



D-Wave Hybrid Solver Service: An Overview

WHITEPAPER

2020-02-25

Overview

This document presents an overview and preliminary performance analysis of the D-Wave hybrid solver service (HSS), available from the Leap™ quantum cloud service. We survey the benefits of taking a hybrid quantum-classical approach to computation, and introduce the concept of *quantum acceleration* to describe the type of computational speedups that may be observed. A comparison of one hybrid solver from the HSS portfolio to a collection of state-of-the art classical alternatives show that the solver performs well on a variety of input structures.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com



Notice and Disclaimer

D-Wave Systems Inc. (“D-Wave”) reserves its intellectual property rights in and to this document, any documents referenced herein, and its proprietary technology, including copyright, trademark rights, industrial design rights, and patent rights. D-Wave trademarks used herein include D-WAVE®, Leap™ quantum cloud service, Ocean™, Advantage™ quantum system, D-Wave 2000Q™, D-Wave 2X™, and the D-Wave logo (the “D-Wave Marks”). Other marks used in this document are the property of their respective owners. D-Wave does not grant any license, assignment, or other grant of interest in or to the copyright of this document or any referenced documents, the D-Wave Marks, any other marks used in this document, or any other intellectual property rights used or referred to herein, except as D-Wave may expressly provide in a written agreement.



Contents

1	Introduction	1
1.1	Operational Overview	2
1.2	Quantum Acceleration of Classical Heuristics	3
2	Performance Overview	4
3	Summary	6
	References	8
A	Details of the Experiments	9
A.1	Time Measurement and Performance Metrics	9
A.2	MQLib	9

1 Introduction

Recent papers and presentations at D-Wave user group meetings have demonstrated hundreds of applications that can run successfully on D-Wave quantum computers. However, in most cases the inputs of interest to practice are too large to fit onto current-model quantum processing units (QPUs) and be solved directly by quantum annealing.

Many ideas have been proposed for overcoming this size limitation by developing hybrid solvers that combine classical and quantum approaches to problem-solving. For developers interested in exploring these ideas, D-Wave has created `dwave-hybrid`, a general Python framework with support for implementing and testing hybrid workflows. Visit [1, 2] to learn more. This framework is part of the Ocean open-source tool suite, which may be found at [3].

For those who prefer to skip the code-development step, D-Wave has launched the **Leap hybrid solver service** (HSS). The HSS contains a collection of hybrid portfolio solvers that target different categories of inputs and use cases. At present the HSS contains one portfolio solver called `hybrid.v1`: it reads an input of size up to 10,000 variables and dispatches one or more hybrid solvers (implementations of individual heuristics) to work on finding solutions. The HSS is available through the Leap quantum cloud service; the portfolio and solver codes are proprietary. Visit [4] to learn more about the Leap quantum cloud service and the HSS.

Benefits of adopting this portfolio approach to hybrid quantum-classical computation include:

- Hybrid solvers in the HSS can accept inputs that are much larger than those solved directly by the QPU. They are designed to leverage the unique capability of the QPU to find good solutions fast, thereby extending this property to larger and more varied types of inputs than would otherwise be possible.
- Solvers in the HSS are designed to take care of low-level operational details for the user: solving problems with this service does not require any knowledge whatsoever about how to select parameter settings for D-Wave QPUs.
- Different types of solvers tend to work best on different types of inputs. Portfolio solvers can run multiple solvers in parallel using a cloud-based platform, and return the best solution from the pool of results. This approach relieves the user from having to know beforehand which solver might work best on any given input, and minimizes the computation time needed to obtain best results.

The remainder of this report presents an overview of how the solvers in the HSS are used and how well they perform.

- Section 1.1 gives an operational overview of the hybrid solvers in the HSS, describing their input/output interface and how the quantum and classical components are organized to work together.
- Section 1.2 presents an illustration of how one hybrid solver in the HSS, called DW, can leverage queries to a D-Wave 2000Q QPU, allowing it to find better solutions

faster than a version without quantum queries in its workflow. We describe this type of performance boost as *quantum acceleration* of the classical workflow.

- Section 2 compares DW to a published report [5] about 37 classical solvers from the MQLib repository. We tested DW on 45 inputs from that repository, and we found that DW compares well to these publicly available classical alternatives:
 - On more than 50 percent of inputs tested, DW found solutions of equal or better quality than all 37 repository solvers.
 - On over 70 percent of the *largest* inputs tested, DW found strictly better solutions.
 - Relative performance of DW generally increased with problem size.

These results should be considered preliminary because the HSS will see continued development and new solvers added in the coming months, which are expected to improve overall performance and widen the scope of inputs that can be solved efficiently.

An upgraded service that reads much larger inputs and incorporates the next-generation Advantage QPU will be announced later in 2020.

1.1 Operational Overview

The current (February 2020) version of the HSS, which includes the hybrid portfolio solver named `hybrid.v1`, provides the following user interface. See [6, 7] for details.

- **Inputs:** To solve a problem using `hybrid.v1`, the user provides two pieces of information:
 - An input for quadratic unconstrained binary optimization (QUBO) or for Ising Model, formulated in D-Wave’s standard binary quadratic model (BQM) format. In the current version, the maximum number of input variables corresponds to a complete graph containing $n = 10,000$ nodes.
 - (Optionally) a time limit T for all solvers to run, in units of seconds. The system calculates a minimum time limit that scales with input size, which may be used by default. The minimum time limit ensures that each hybrid solver has enough time to both perform a first step and to query and receive at least one response from the QPU. In the current version of `hybrid.v1` the minimum time limit for any input size is three seconds and the maximum time limit is 24 hours.
- **Outputs:** The output of `hybrid.v1` consists of the following:
 - A lowest-cost solution from among those found by all solvers in the portfolio, running within the specified time limit.
 - Information about the time the portfolio solver spent working on the problem: `run_time` is the time spent running the problem, including system overhead; `charge_time` is a subset of `run_time` (omitting overhead) that is charged to the user’s account; and `qpu_access_time` is the time spent accessing QPU. Note that

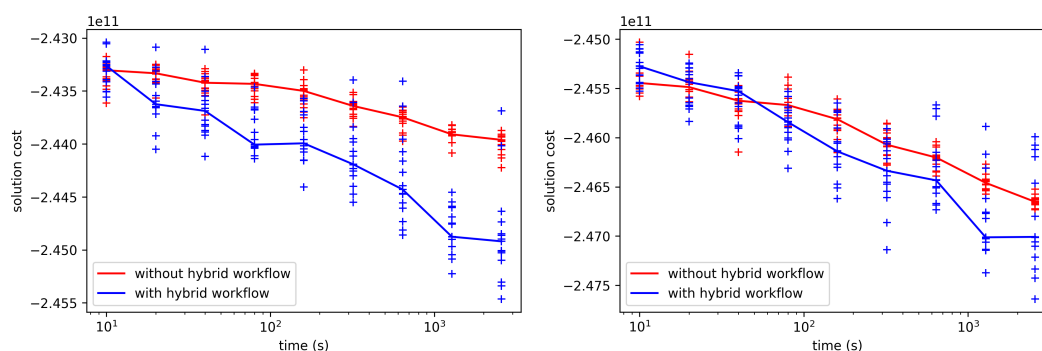


Figure 1: The left and right panels correspond to two different inputs — one dense and one sparse, both containing 4096 variables. Each panel plots solution cost (y-axis, scaled by 10^{11}) versus computation time (x-axis, note logarithmic scale). A column of points represents a sample solution costs obtained from 15 independent random trials at each time limit; the lines connect the median points in each sample. Red points show performance of the DW heuristic without quantum queries; blue points show DW when quantum queries are incorporated into the workflow.

classical and quantum components operate asynchronously in parallel, so total elapsed time does not necessarily equal the sum of component times.

Each hybrid solver contains both classical and quantum components. Upon receiving an input Q , the portfolio front end chooses one or more solvers to work on Q , and starts them running in parallel on a collection of Amazon Web Services (AWS) CPUs and/or GPUs.¹

During its operation each classical component formulates some number of *quantum queries*, which are partial representations of Q that are small enough to be solved directly on a D-Wave 2000Q QPU. At the end of time interval T , all solvers stop and return their results to the front end, which forwards the best solution found to the user.


A D-Wave 2000Q system acts as a quantum query server, receiving queries from active solvers and generating replies. The classical and quantum components in each solver communicate asynchronously so that contention or latency issues in one part of the system do not block progress in the other.

1.2 Quantum Acceleration of Classical Heuristics

The DW solver can be operated in two modes: the *heuristic workflow* implements a fairly standard classical optimization heuristic; and the *hybrid workflow* incorporates a module that formulates a series of quantum queries to be sent to a D-Wave 2000Q QPU; the module also interprets quantum responses as new information to be incorporated into the heuristic workflow.

Figure 1 compares performance of the heuristic and hybrid workflows used in DW, for sparse (left panel) and dense (right panel) inputs that were specifically designed to illustrate a capability that we refer to as *quantum acceleration*. Both panels show how *solution*

¹AWS and AWS Cloud Service are trademarks of Amazon Technologies, Inc.



cost (y-axis), which measures the quality of a given solution, generally tends to improve (decrease) as more and more computation time (x-axis) is spent working on the problem. The columns of points represent costs obtained from 15 independent random trials from each workflow; the lines connect median points in each cost sample.

The heuristic workflow (red) proceeds incrementally, searching an enormous space of possible solutions by making small modifications to a current working solution. Incremental changes can improve the cost function by a only small amount at each step, which limits the rate of progress.

With the hybrid workflow (blue), the information obtained from quantum queries leads to more rapid gains in solution quality, indicated by the fact that the distribution of blue points is generally below the distribution of red points. Note that the red and blue points show considerable overlap, which means that the heuristic workflow could outperform the hybrid workflow in any individual test. However, over most of this time range, the blue line is strictly below the red line, indicating that probabilities tend to favor the hybrid workflow.

Within this hybrid framework, the D-Wave 2000Q QPU is able to exploit limited information about the full-sized problem, and generate useful suggestions about promising regions of the search space to explore. We refer to this difference as an *acceleration gap*: a range of computation times for which the blue solver tends to find better solutions faster than the red solver. We expect that as computation time increases, both solvers will be able to routinely find optimal solutions and the two lines will merge at some point. That is, no acceleration gap can be observed when computation time is very large compared to input complexity.

Note also that the convergence patterns shown here are artifacts of both input and solver structures: different patterns arise for different types of hybrid solvers, and sometimes no acceleration gap will appear. The performance studies in the next section do not attempt to identify or characterize these patterns (an interesting subject for future research); rather we simply measure solution quality in random trials over a small set of time limits.

2 Performance Overview

We present a small performance study comparing DW to a large collection of CPU-based classical solvers.

Dunning et al. [5] evaluate performance of 37 Max Cut and QUBO solvers using a testbed of 3296 inputs that were gathered from problem repositories around the world. The solvers, inputs, and data analysis tools from this extensive project are available online at the MQLib repository.

As mentioned in Section 1.2, solver-to-solver differences in solution quality tend to disappear as solvers are allowed longer runtimes. Therefore, for each input \mathcal{I} , the authors provide a recommended time limit $T_{\mathcal{I}}$, short enough that only a small number of solvers were able to match the best cost $C_{\mathcal{I}}$ found. In their tests, all solvers ran for the recommended time limit for five independent trials each: thus we may consider $C_{\mathcal{I}}$ to be the best result that would be observed by a hypothetical parallel portfolio running five independent copies of each solver, totaling 185 solvers. Here we refer to $C_{\mathcal{I}}$ as the *reference cost*.

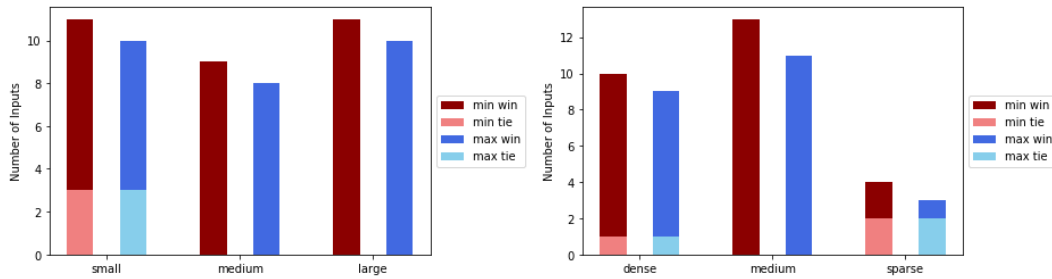


Figure 2: DW performance on 45 MQLib inputs. The left and right panels show results categorized by input size and by input density. Five independent trials of DW were run on each input. Red and pink bars correspond to the minimum-cost solution (best-of-5) found by DW on each input: this result matches the MQLib test design. Blue and light blue bars correspond to the maximum-cost solution (worst-of-5) found by DW: this result shows what would have happened if DW were only allowed one trial per input. The height of a bar indicates the number of inputs (out of 15 per category) for which DW solution cost was strictly better than (dark bars) or tied with (pale bars) the MQLib reference cost.

To evaluate DW performance, we selected 45 inputs from among the “hardest” in MQLib: all have a maximum recommended time limit of $T_I = 20$ minutes, and in each case the reference cost was found by only one MQLib solver (that is, all other solvers found strictly worse solutions). The selected inputs are organized into nine groups of five inputs each, according to graph size (small, medium, large) and density (dense, medium, sparse). See Appendix A.2 for details.


To match the MQLib test design, for each input we ran five independent trials of DW, for 20 minutes per trial. In terms of computational resources, the 37x5 MQLib tests required 62 CPU-hours of total work, or 20 minutes using 185-fold parallelization. Five trials of DW required 1.7 (CPU/GPU/QPU) hours of total work, or 20 minutes using 5-fold parallelization.

For each input, the *minimum* cost is the best result obtained from five independent trials of DW: this number gives an apples-to-apples comparison of DW versus the MQLib solvers. The *maximum* cost is the worst result obtained in five independent trials: this number gives a conservative estimate of how DW would have performed if limited to just one trial as in typical use.

For each trial and each input there are three possible outcomes: the DW solution cost is lower than (win), equal to (tie), or higher than (lose) the reference cost. Figure 2 presents two views of the results. The left panel shows results for input size categories (small, medium, large), and the right panel shows results for input density categories (dense, medium, sparse).

In each panel, the height of each bar shows the number of inputs in each category (out of 15) for which a DW solution either won or tied the MQLib reference cost. The red bars show DW results for the minimum cost, and the blue bars show DW results for the maximum cost observed for each input. The dark red and blue bars show wins and the light bars show ties.

For example, the leftmost bars show that in a best-of-5 comparison, on the 15 smallest inputs, DW tied the best of 185 MQLib solvers on 3 inputs and won on 7 inputs, totaling




11 wins-or-ties; on the remaining 4 inputs, the DW solution was worse than the best of 185 MQLib solvers. The adjacent blue bars show that DW performance in a worst-of-5 comparison on small inputs is not much different: the number of ties is the same and the total number of wins-or-ties is reduced from 11 to 10. Overall, the similarities between red and blue bars indicate that DW solution quality is fairly consistent from trial to trial.

- The left panel shows that DW solutions won or tied on more than half of all inputs tested, in all three size categories. This is true for both the minimum-of-five and maximum-of-five comparisons.
- Ignoring ties, the proportion of wins increases with input size, under both the minimum-of-five and the maximum-of-five metrics.
- On the largest inputs, the maximum-of-5 solution wins on 10 inputs out of 15. Thus, on 2/3 of the largest inputs, just one trial of DW would have sufficed to find strictly better solutions than the hypothetical portfolio of 185 MQLib solvers.
- The right panel indicates that DW performs relatively better on medium and dense inputs, and relatively worse on sparse inputs. This points out the benefit of the portfolio-based approach, since no solver (classical or hybrid) is expected to be always best on all input types: new solvers tuned for efficiency on sparse inputs can be developed and added to the `solver_v1` portfolio, yielding robust performance over a wider variety of input categories.
- A few ties appear on smallest inputs, suggesting that 20 minutes might be enough for both groups of solvers to find optimal solutions.

3 Summary

This report describes general features of the D-Wave hybrid solver service and presents a preliminary overview of performance. To summarize our results:

- The D-Wave hybrid solver service (HSS) represents a significant step forward in lowering barriers to usability of D-Wave quantum computers, by providing an interface that hides low-level details of their operation.
- The HSS portfolio approach relieves the user from having to decide which solution approach is best for any given input.
- The hybrid quantum-classical approach to computation creates new potential for leveraging quantum acceleration to solve much larger and more varied inputs than is possible for a standalone QPU.
- A small performance test of one of the hybrid solvers from the HSS indicates that it compares well to a collection of 37 publicly-available solvers on a variety of inputs that are relevant to practice. The relative performance of the hybrid solver increases with problem size.



An obvious question to ask is how much of the performance advantage we observe may be due to design choices made in different components of the hybrid solver service, e.g., quantum acceleration, fast computing platforms, high-quality code, or superior algorithmic ideas. Characterizing the relative contribution of these factors would require extensive systematic tests that are well outside the scope of this brief report. This is an interesting subject for further study.

References

- ¹ *D-Wave Hybrid*, <http://github.com/dwavesystems/dwave-hybrid>, accessed February 2020.
- ² *D-Wave Ocean Documentation*, http://docs.ocean.dwavesys.com/en/latest/docs_hybrid/sdk_index.html, accessed February 2020.
- ³ *D-Wave Ocean*, <http://ocean.dwavesys.com>, accessed February 2020.
- ⁴ *D-Wave Leap*, <http://cloud.dwavesys.com/leap>, accessed February 2020.
- ⁵ Dunning et al., “What works best when? A systematic evaluation of heuristics for Max-Cut and QUBO,” *INFORMS Journal on Computing* **30**, <http://github.com/MQLib> (2018).
- ⁶ *D-Wave Solver Properties and Parameters Reference: User Manual*, D-Wave Technical Report 09-1169A-1, D-Wave Systems, Inc. available from <http://dwavesys.com/resources/publications>, 11 February 2020.
- ⁷ *Solver Computation Time*, D-Wave Technical Report 09-1107A-M, D-Wave Systems, available from <http://dwavesys.com/resources/publications>, 11 February 2020.

A Details of the Experiments

This appendix presents details of our test protocols as well as the inputs and solvers selected for study.

A.1 Time Measurement and Performance Metrics

The hybrid DW solver evaluated here has code components that run on three types of platforms: CPU, GPU, and QPU. Based on an initial scan of the input, the `hybrid.v1` portfolio solver selects some number of CPU and GPU platforms to be used in the computation; these components run independently and asynchronously, occasionally sending quantum queries to a D-Wave QPU, and incorporating its responses when they arrive.

Naturally this complex arrangement creates considerable challenges in obtaining accurate and repeatable runtime measurements. Our policies for measuring and reporting runtimes were as follows:

- The CPU and GPU code ran in a `p3.2xlarge` AWS instance containing an NVIDIA Tesla V100 platform with 16GB memory, 60Gb of RAM and eight virtual cores; and an Intel Xeon(R) E5-2686 v4 (Broadwell) platform. CPU time measurements were taken with hyperthreading turned on.
- Programming languages used in this system include python, cython, C++, and C++ with the CUDA toolkit. C++ code was always compiled with the `-Ofast` or `-O3` optimization flags; the CUDA compiler has optimization turned on by default.
- Quantum queries were sent to a D-Wave 2000Q low-noise system located at D-Wave headquarters in Burnaby, BC. Quantum algorithm parameters were set to default values throughout: $20\mu s$ anneal time and 100 reads per input.
- Elapsed computation time for the portfolio solver is specified by the user as a time limit T . The global runtime clock starts and finishes on the same platform, in a single thread that forks the processes needed to invoke solvers on other platforms. The child processes are responsible for monitoring their own progress and sending solutions to the parent process before the time limit T is reached.
- Since individual solvers work in parallel and communicate asynchronously with the QPU, total computation time does not equal the sum of component times. Note that this design ensures that computational progress is not stalled by issues such as network latency or contention for the QPU, which are impossible to control or predict.

A.2 MQLib

The MQLib repository [5] contains 3296 inputs for Max Cut or QUBO problems and a collection of 10 Max Cut and 27 QUBO heuristic solvers. The solvers are implemented in C++ and run on standard CPU cores. It also contains an extensive collection of support

tools for, e.g., translating inputs between problem formulations, running tests, analyzing results, and selecting the best solvers for any given input.

We did not carry out independent tests using these solvers, but instead refer to performance data available in the MQLib repository. In particular, the `/data/` directory contains a recommended run time limit T for each input, and a *reference* cost obtained by running all 37 solvers for time limit T , over five independent trials each. Our tests use the same time limit and record solution costs obtained in five independent trials of the DW hybrid solver.

Solvers MQLib runtime measurements on 37 solvers were performed in 2018 using AWS cores, as follows [5]:

We performed heuristic evaluation using 60 `m3.medium` Ubuntu machines from the Amazon Elastic Compute Cloud (EC3), which have Intel Xeon E5-2670 processors and 4 GiB of memory; code was compiled with the `g++` compiler using the `-O3` optimization flag.

The authors report that their full evaluation (37 solvers and 3296 inputs) required 2.0 CPU-years of processing power, 12.4 days of computation (over 60 machines), and cost \$1196.00 in AWS compute time.

Inputs We have selected 45 instances from the MQLib repository for testing purposes, using the following procedure.

- The MQLib designers noted that results tend to be identical when all solvers are given too much time to work on a given instance. Thus, to each instance they assign a *suggested runtime* T that can distinguish performance among solvers. Our selection protocol considers only “hard” instances with a maximum recommended runtime of $T = 20$ minutes, and instances for which just one of the 37 solvers found the best reported solution (no ties).
- Inputs that are unconnected, contain fewer than $n = 1000$ variables, or more than $n = 10,000$ variables, were not selected.
- The remaining input pool was partitioned into groups according to size [small ($1000 \leq n \leq 2500$), medium ($2500 < n \leq 5000$) and large ($5000 < n \leq 10000$)] and edge density [sparse ($d \leq 0.1M$), medium ($0.1M < d \leq 0.5M$), and dense ($d > 0.5M$)]. Here d is the mean edge density of the instance and M is the number of edges in a complete graph with n nodes. This yields nine input groups: for each group we select 5 inputs uniformly at random after filtering as described above.
- Inputs stored in Max Cut form were translated to an equivalent QUBO form for testing on our system.