D-Wave
The Quantum Computing Company™

D-Wave Hybrid Solver Service + Advantage: Technology Update

TECHNICAL REPORT

Catherine McGeoch, Pau Farré and William Bernoudy

2020-09-25

Overview

The D-Wave hybrid solver service (HSS), available from the Leap™ quantum cloud service, was launched in February 2020. This document presents an overview and performance evaluation of an upgraded version of HSS launched in September 2020. The new version incorporates the Advantage™ quantum computer, and contains software enhancements that expand the size and scope of problems that can be solved efficiently. This report shows how a hybrid approach incorporating both quantum and classical components can outperform either method used alone. A small performance study demonstrates that the upgraded version outperforms its predecessor as well as state-of-the art classical alternatives.

14-1048A-A
D-Wave Technical Report Series

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

Summary

The D-Wave Hybrid Solver Service (HSS) was launched in February 2020. This report describes an upgraded version of the HSS made available in September 2020, with a comparison to its predecessor. Key points are summarized below.

- The HSS contains a portfolio of hybrid solvers that exploit both classical and quantum computation methods to find solutions to inputs much larger than the quantum chip alone can read. With its easy-to-use interface, HSS represents a significant step forward in lowering barriers to usability of D-Wave quantum computers.
- We demonstrate a phenomenon known as *quantum acceleration*, whereby queries to the quantum processing unit (QPU) are used to guide the classical solver to find better-quality solutions faster than would otherwise be possible. This illustrates the unique power and potential of the hybrid approach to problem-solving.
- The new version of HSS improves over the previous in two major ways: it incorporates the 5000-qubit Advantage (QPU) rather than its predecessor, the 2000-qubit 2000Q QPU; and it can read much larger inputs, with up to one million variables (if not fully connected), or to 20,000 variables (if fully connected).
- A small performance study shows that both versions of HSS solvers can outperform a collection of 37 publically-available solvers, on a variety of inputs that are relevant to practice. The upgraded version also outperforms its predecessor in these tests: when given the same amount of computation time, version 1 finds solutions of better or equal quality on 67 percent of inputs, while version 2 finds solutions of better or equal quality on 84 percent of inputs. Version 2 also compares well to the public solvers on inputs that are too large to fit on version 1.

Contents

1 Introduction

Numerous research papers and presentations at D-Wave user-group meetings have demonstrated over 200 types of problems that can be formulated and run on D-Wave's annealing-based quantum computers [1]. A review of this body of work shows that application inputs, i.e. those of interest in real-world practice, are typically too large to fit onto current-model quantum processing units (QPUs), which means that they cannot be solved directly by the quantum system.

Many ideas have been proposed for overcoming this size limitation by developing *hybrid solvers* that combine classical and quantum approaches to problem-solving, exploiting the best features of each computing paradigm. For developers interested in exploring these ideas, D-Wave has created **dwave-hybrid**, a Python framework with support for implementing and testing hybrid workflows. This framework is part of the open source Ocean developer's tool suite: visit [2, 3] to learn more.

For those who prefer to skip the code-development step, D-Wave launched the **Leap hybrid solver service (HSS)** in February 2020, containing a collection of hybrid portfolio solvers that target different categories of inputs and use cases. 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 read and solve much larger inputs than current-model QPUs. The solvers work by sending *quantum queries* to a D-Wave quantum processor, using the replies to guide their search of the larger solution space. This approach leverages the unique problem-solving capabilities of the QPU, and extends those capabilities 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 choose parameter settings for D-Wave QPUs.
- Different types of classical solvers tend to work best on different types of inputs. A portfolio solver can select and 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 guess beforehand which solver might work best on any given input, and minimizes the computation time needed to obtain best-quality results.

As of September 2020, the HSS contains two portfolio solvers here referred to as `version1` and `version2`.[1] The former is essentially the original solver launched in February 2020 (with small upgrades), and the latter supports several new features:

- Solvers in `version1` send quantum queries to a 2000-qubit 2000Q QPU, while solvers in `version2` employ a 5000-qubit Advantage QPU. See [5] to learn more about the Advantage quantum system.

[1] For search engine purposes, their official names are `hybrid_binary_quadratic_model_version1` and `hybrid_binary_quadratic_model_version2`.

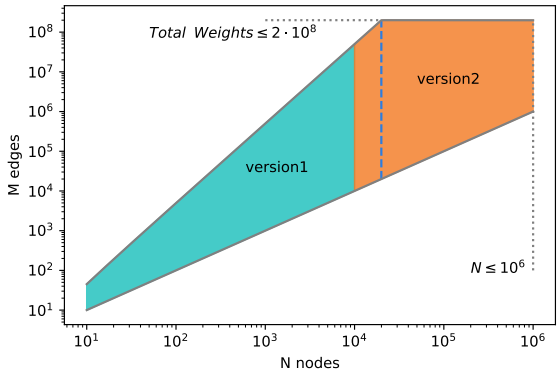


Figure 1: Input graph sizes for HSS solvers, plotted as number of edges M versus number of nodes N (note the double logarithmic scale). The teal region shows size limits for `version1` and the orange region shows the extended size limits supported by `version2`. The blue dashed line marks the size limit for fully-connected graphs read by `version2`.

- Solvers in `version1` can read fully-connected input graphs with up to 10 thousand nodes. Solvers in `version2` can read fully-connected input graphs with up to 20 thousand nodes; for graphs that are not fully-connected, `version2` can read inputs with up to 1 million nodes and 2 million total weights (weights may be assigned to both nodes and edges). This represents an increase of between 2-fold and 100-fold in the number of problem variables (i.e. nodes) that these solvers can read. Figure 1 illustrates the difference in input sizes for the two portfolio versions, in terms of number of nodes, edges, and total number of input weights.
- Some hybrid solvers in `version2` have been upgraded for improved performance on certain categories of inputs, as described in Section 2.

Starting October 2020, HSS will contain a third portfolio solver that can read *discrete quadratic models* (DQMs) as inputs. That is, instead of binary solvers that read problems defined on binary values like $[0, 1]$ or the DQM solver reads problems with variables that are defined on finite discrete sets of values. For example, the user can specify that variables are to be assigned values corresponding to four DNA bases [A,C,G,T], or to 24 starting times [00:00, 00:30, . . ., 24:00, 24:30] See the D-Wave white paper and documentaion [6] for more about this new category of hybrid solvers in the HSS.

This report presents an overview of the two binary portfolio solvers currently in the HSS, together with a small performance comparison, as follows.

- Section 1.1 gives an operational overview of HSS solvers, describing their input/output interface and explaining how the quantum and classical components are organized to work together.
- Section 1.2 demonstrates how a `version2` solver can leverage queries to an Advantage QPU, to find better solutions faster than an implementation without quantum queries in its workflow. We describe this type of performance boost as *quantum acceleration* of the classical heuristic workflow.

- Section 2 compares a `version1` and a `version2` solver to a published performance report [7] about 37 classical solvers from the MQLib repository [7]. We find that both HSS solvers outperform the best of the MQLib solvers, and that `version2` outperforms `version1`.
  - The first test uses 45 MQLib inputs with up to $N = 10,000$ variables (which we call *standard inputs*), which can fit on both hybrid solvers. On the set of standard inputs, the `version1` solver found solutions of better or equal quality than all 37 MQLib solvers, on 30 of 45 inputs (67 percent), while the `version2` solver found better or equal solutions on 38 of 45 inputs (84 percent).
  - The standard inputs fall in three categories: dense, medium, and sparse. The `version1` solver performs well against MQLib solvers on dense and medium inputs, but can only beat or match MQLib on 4 of 15 sparse inputs (33 percent). The `version2` portfolio has been upgraded for better performance on sparse inputs, and finds better or equal solutions in 13 of 15 cases (87 percent).
  - Our second test (*extra-large inputs*) looks at performance of `version2` on an additional set of 20 MQLib inputs with between $N > 10,000$ and $N = 53,130$ variables (the largest available in MQLib). On these inputs the `version2` solver finds better solutions than the best of 37 MQLib solvers in half the cases.

These performance results should be considered preliminary because the HSS will see continued improvement and expansion of problem scope, with new solvers to be added in the coming months and years.

1.1 Operational overview

The hybrid portfolio solvers currently in the HSS provide the following user interface; see [8] for details.

- **Inputs:** The user provides two pieces of information:
  - **Input** Q. The solver reads an input for the quadratic unconstrained binary optimization (QUBO) problem (defined on variables (0,1)), or for the Ising Model optimization problem (defined on variables (-1, +1)). The input Q is formulated in D-Wave's standard binary quadratic model (BQM) format.

    For `version1` the maximum input size corresponds to a complete graph containing $N = 10,000$ nodes and $M = 49,995,000$ edges. For `version2` the graph may contain up to one million nodes, assuming a maximum total of two hundred million weights on nodes and edges. The largest complete graph that fits these constraints has $N = 20,000$ nodes. Figure 1 illustrates these size boundaries.
  - **Time limit** T: The user can (optionally) provide a maximum time limit for all solvers to run, in units of seconds. The portfolio solver calculates a minimum time limit based on input size, which may be used by default, if desired. The minimum time limit ensures that each hybrid solver has enough time to perform initialization and to query and receive at least one response from the QPU. The minimum and maximum time limits for any input size are 3 seconds and 24 hours, respectively.

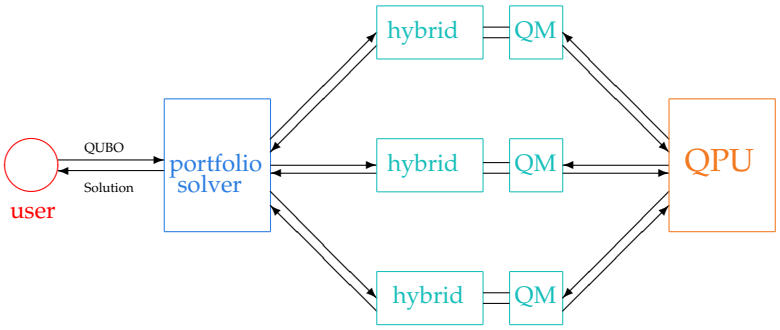


Figure 2: Structure of a portfolio solver. The portfolio front end (blue) reads an input Q and optionally at time limit T. It it invokes some number of hybrid solvers running on classical CPUs and GPUs (teal), to find solutions to Q. The hybrid solvers contain a quantum module (QM) that formulates and sends quantum queries to a D-Wave quantum processor (orange), which supplies answers to the queries. At time limit T, the hybrid solvers send their results to the portfolio front end, which forwards the best solution found to the user.

- **Outputs:** The solver output consists of the following:
  - A lowest-cost solution from among those found by all solvers in the portfolio, while running within the specified time limit.
  - Information about the time the portfolio solver spent working on the problem: *run_time* is the total elapsed time 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 the classical and quantum solver components operate asynchronously in parallel, so the total elapsed time does not necessarily equal the sum of component times.

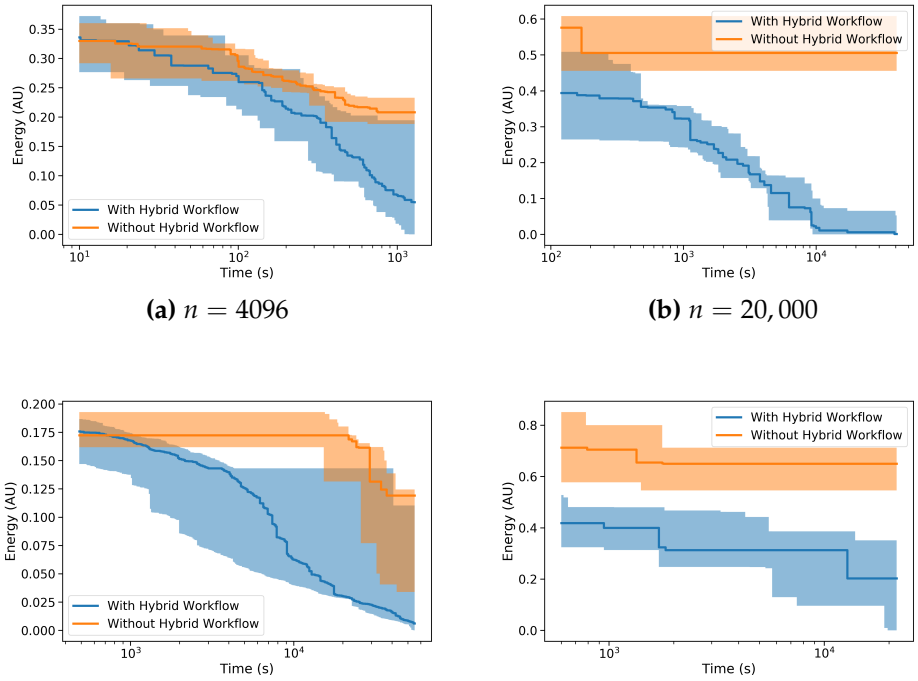
Each hybrid solver within a porfolio solver contains both classical and quantum components. Upon receiving an input Q, the portfolio front end chooses one or more solvers to work on Q, and starts them running in parallel on a collection of CPU and/or GPU platforms provided by Amazon Web Services (AWS).

Figure 2 shows how classical and quantum components are structured to work together. The portfolio front end (blue) reads an input Q and optionally a time limit T, submitted by the user. Depending on the size and structure of Q, it invokes some number of hybrid solvers running on classical CPUs and GPUs (teal). The hybrid solvers contain a query module (QM) that communicates with a specific D-Wave quantum processor (orange). Before time limit T is reached, the hybrid solvers send their results to the portfolio front end, which forwards the best solution found to the user.

During the computation, the QM formulates and sends *quantum queries*, which are partial representations of input Q that are small enough to be solved directly on an Advantage QPU (used by `version2`) or a 2000Q QPU (used by `version1`).[2] The QM receives replies from the QPU, and formulates those replies into suggestions for the hybrid solvers about promising regions of the solution space to be searched.

In this framework the D-Wave quantum processor acts as a quantum query server, receiving queries from active solvers and generating replies in the form of samples of solutions

[2] This assignment holds in normal use. In the extremely rare event that access to the preferred QPU fails during the computation, the hybrid solver may switch to a backup query server of a different type.



(a) $n = 4096$ **(b)** $n = 20,000$

(c) $n = 80,000$ **(d)** $n = 1,000,000$

Figure 3: Performance of a `version2` solver using a hybrid workflow with quantum queries enabled (blue) and a heuristic workflow with quantum queries disabled (orange). The four panels show results for four inputs of sizes between $n = 4096$ and $n = 10^6$, plotting solution quality versus computation time (note the logarithmic x-scale). On each input, the `version2` solver was run for five independent trials, and progress (solution quality versus computation time) was sampled over a range of times. Solution quality is measured as the scaled residual distance from the best solution found in all trials. The shaded regions show the minimum and maximum solution energies found over five trials, and the lines trace median solution energies, at each point in time.

to its inputs. 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

Internal versions of HSS solvers can operate in two modes, called *workflows*. The *hybrid workflow* implements a classical optimization heuristic that incorporates a query module (QM): the QM formulates quantum queries, sends them to the QPU, receives replies, and incorporates them back into the classical workflow. *Heuristic workflow* refers to the same hybrid solver with quantum queries disabled.[3]

Figure 3 compares performance of the heuristic (orange) and hybrid (blue) workflows,

[3]For reasons of solver efficiency and interface simplicity, parameters such as choice of workflow mode are not supported in implementations of HSS solvers available to the public.

using one of the solvers in the `version2` portfolio. The figure shows results from tests on four inputs (one per panel) that were designed to illustrate a phenomenon that we refer to as *quantum acceleration*.

The four inputs contain between $n = 4096$ and $n = 1,000,000$ variables. For each input, the solver was run for five independent trials in each mode, for a range of stopping times. The shaded regions show the minimum and maximum values of solution energies e observed over five trials at each point in time, and the lines show median energies.

The y-axis corresponds to scaled energy distance $\delta = (e - e_{min})/e_{min}$, where e_{min} is the minimum energy discovered for this input, over all trials. The x-axis corresponds to times at which improved energies were reported (note logarithmic x-scale). The lower limit of the x-axis corresponds to the minimum time limit set by the portfolio solver for each input size. The upper time bounds ranged from 20 minutes in panel (a) to just under 14 hours in panel (c).

As is typical for opimization heuristics, we see that both workflows show progress, finding better solutions with longer computation times. As a general rule, we expect that both solvers will eventually converge to the same (optimal) solution energy, if given enough time, as suggested by the overlap of shaded regions in panel (c); however it is always possible for a solver to get "stuck" in a local minimum and fail to make progress (which may have happened to the heuristic workflow in panels (b) and (d).

The four panels illustrate a variety of convergence paths that may be observed, from input to input and trial to trial, but in every case the the hybrid workflow converges *faster* toward better-quality solutions. We refer to this phenomenon as *quantum acceleration*: the Advantage QPU is able to exploit limited information about the full problem, and to generate useful suggestions about new promising regions of the search space to explore. Quantum acceleration guides the classical heuristic to find better solutions faster than would otherwise be possible, over the range of computation times.

Note that these inputs were constructed specifically to demonstrate this property, and that quantum acceleration does not necessarily take place on every input submitted to HSS. For example, some inputs may have a simple combinatorial structure, which means that the classical solver does not require a quantum boost to be able to converge quickly to near-optimal solutions. Other inputs may contain structures that are too complex to be captured by the restricted information in a quantum query.

Even when quantum acceleration does occur, it may not necessarily be observed if the time limit T is too small or too large. For example, in panel (a) comparison of solution energies at times below 30 seconds would reveal no real difference between the two workflows. Similarly, as mentioned earlier, we expect both workflows to eventually converge to the same solution, which would again close the performance gap between the two.

2 Performance overview

This section presents a small performance study comparing `version1` and `version2` to a large collection of CPU-based classical solvers available in an online repository.

Dunning et al. [7] evaluate performance of 37 Max Cut and QUBO solvers using a testbed of 3296 inputs, which were culled from online optimization repositories worldwide, and

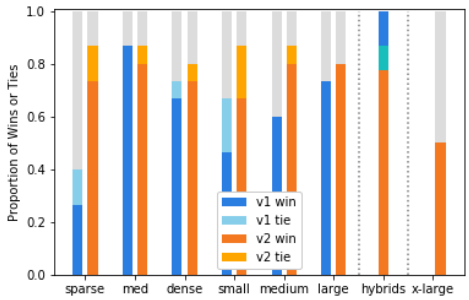


Figure 4: Performance of HSS `version1` (blue) and `version2` (orange) solvers on MQLib inputs. In each pair, the blue bars show the proportion of wins (darker) and ties (lighter) achieved by `version1` in comparison to MQLib. The orange bars show proportion of wins (darker) and (lighter) achieved by `version2` in comparison to MQLib. The first six pairs of bars describe performance on 45 original MQLib inputs, divided into groups according to graph density (**sparse, medium, dense**) and size (**small, medium large**). The next bar marked **hybrids** shows a head-to-head comparison of `version1` (blue) and `version 2` (orange), with ties marked in teal. The rightmost bar **x-large** compares performance of `version2` against MQLib solvers on 20 very large inputs that are too big for `version1`.

translated to Max Cut and QUBO form for this repository. The solvers, inputs, and data analysis tools from this extensive project are available from the MQLib repository (see [7]).

As mentioned in Section 1.2, solver-to-solver differences in solution quality tend to disappear when heuristic solvers are allowed long runtimes, because many will eventually converge to the same (optimal) solution energy. Similarly, performance differences are hard to detect when solvers are allowed very short runtimes, because initial solutions tend to be more-or-less random. Thus distinctions in solver performance with respect to solution quality can only be observed when time limits used in the experiment are somewere between these two extremes.

For this reason, for each input Q the authors provide a recommended time limit T_Q, short enough that only a small number of solvers were able to match the best solution cost C_Q found. Running multiple tests for the recommended time limit makes it possible to observe systematic differences in solver performance.

In their tests, all solvers ran for five independent trials using recommended time limits. Thus we may consider C_Q 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_Q as the *reference cost*, or *reference energy*.

Our tests use two subsets of MQLib inputs that are selected from among the "hardest" in MQLib. That is, all have a maximum recommended time limit of $T_Q = 20$ minutes, and in each case the reference cost was found by only one or two MQLIb solvers: that is, the remaining solvers found strictly worse solutions. The inputs were selected at random in batches to cover a range of sizes and densities: see Appendix A.2 for details.

To match the MQLib test design, for each input we ran five independent trials of a single solver from `version1` and a single solver from `version2`, for 20 minutes per trial. In terms of computational resource usage, the 37×5 MQLib tests required 62 CPU-hours of total work per input, or 20 minutes using 185-fold parallelization. Five trials of a single HSS

solver required 1.7 (CPU/GPU) hours of classical work per input (running in tandem with the QPU), or 20 minutes using 5-fold parallelization.

Our first experiment compares performance of the `version1` and `version2` solvers against the best of 37 MQLib solvers, using a set of 45 inputs from the MQLib repository (the "standard set") that are small enough to fit on both solvers. The second experiment looks at performance of the `version2` solver on an additional 20 inputs (the "x-large set") that are too large for `version1` and among the largest available in MQLib. Details about input properties and our selection procedure may be found in Appendix A.2.

For each input Q we record the best solution energy obtained from five independent trials of each hybrid solver, and compare it to the MQLib reference cost C_Q (the best found in all trials of all 37 solvers). For each input there are three possible outcomes: the best solution energy found by the hybrid solver in five trials is lower than (win), equal to (tie), or higher than (lose) the reference cost.[4]

Figure 4 presents the tallys for the two HSS solvers.

The leftmost group of bars shows performance of `version1` (blue) and `version2` (orange) solvers versus MQLib solvers, using our standard set of 45 inputs. These inputs fall into three groups of 15 according to graph density (dense, medium, sparse); the same inputs fall into three groups of 15 according to graph size (small, medium, large), totaling 9 groups of 5 inputs each.

The next bar, labeled "hybrids," shows results of a head-to-head comparison of `version1` and `version2` on the 45 standard inputs, with ties marked in teal. The rightmost bar labelled "x-large" shows a comparison of `version2` to the MQLib solvers on the 20 extra-large inputs (no ties).

Hybrid solvers versus MQLib on standard inputs. The first group of six pairs shows test results for "standard" inputs organized by density (sparse, medium, dense) and by size (small, medium, large). The blue bars show performance of the `version1` solver using the 2000Q quantum processor, versus 37 MQLib solvers, and orange bars show performance of the `version2` solver using Advantage system (orange), again versus the MQLib 37. The length of each bar shows the proportion of tests that resulted in wins (darker blue or orange) or ties (lighter blue or orange) for the HSS solvers. The grey sections of each bar show the proportion of wins for MQLib solvers.

In the first three cases (sparse, medium, dense) we see that the `version2` solver significantly outperforms the `version1` solver on sparse inputs, improving from wins-or-ties on 40 percent of tests (6 of 15) up to 87 percent of tests (12 of 15). This improvement reflects a combinaton of software enhancements and the move from the D-Wave 2000Q to Advantage as a back-end query server. On medium and dense inputs, the two hybrid solvers show about the same performance.

Considering the next three cases (small, medium, large) we observe that the `version2` solver shows good improvement over `version1` on all three size categories, with greater performance gap seen on small and medium-sized inputs.

[4] A look at the data suggests that some wins and losses are by very small margins, and may in fact be misrecorded ties. That is, it is possible that the same solution gives rise to slightly different costs when they are calculated in different computing environments, due to numerical precision issues for the enormous input sizes being tested. However, there is no way to verify this hypothesis without access to the original solutions and calculations used in MQLib.

Over all 45 inputs in this test, the `version1` solver has 27 wins and 3 ties versus the best of 37 MQLib solvers, totalling 67 percent, while the `version2` solver has 34 wins and 4 ties versus MQLib solvers, totalling 84 percent.

`version1` versus `version2` on standard inputs. The next column labeled "hybrids" shows results of a head-to-head comparison of the two hybrid solvers on the set of 45 standard inputs. Here, `version2` solver found better solutions in 35 cases (78 percent), and the two solvers tied in 5 cases (11 percent). We attribute the superior performance of `version2` to a combination of coding and algorithmic improvements to the hybrid software — particularly targeting performance on sparse inputs — and to the upgrade from the 2000-qubit 2000Q QPU to the 5000-qubit Advantage QPU as the back-end quantum query server.

`version2` vs MQLib solvers on extra-large inputs. The rightmost bar of Figure 4 shows results for the 20 extra-large MQLib inputs, of sizes between $N > 1000$ and $N = 53,130$ (the largest available in MQLib). On these inputs, in five trials the `version2` solver found better solutions than the best of 37 MQLib solvers (each run for five trials, totaling 185 trials), on exactly half of the inputs.

Because none of the extra-large inputs in MQLib meet our selection criteria for medium and dense inputs, all 20 inputs in the test set are categorized as sparse. Within that context, we observe that `version2` tends to perform relatively better on the less-sparse problems in this test set. See Appendix 2.1 for details about the inputs used in these tests.

More extensive evaluation of our hybrid solvers, on a variety of inputs in the full range of sizes that can be accomodated by `version2`, is of course an interesting question future research. Progress on this front has been somewhat hampered by unavailability of suitable classical competition that can also handle the larger input sizes; however we look forward to removal of this temporary obstacle in the next several months.

3 Summary

This report describes general features and properties of the upgraded D-Wave hybrid solver service avalailable as of September 2020, and presents a preliminary look at hybrid solver performance. A comparison of an earler `version1` solver to an upgraded `version2` solver in the HSS portfolio shows that `version2` significantly outperforms `version1`. Both solvers outperform a collection of 37 classical solvers available in a public repository, when tested on a wide variety of inputs that have structures that are relevent to applications practice.

We look forward to future work to expand this preliminary study with more tests comparing performance of solvers in the HSS portfolio against state-of-the-art classical solvers, on a greater variety of inputs of relevance to applications practice.

References

1 *D-Wave Applications*, (web page and search tool), available at http://dwavesys.com/applications, accessed September 2020.

2 *D-Wave Hybrid*, http://github.com/dwavesystems/dwave-hybrid, 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 C. McGeoch and P. Farré, *The D-Wave Advantage System: An Overview*, https://dwavesys.com/resources/publications (D-Wave Technical Report 14-1049A-A, 2020).

6 C. McGeoch and P. Farré, *Discrete Quadratic Model Solver: An Overview*, https://dwavesys.com/resources/publications (D-Wave Technical Report (to appear), 2020).

7 Dunning et al., "What works best when? A systematic evaluation of heuristics for Max-Cut and QUBO," INFORMS Journal on Computing **30**, also visit http://github.com/MQLib (2018).

8 *D-Wave Documentation: Hybrid Solvers*, https://docs.ocean.dwavesys.com/en/stable/overview/hybrid.html, accessed August 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 Measurement and metrics

The hybrid DW solvers evaluated here have code components that run on three types of platforms: CPU, GPU, and QPU. In everyday use, for some given input, a front-end portfolio solver selects some number of solvers (programs implementing specific heuristics) and some number of CPU and GPU platforms to be used in the computation.

At present, HSS contains two portfolio solvers, one from the original launch and an upgraded version that reads larger inputs and incorporates an Advantage QPU instead of the 2000Q QPU. In our tests, both portfolios are set to choose one hybrid solver for all inputs, which we call `version1` and `version2`. Both versions run in asynchronously in a distributed computing framework, sending quantum queries to their respective quantum processors and incorporating QPU responses when they arrive.

Naturally this 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 (`version1` and to a D-Wave Advantage system `version2` located at D-Wave headquarters in Burnaby, BC. Quantum algorithm parameters were set to default values throughout: 20μs anneal time and 1000 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 [7] 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 refered to published 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 the 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 [7]:

> 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 CPU-years of processing power, 12.4 days of computation (over 60 machines), and cost $1196.00 in AWS compute time.

Inputs For the "standard" test set described in Section 2, we selected 45 instances from the MQLib repository using the following procedure.

- The MQLib designers noted that results tend to be similar or identical when all solvers are given either too little or too much time to work on a given instance. Thus, to each instance they assign a *suggested runtime* T that can be used to distinguish performance among solvers.

  Our selection protocol considered only "hard" instances with a maximum recommended runtime of $T = 20$ minutes, and instances for which at most two of the 37 solvers found the best reported solution. On 42 of the 45 inputs exactly one solver found that solution; on the remaining three inputs, two solvers tied.

- Inputs that are unconnected, contain fewer than $n = 1000$ variables, or more than $n = 10,000$ variables, were not selected for the standard set.

- 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 in MQLib were translated to an equivalent QUBO form for testing on our system.

For the "extra-large" input set we filtered the MQLib input set as described above. We selected 20 inputs uniformly at random from the resulting filtered set, with sizes $n > 10,000$. This selection process included the largest input in MQLib, which contains $n = 53,130$ nodes.

The extra-large MQLib inputs are extremely sparse, and fail to meet our criteria for medium and dense inputs. The selected inputs in this group have sizes in range $24,052 \leq n \leq 53,130$, and densities in range $.000095M \leq d \leq .0031M$

Both sets of inputs, translated from MQLib (Max Cut) format to D-Wave DIMOD (QUBO) format, are available at the D-Wave github site *github.com/dwavesystems/hss-overview-benchmarks*.