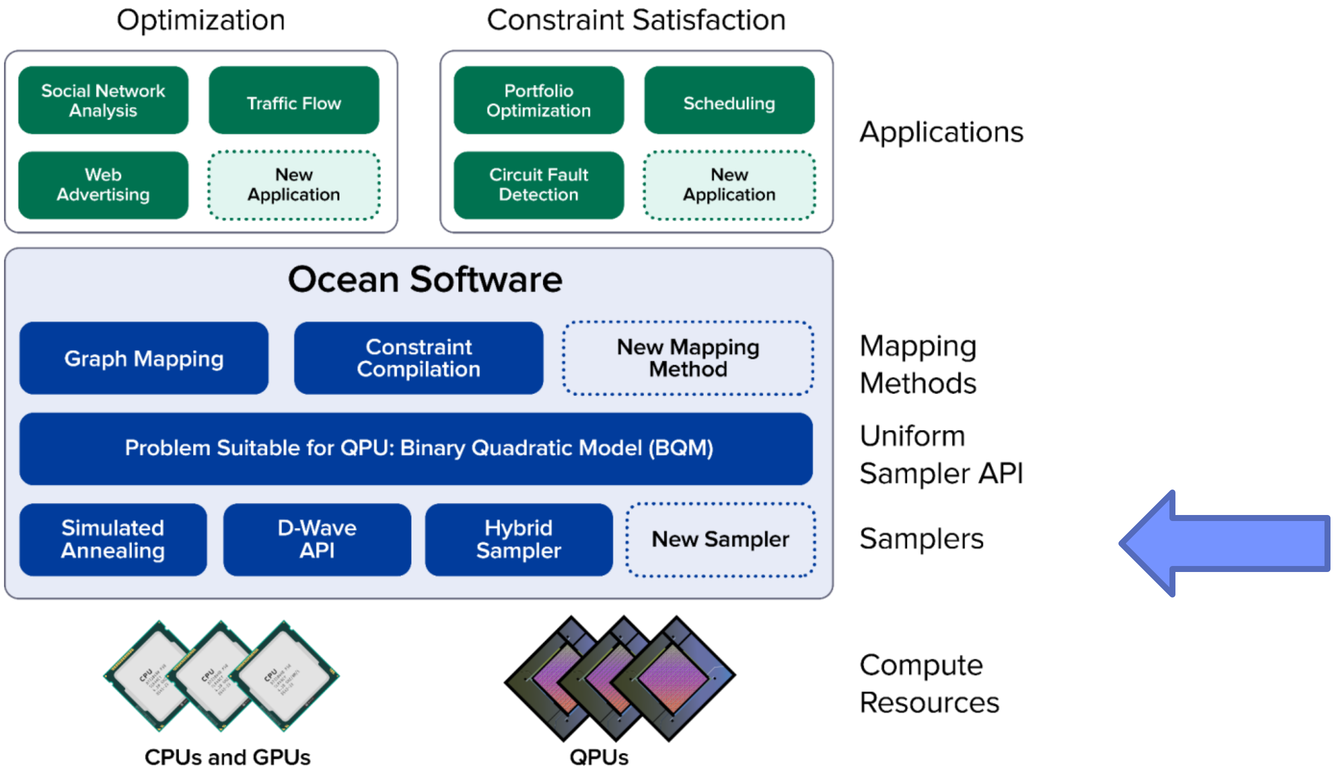
D-Wave Hybrid

An Overview

Radomir Stevanovic

D-Wave

Ocean Software Stack



© D-Wave Systems Inc.

github.com/dwavesystems/**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)

© D-Wave Systems Inc.

D-Wave The Quantum Computing Company™

Motivation

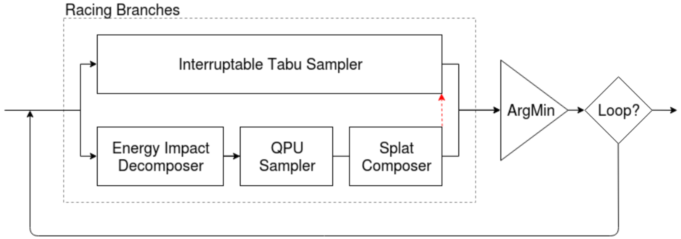
Algorithm 1 Partitioning algorithm implemented by `qbsolv`

```
Input: QUBO instance
# best_energy is the lowest value found to date
# best_solution is the solution bit vector corresponding to the lowest value so far
# index is the indices of the bits in the solution, sorted from
#    most to least impact on value

# Get initial estimate of minimum value and backbone
solution ← random 0/1 vector
(best_energy, best_solution) ← TabuSearch(QUBO, solutio
index ← OrderByImpact(QUBO, best_solution)
passCount ← 0
solution ← best_solution
while passCount < numRepeats do
    change ← false
    for i = 0; i < fraction * Size(QUBO); i += subQUBOSize
        # select subQUBO with other variables clamped
        sub_index ← i : i+subQUBOSize−1
        subQUBO ← Clamp(QUBO, solution, index[sub_index])
        (sub_energy, sub_solution) ← DWaveSearch(subQUB
        # project onto full solution
        if (solution[sub_index] ≠ sub_solution) then
            solution[sub_index] ← sub_solution
            change ← true
        end if
    end for
    if not change then
        Randomize(solution[0 : i − 1])
    end if
    (energy, solution) ← TabuSearch(QUBO, solution)
    if energy < best_energy then
        best_energy ← energy
        best_solution ← solution
        passCount ← 0
    else
        passCount ++
    end if
    index ← OrderByImpact(QUBO, solution)
end while
Output: best_energy, best_solution
```

```c
double evaluate(int8_t *const solution, const uint qubo_size, const double **const qubo, double *const flip_cost) {
    double result = 0.0;

    for (uint ii = 0; ii < qubo_size; ii++) {
        double row_sum = 0.0;
        double col_sum = 0.0;

        // qubo an upper triangular matrix, so start right of the diagonal
        // for the rows, and stop at the diagonal for the columns
        for (uint jj = ii + 1; jj < qubo_size; jj++)
            if (solution[jj]) row_sum += qubo[ii][jj];

        for (uint jj = 0; jj < ii; jj++)
            if (solution[jj]) col_sum += qubo[jj][ii];
```

```c
            }
        }  // end of outer loop

        // all done print results if needed and free allocated arrays
        if (WriteMatrix_) print_solution_and_qubo(Qbest, qubo_size, qubo);

        if (Verbose_ == 0) {
            Qbest = &solution_list[Qindex[0]][0];
            best_energy = energy_list[Qindex[0]];
            // printf(" evaluated solution %8.2lf\n",
            //     sign * Simple_evaluate(Qbest, qubo_size, (const double **)qubo));
            print_output(qubo_size, Qbest, numPartCalls, best_energy * sign, CPSECONDS, param);
        }

        free(solution);
        free(tabu_solution);
        free(flip_cost);
        free(index);
        free(TabuK);
        free(Pcompress);

        return;
    }
```

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

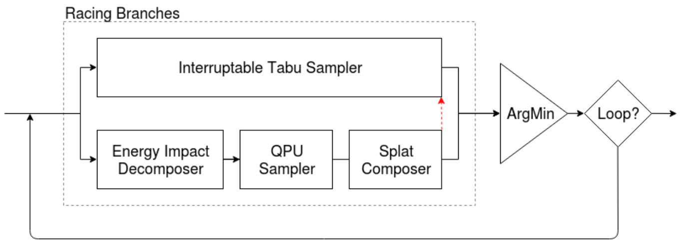


© D-Wave Systems Inc.

Goals

- Code outlines/visualizes the algorithm
- Code is easy to tweak, extend, experiment, benchmark and profile
- Simplicity is balanced with expressiveness
- Library of building blocks provided, extendable by developers

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



© D-Wave Systems Inc.

D-Wave The Quantum Computing Company™

Demo

```
In [1]: import dimod, hybrid
   ...:
   ...: bqm = dimod.BinaryQuadraticModel({}, {'ab': 1, 'bc': -1, 'ca': 1}, 0, dimod.SPIN)
   ...: state = hybrid.State.from_sample(hybrid.min_sample(bqm), bqm)
   ...:
   ...: workflow = hybrid.Loop(hybrid.RacingBranches(
   ...:     hybrid.InterruptableTabuSampler(),
   ...:     hybrid.EnergyImpactDecomposer(size=1)
   ...:     | hybrid.QPUSubproblemAutoEmbeddingSampler()
   ...:     | hybrid.SplatComposer()
   ...: ) | hybrid.ArgMin(), convergence=3)
```

```
In [3]: hybrid.profiling.print_structure(workflow)
Loop
  Branch
    RacingBranches
      InterruptableTabuSampler
      Branch
        EnergyImpactDecomposer
        QPUSubproblemAutoEmbeddingSampler
        SplatComposer
    ArgMin
```

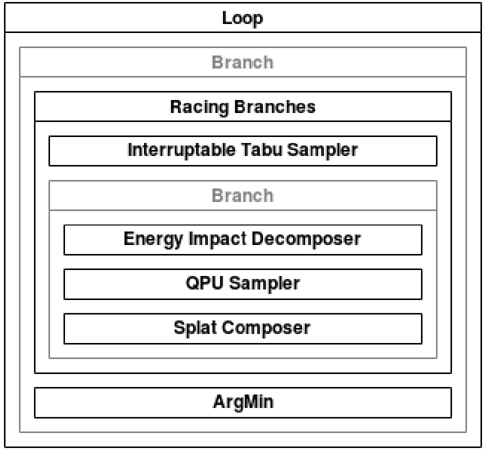
```
In [2]: state
Out[2]:
{'problem': BinaryQuadraticModel({'b': 0, 'a': 0, 'c': 0}, {('a', 'c'): 1, ('a', 'b'): 1, ('b', 'c'): -1}, 0, Vartype.SPIN),
 'samples': SampleSet(rec.array([([-1, -1, -1], 1., 1)],
          dtype=[('sample', 'i1', (3,)), ('energy', '<f8'), ('num_occurrences', '<i8')]), ['b', 'a', 'c'], {}, 'SPIN')}
```

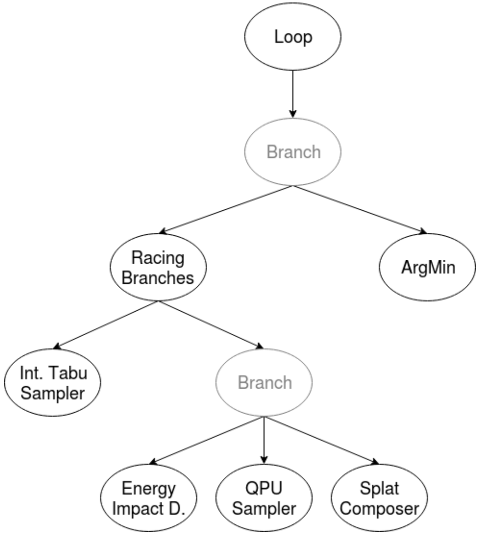
```
In [4]: f = workflow.run(state)

In [5]: f
Out[5]: <Future at 0x7f5e777a9710 state=running>

In [6]: f
Out[6]: <Future at 0x7f5e777a9710 state=finished returned State>

In [7]: f.result()
Out[7]:
{'problem': BinaryQuadraticModel({'b': 0, 'a': 0, 'c': 0}, {('a', 'c'): 1, ('a', 'b'): 1, ('b', 'c'): -1}, 0, Vartype.SPIN),
 'samples': SampleSet(rec.array([([-1,  1, -1], -3., 1)],
          dtype=[('sample', 'i1', (3,)), ('energy', '<f8'), ('num_occurrences', '<i8')]), ['b', 'a', 'c'], {}, 'SPIN')}
```


D-Wave The Quantum Computing Company™

Framework Primitive: Runnable Type

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

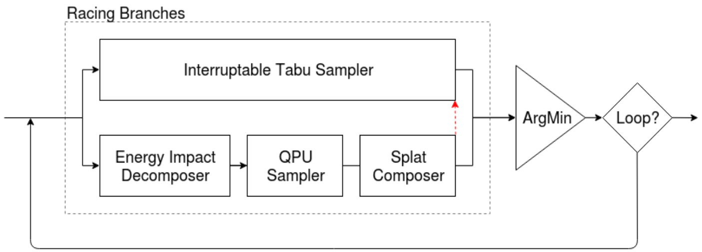




- All components implement the `Runnable` type
- Act on input `State`(s), produce output `State`(s)
- Execute asynchronously (`.run()` and `.stop()`)
- Composable top-down (tree); traits constrain connectivity; profiled by default

Framework Primitive: State

```
In [4]: f = workflow.run(state)

In [5]: f
Out[5]: <Future at 0x7f5e777a9710 state=running>

In [6]: f
Out[6]: <Future at 0x7f5e777a9710 state=finished returned State>

In [7]: f.result()
Out[7]:
{'problem': BinaryQuadraticModel({'b': 0, 'a': 0, 'c': 0}, {('a', '
 'samples': SampleSet(rec.array([([-1,  1, -1], -3., 1)],
           dtype=[('sample', 'i1', (3,)), ('energy', '<f8'), ('num_
```



- Immutable mapping type
- Passed between Runnable components, wrapped in Future
- Carries the problem, subproblem, samples, etc.
- Compliance with component's traits checked during runtime

Modifying Workflow Parameters

```
subproblem = EnergyImpactDecomposer(size=50, rolling_history=0.15)

subsampler = QPUSubproblemAutoEmbeddingSampler()
           | SplatComposer()

iteration = RacingBranches(
    InterruptableTabuSampler(),
    subproblems | subsampler
) | ArgMin()

workflow = Loop(iteration, max_iter=1e3, convergence=3)
```

- Solve subproblems (of size 50 variables), at different points (samples), one per iteration
  - Keep unrolling (deconstructing) up to 15% of the input problem variables (in order of energy impact)
- Upper bound on loop count, terminate if no improvement after 3 iterations

© D-Wave Systems Inc.

D-Wave The Quantum Computing Company™

Modifying Workflow Structure

```
subproblems = Unwind(
    EnergyImpactDecomposer(size=50, rolling_history=0.15, silent_rewind=False))

subsampler = Map(QPUSubproblemAutoEmbeddingSampler())
             | Reduce(Lambda(merge_substates))
             | SplatComposer()

iteration = RacingBranches(
    InterruptableTabuSampler(),
    subproblems | subsampler
) | ArgMin()

workflow = Loop(iteration, max_iter=1e3, convergence=3)
```

- Deconstruct 15% of the problem into multiple subproblems (at the same sample)
  - Solve them all in parallel on the QPU
  - Merge subsamples
  - Compose with the original sample

© D-Wave Systems Inc.

D-Wave The Quantum Computing Company™

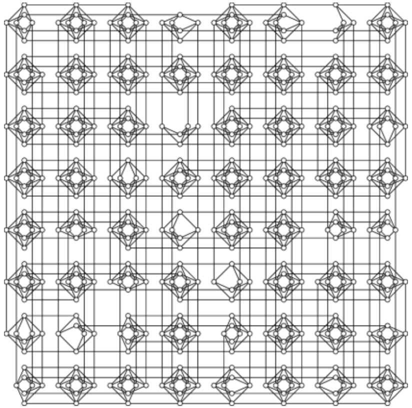
Modifying Workflow Structure

```
subproblems = Unwind(
    EnergyImpactDecomposer(size=50, rolling_history=0.15, silent_rewind=False))

qpu = Map(QPUSubproblemAutoEmbeddingSampler())
    | Reduce(Lambda(merge_substates))
    | SplatComposer()

random = Map(RandomSubproblemSampler())
       | Reduce(Lambda(merge_substates))
       | SplatComposer()

subsampler = Parallel(qpu, random, endomorphic=False) | ArgMin()

iteration = RacingBranches(
    InterruptableTabuSampler(),
    subproblems | subsampler
) | ArgMin()

workflow = Loop(iteration, max_iter=1e3, convergence=3)
```

- Deconstruct 15% of the problem into subproblems
  - Solve them all in parallel on the QPU
  - But also solve them using a second/classical subsampler (random here)
  - All in parallel

D-Wave The Quantum Computing Company™

On Problem Decomposition



- When problem doesn't fit the computing device
  - Memory, parallelism/cores, bit-length, GPU pipeline, QPU size/structure
- Tailored to problem class and purpose/device
  - "no free lunch"
  - no "right", or general, approach to problem decomposition
- No shortage of ideas for decomposition:
  - Energy based, connectivity/structure based...
- (Or hybrid solvers):
  - Based on tabu search, parallel tempering, dialectic search, branch and bound, diversity-preserving sampling, genetic algorithms...

© D-Wave Systems Inc.

Constructing Runnables

```
In [48]: class Sleeper(hybrid.Runnable):
    ...:     def next(self, state):
    ...:         import time
    ...:         time.sleep(3)
    ...:         return state
    ...:

In [49]: tabu = hybrid.InterruptableTabuSampler()

In [50]: workflow = hybrid.RacingBranches(tabu, Sleeper())

In [51]: result = workflow.run(state).result()

In [52]: workflow.timers
Out[52]:
{'dispatch': [6.561802001670003e-05],
 'dispatch.init': [6.191025022417307e-06],
 'dispatch.next': [3.0666631020139903],
 'dispatch.resolve': [1.605699071660638e-05]}
```

- Extend hybrid.Runnable's methods:
  - init(), next(), error(), halt()
- Implement a flow control block, a sampler, or a problem decomposer tailored to your problem (class)
- Share it!

© D-Wave Systems Inc.

The Quantum Computing Company™

Contributions Welcome

- https://github.com/dwavesystems/dwave-hybrid/issues
  - more samplers (parallel tempering, ICM, reverse anneal)
  - more decomposing strategies (e.g. BFS, PFS traversal in EID)
  - more composing strategies – better support for multiple samples per state (alternative to "best sample splat")
  - more flow control blocks
  - sample diversity-preserving sampleset pruning
  - CoW State
  - more Runnable Executors (celery, asyncio?)
- Developer survey
  - https://www.surveymonkey.com/r/LJM96GT

The Quantum Computing Company™