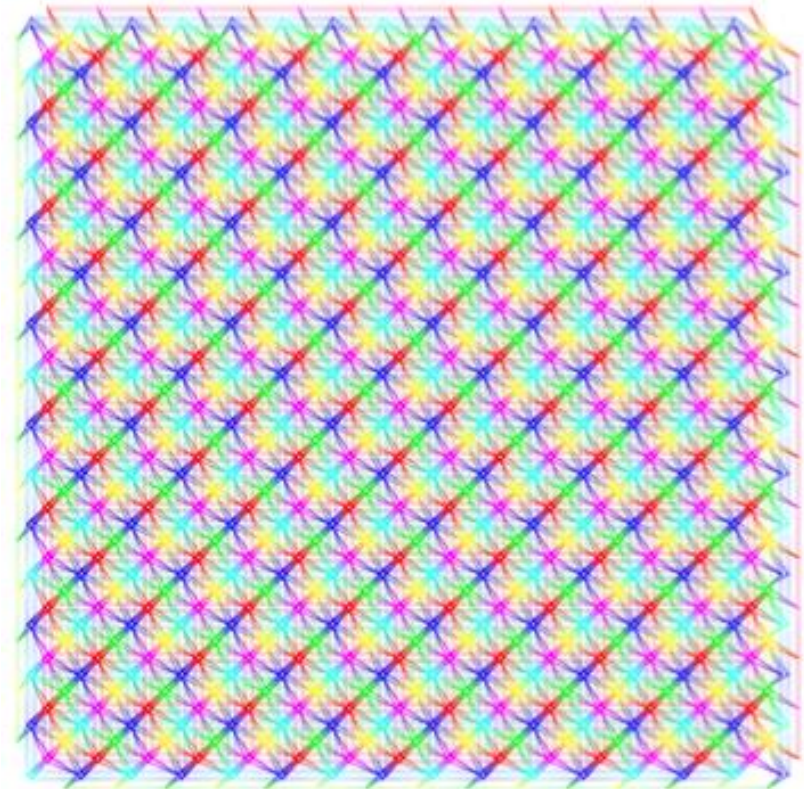
2000

6000

Qubits

Couplers

Advantage

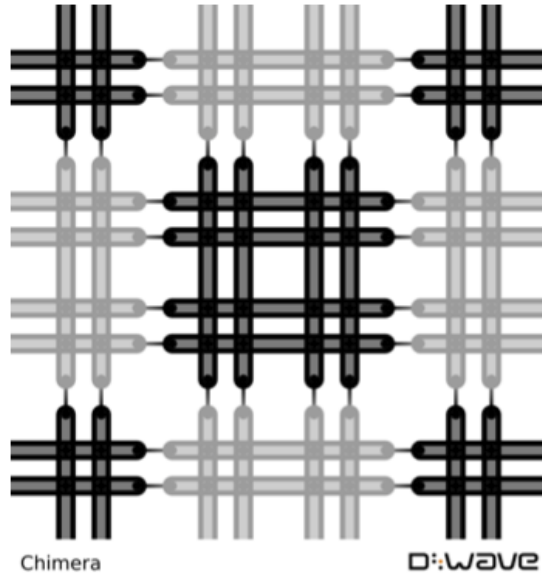


5000

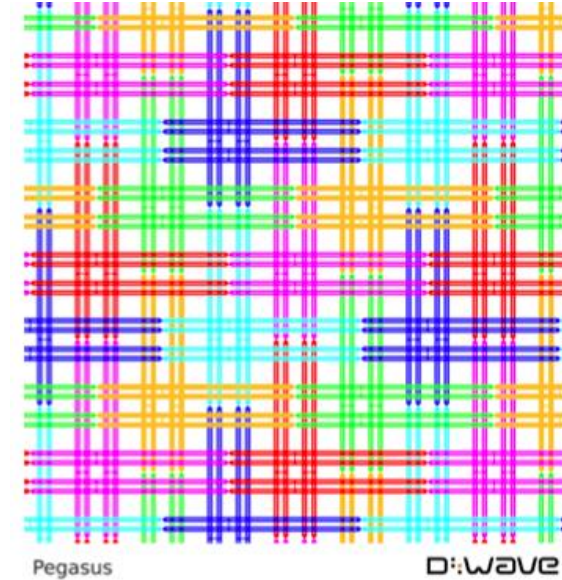
40000

A Closer Look...

2000Q



Advantage



6

Average Degree

15

Hybrid Algorithm Development

dwave-hybrid

- Hybrid Asynchronous Decomposition Sampler framework
 - Minimal, Python, solver/sampler-building framework, built atop Ocean tools
 - Leverages **quantum** and **classical** resources
 - Independent parts are executed **concurrently**
 - Problems are **broken into pieces** that fit the compute resources
 - Uses sample sets (probabilistic approach)

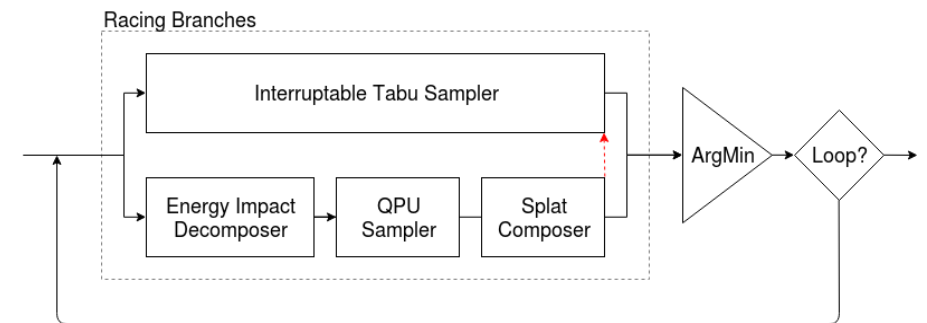
Motivation

Algorithm 1 Partitioning algorithm implemented by qbsolv

```
1: Input: QUBO instance
2: # best_energy is the lowest value found to date
3: # best_solution is the solution bit vector corresponding to the lowest value so far
4: # index is the indices of the bits in the solution, sorted from
5: #   most to least impact on value
6:
7: # Get initial estimate of minimum value and backbone
8: solution ← random 0/1 vector
9: (best_energy, best_solution) ← TabuSearch(QUBO, solution)
10: index ← OrderByImpact(QUBO, best_solution)
11: passCount ← 0
12: solution ← best_solution
13: while passCount < numRepeats do
14:   change ← false
15:   for i = 0; i < fraction * Size(QUBO); i += subQUBOSize do
16:     # select subQUBO with other variables clamped
17:     sub_index ← i : i + subQUBOSize - 1
18:     subQUBO ← Clamp(QUBO, solution, index[sub_index])
19:     (sub_energy, sub_solution) ← DWaveSearch(subQUBO, solution)
20:     # project onto full solution
21:     if (solution[sub_index] ≠ sub_solution) then
22:       solution[sub_index] ← sub_solution
23:       change ← true
24:   end if
25: end for
26: if not change then
27:   Randomize(solution[0 : i - 1])
28: end if
29: (energy, solution) ← TabuSearch(QUBO, solution)
30: if energy < best_energy then
31:   best_energy ← energy
32:   best_solution ← solution
33:   passCount ← 0
34: else
35:   passCount ++
36: end if
37: index ← OrderByImpact(QUBO, solution)
38: end while
39: Output: best_energy, best_solution
```

```
72 double evaluate(int8_t *const solution, const uint qubo_size, const double **const qubo, double *const flip_cost) {
73   double result = 0.0;
74
75   for (uint ii = 0; ii < qubo_size; ii++) {
76     double row_sum = 0.0;
77     double col_sum = 0.0;
78
79     // qubo an upper triangular matrix, so start right of the diagonal
80     // for the rows, and stop at the diagonal for the columns
81     for (uint jj = ii + 1; jj < qubo_size; jj++)
82       if (solution[jj]) row_sum += qubo[ii][jj];
83
84     for (uint jj = 0; jj < ii; jj++)
85       if (solution[jj]) col_sum += qubo[jj][ii];
86
87     // If the variable is currently 1, then by flipping it we
88     // will change the value by 2 * (row_sum + col_sum).
89     // If (CPSECONDS > timeout), then negate the value.
90     // Continue while = false;
91   } // end of outer loop
92
93   // all done print results if needed and free allocated arrays
94   if (WriteMatrix_) print_solution_and_qubo(Qbest, qubo_size, qubo);
95
96   if (Verbose_ == 0) {
97     Qbest = &solution_list[Qindex[0]][0];
98     best_energy = energy_list[Qindex[0]];
99     // printf("evaluated solution %8.2lf\n",
100     //       sign * Simple_evaluate(Qbest, qubo_size, (const double **)qubo));
101     // print_output(qubo_size, Qbest, numPartCalls, best_energy * sign, CPSECONDS, param);
102   }
103   free(solution);
104   free(tabu_solution);
105   free(flip_cost);
106   free(index);
107   free(TabuK);
108   free(Pcompress);
109   return;
110 }
```

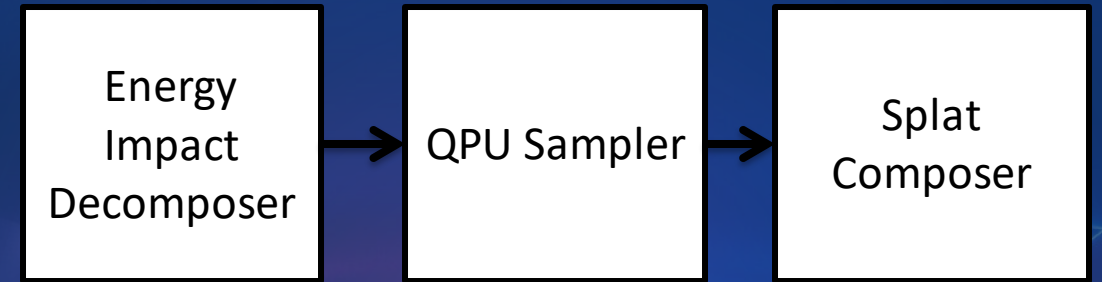
```
Loop(RacingBranches(
  InterruptableTabuSampler(),
  EnergyImpactDecomposer(size=50)
  | QPUSubproblemAutoEmbeddingSampler()
  | SplatComposer()
) | ArgMin())
```



Decomposition

Example

- Find a sub-problem with a high energy impact and solve that on the QPU

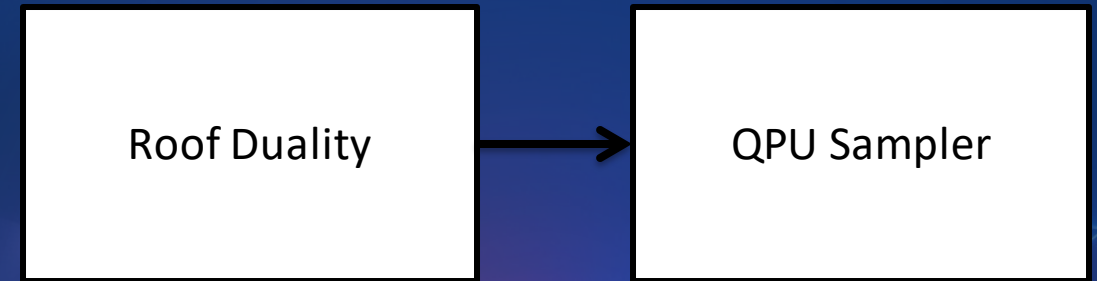


```
EnergyImpactDecomposer(size=50)  
| QPUSubproblemAutoEmbeddingSampler()  
| SplatComposer()
```


Pre-processing

Example

- Use roof duality to determine and fix the assignments of some variables in polynomial time



```
RoofDualityDecomposer()  
| QPUSubproblemAutoEmbeddingSampler()  
| SplatComposer()
```

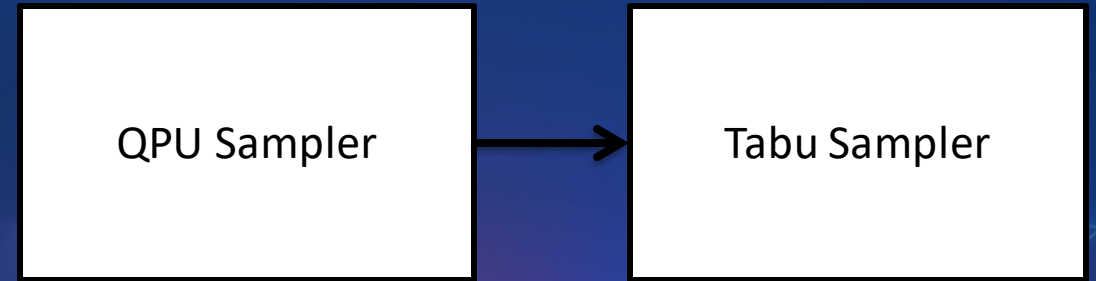
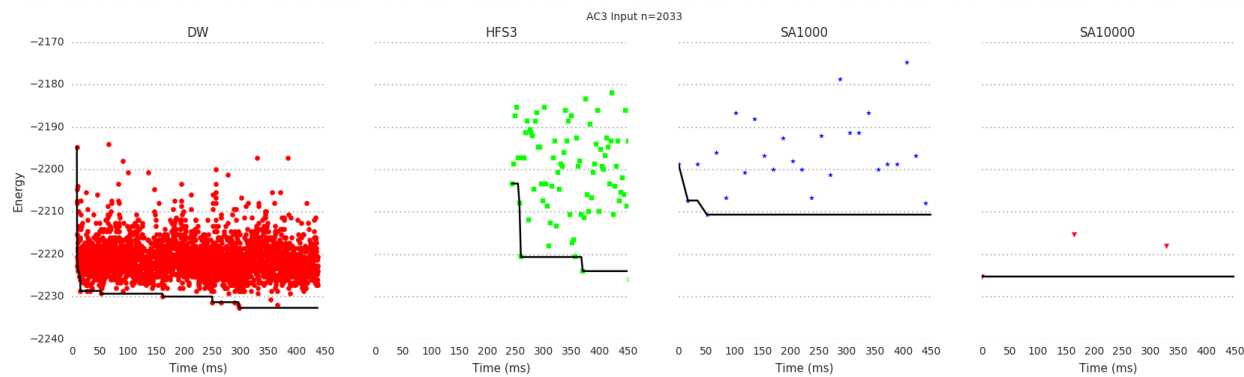
Boros, E., P.L. Hammer, G. Tavares. Preprocessing of Unconstrained Quadratic Binary Optimization. Rutcor Research Report 10-2006, April, 2006

Boros, E. P.L. Hammer. Pseudo-Boolean optimization. Discrete Applied Mathematics 123, 2002, pp. 155-225

Post-processing

Example

- Use the QPU to seed another classical algorithm



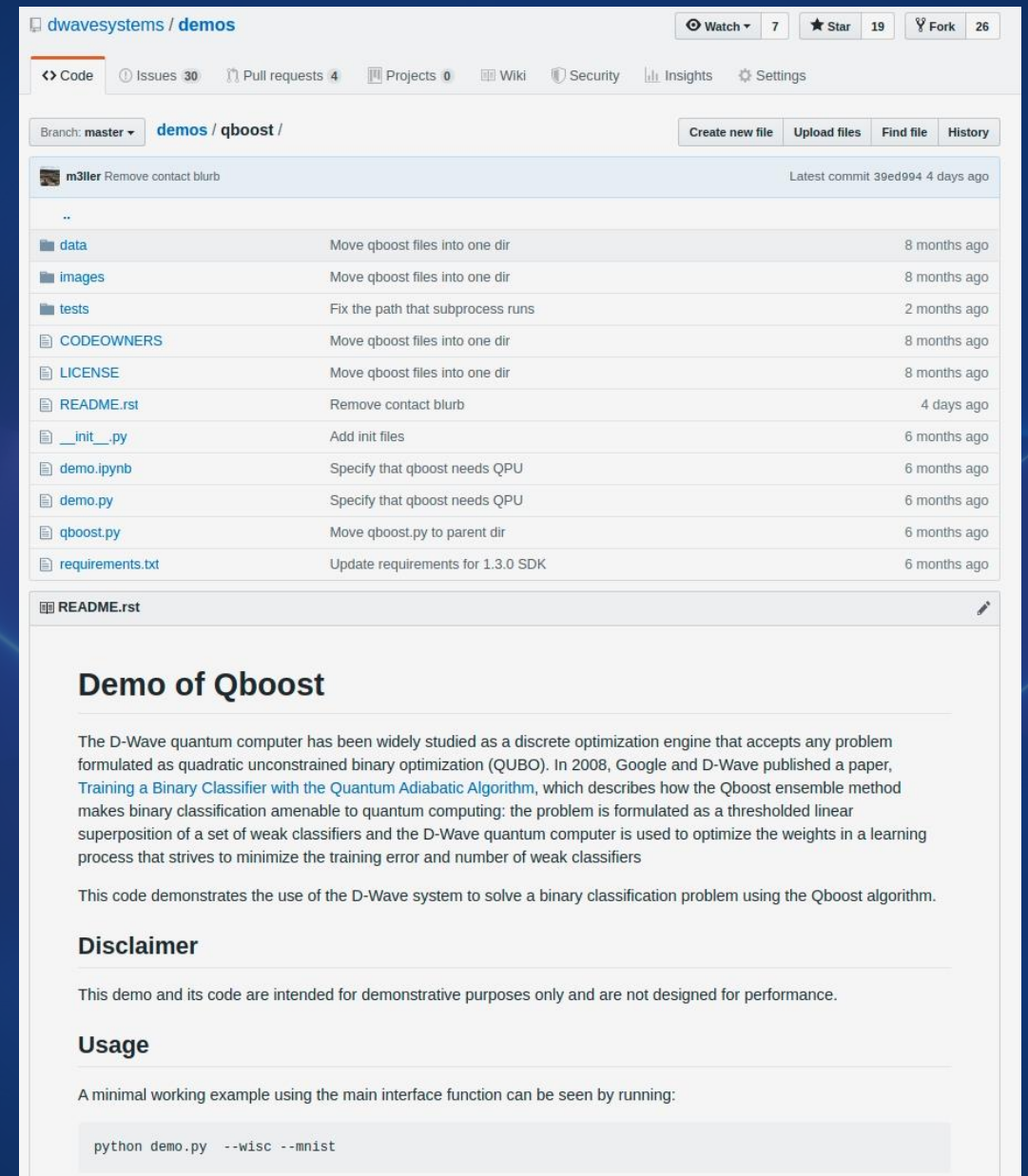
QPUSubproblemAutoEmbeddingSampler()
| TabuSubproblemSampler()

Meta-algorithms

Example

- Use a quantum computer to build a strong classifier from a selection of weak classifiers
- Co-developed with Google to train image classifiers for cars

Neven, Denchev, Rose, Macready. Training a Large Scale Classifier with the Quantum Adiabatic Algorithm. <https://arxiv.org/abs/0912.0779>, Dec 2009



dwavesystems / demos

Watch 7 Star 19 Fork 26

Code Issues 30 Pull requests 4 Projects 0 Wiki Security Insights Settings

Branch: master demos / qboost /

Create new file Upload files Find file History

m3ller Remove contact blurp Latest commit 39ed994 4 days ago

..		
data	Move qboost files into one dir	8 months ago
images	Move qboost files into one dir	8 months ago
tests	Fix the path that subprocess runs	2 months ago
CODEOWNERS	Move qboost files into one dir	8 months ago
LICENSE	Move qboost files into one dir	8 months ago
README.rst	Remove contact blurp	4 days ago
__init__.py	Add init files	6 months ago
demo.ipynb	Specify that qboost needs QPU	6 months ago
demo.py	Specify that qboost needs QPU	6 months ago
qboost.py	Move qboost.py to parent dir	6 months ago
requirements.txt	Update requirements for 1.3.0 SDK	6 months ago

Demo of Qboost

The D-Wave quantum computer has been widely studied as a discrete optimization engine that accepts any problem formulated as quadratic unconstrained binary optimization (QUBO). In 2008, Google and D-Wave published a paper, [Training a Binary Classifier with the Quantum Adiabatic Algorithm](#), which describes how the Qboost ensemble method makes binary classification amenable to quantum computing: the problem is formulated as a thresholded linear superposition of a set of weak classifiers and the D-Wave quantum computer is used to optimize the weights in a learning process that strives to minimize the training error and number of weak classifiers

This code demonstrates the use of the D-Wave system to solve a binary classification problem using the Qboost algorithm.

Disclaimer

This demo and its code are intended for demonstrative purposes only and are not designed for performance.

Usage

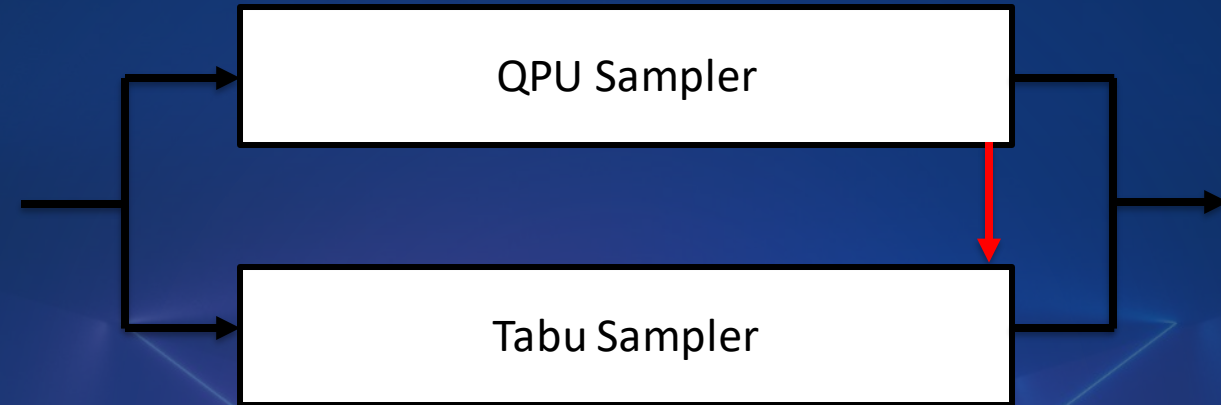
A minimal working example using the main interface function can be seen by running:

```
python demo.py --wisc --mnist
```

Racing

Example

- Run tabu while waiting on the result from the QPU



```
RacingBranches(  
    InterruptableTabuSampler(),  
    EnergyImpactDecomposer(size=50)  
    | QPUSubproblemAutoEmbeddingSampler()  
    | SplatComposer()  
    ) | ArgMin())
```