Using Leap's Hybrid Solvers

Introduces Leap's quantum-classical hybrid solvers and provides references to usage information.

> **Note**
>
> Not all accounts have access to this type of solver.

Leap's Hybrid Solvers

Leap's quantum-classical hybrid solvers are intended to solve arbitrary application problems formulated as quadratic models[1].

[1] Problems submitted directly to quantum computers are in the binary quadratic model (BQM) format, unconstrained with binary-valued variables and structured for the topology of the quantum processing unit (QPU). Hybrid solvers may accept arbitrarily structured quadratic models (QM), constrained or unconstrained, with real, integer, and binary variables.

These solvers, which implement state-of-the-art classical algorithms together with intelligent allocation of the quantum computer to parts of the problem where it benefits most, are designed to accommodate even very large problems. Leap's hybrid solvers enable you to benefit from D-Wave's deep investment in researching, developing, optimizing, and maintaining hybrid algorithms.

The generally available hybrid solvers depend on your Leap™ account. Check your Leap dashboard to see which hybrid solvers are available to you.

Generally-available hybrid solvers currently supported in Leap include:

- Hybrid BQM solver (e.g., `hybrid_binary_quadratic_model_version2`)
  Binary quadratic models (BQM) typically represent problems of decisions that could either be true (or yes) or false (no); for example, should an antenna transmit, or did a network node fail?

  These solvers accept arbitrarily structured, unconstrained problems formulated as BQMs, with any constraints typically represented through penalty models.

- Hybrid CQM solver (e.g., `hybrid_constrained_quadratic_model_version1`)
  Constrained quadratic models (CQM) typically represent problems that might include real, integer and/or binary variables and one or more constraints.

  These solvers accept arbitrarily structured problems formulated as CQMs, with any constraints represented natively.

- Hybrid DQM solver (e.g., `hybrid_discrete_quadratic_model_version1`)
  Discrete quadratic models (DQM) typically represent problems with several distinct options; for example, which shift should employee X work, or should the state on a map be colored red, blue, green, or yellow?

  These solvers accept arbitrarily structured, unconstrained problems formulated as DQMs, with any constraints typically represented through penalty models.

Contact D-Wave at sales@dwavesys.com if your application requires scale or performance that exceeds the currently advertised capabilities of Leap's generally available hybrid solvers.

Solver Properties and Parameters

The Solver Properties and Parameters Reference guide details limitations on problem size, the range of time allowed for solving problems, etc and input arguments that enable you to configure execution.

Solver Timing

The table below lists the timing information returned from Leap's quantum-classical hybrid solvers. Ocean users access to this information is from the `info` field in the dimod `SampleSet` class, as in the example below.

```
>>> import dimod
>>> from dwave.system import LeapHybridSampler
...
>>> sampler = LeapHybridSampler(solver={'category': 'hybrid'})
>>> bqm = dimod.generators.ran_r(1, 300)
>>> sampleset = sampler.sample(bqm)
>>> sampleset.info
{'qpu_access_time': 41990, 'charge_time': 2991424, 'run_time': 2991424}
```

Table 3 Timing Information from Leap's Hybrid Solvers

Field	Meaning
run_time	Time, in microseconds, the hybrid solver spent working on the problem.
charge_time	Time, in microseconds, charged to the account. [2]
qpu_access_time	QPU time, in microseconds, used by the hybrid solver. [3]

[2] *charge_time* and *run_time* may differ due to the time granularity of the solver. For example, if a hybrid solver has a time granularity of 0.5 sec and your submission specifies 4.7 sec, *run_time* might be 5 sec and *charge_time* 4.5 sec.

[3] *qpu_access_time* might be zero for submissions with short *run_time*. Because the QPU is a shared, remote resource, the first set of samples from the QPU might not be received before a short explicitly-specified or default *time_limit* is reached. In such cases, the hybrid solver respects the time limitation and returns without the QPU having a chance to contribute to the solution. On the large, complex problems for which hybrid solvers are intended, this is unlikely to occur.

Solver Usage Charges

D-Wave charges you for time that solvers run your problems, with rates depending on QPU usage. You can see the rate at which your account's quota is consumed for a particular solver in the solver's quota_conversion_rate property.

You can see the time you are charged for in the responses returned for your submitted problems. The relevant field in the response is `'charge_time'`. The example in the Solver Timing section shows `'charge_time': 2991424` in the returned response, meaning almost 3 seconds are being charged.

Instantiating the needed compute resources for your problem can introduce a delay before the problem is processed. This delay tends to be small compared to the overall solution time for large problems. The *charge_time* does not include this delay.

Examples

Use a hybrid solver as you would any Ocean sampler. For example, given a 1000-variable problem formulated as a quadratic unconstrained binary optimization (QUBO) model, Q, you can submit it for solution to a hybrid BQM solver as follows:

```
from dwave.system import LeapHybridSampler

# Select a solver
sampler = LeapHybridSampler()

# Submit for solution
answer = sampler.sample_qubo(Q)
```

Further Information

You can find more detailed information and usage examples here:

- The D-Wave Problem-Solving Handbook guide's Using Hybrid and Classical Solvers section.
- Ocean software's Getting Started documentation examples:
  - Bin Packing and Stock-Sales Strategy demonstrate Leap's hybrid CQM solver.
  - Structural Imbalance in a Social Network demonstrates Leap's hybrid BQM solver.
  - Map Coloring: Hybrid DQM Sampler demonstrates Leap's hybrid DQM solver.
- Hybrid Computing Jupyter Notebooks.
- D-Wave's GitHub repository, dwave-examples, coding examples such as Knapsack Problem.
