The Quantum Computing Company™

Hybrid Solvers for Quadratic Optimization

WHITEPAPER

Summary

We present an overview introduction to D-Wave's hybrid solver service (HSS). HSS contains three solvers — BQM, DQM, and CQM — that can be applied to quadratic optimization problems defined on discrete or continuous variables. The most general CQM solver provides support for problems defined on binaries, integers, and reals, as well as a convenient notation for expressing linear and quadratic constraints. All solvers in the HSS are designed to integrate classical and quantum solution methods, exploiting the power of both computing paradigms.

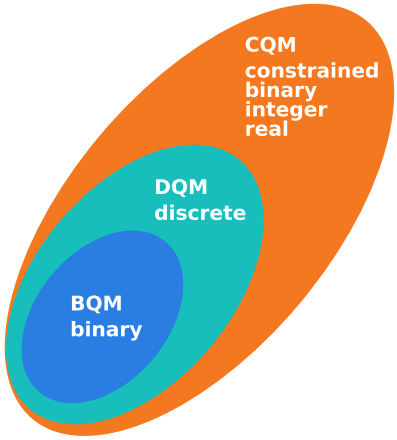


Figure 1: Expanding scope of problems and variable types supported by HSS solvers.

1 Introduction

D-Wave's hybrid solver service (HSS) contains a portfolio of heuristic solvers that leverage both quantum and classical solution approaches to read and solve optimization problems much larger than can fit on Advantage™ quantum processors. Furthermore, HSS solvers provide interface support for applications well outside the native problem formulation, which is quadratic, unconstrained and binary. This interface reduces — and sometimes completely eliminates — the need for users to translate their application problems into a formulation that matches the quantum hardware.

Figure 1 illustrates the result of D-Wave's continuing efforts to expand the variety of problems that fall within scope of the HSS portfolio. The BQM and DQM solvers read unconstrained quadratic problems defined on binary variables (taking two values), and on discrete variables (taking multiple values), respectively. In May 2022, CQM solver support was extended to continuous models: it now can read constrained problems defined on binary, integer, *and* real variables.

To our knowledge, this is the world's first and only hybrid solver capable of leveraging quantum computation to address both discrete and continuous problems.[1]

[1]Some notational conflict is unavoidable in standard usage: `binary`, `discrete`, and `integer` variables in computer science are examples of discrete number domains in mathematics, and `real` variables belongs to the continuous number domain.

This white paper presents an overview of solvers in the HSS portfolio, as follows.

- Section 2 surveys the varieties of problem types that can be addressed by solvers in the HSS.
- Section 3 presents a small performance comparison illustrating how problem features can be matched to solver types.
- All solvers in the HSS implement a classical front end that works in tandem with a quantum back end. Section 4 describes this arrangement and shows how the hybrid workflow can demonstrate *quantum acceleration* of a classical computation.

The hybrid solver service is cloud-based and offered by subscription via the Leap™ web portal; see [1,2] to learn more about Leap and the hybrid solver service.

For developers who prefer to implement their own approaches to combining quantum and classical computation, D-Wave offers **dwave-hybrid**, an open-source Python framework with support for building hybrid workflows that interface with D-Wave's quantum annealing processors; visit [3] to learn more.

2 Quadratic Models for Real-World Problems

This section illustrates the differences in types of problem formulations — called models — that are supported by the three solvers in HSS.

We start with the binary quadratic model (BQM), which matches the native formulation used by Advantage quantum processing units (QPUs). In principle, any NP-hard problem can be efficiently translated to a BQM and solved directly in quantum hardware. However, most BQMs of real-world interest are simply too large to fit on current-sized Advantage quantum processing units (QPUs). The BQM hybrid solver in HSS removes this obstacle by providing an interface that can read general BQMs containing up to one million variables (see Table 1).

Researchers have built up a large "cookbook" of problem translations to DQMs: see [4] to learn about the hundreds of different applications that have been demonstrated to run on D-Wave systems. However, new application problems continually arise, and new and better translation strategies are continually needed.

The DQM and CQM solvers are part of an outreach effort by D-Wave developers to expand the scope of models that can be solved directly in HSS, without needing additional translation to BQMs. These solvers can be more convenient to use and, in some cases, can deliver better hybrid performance than the BQM solver.

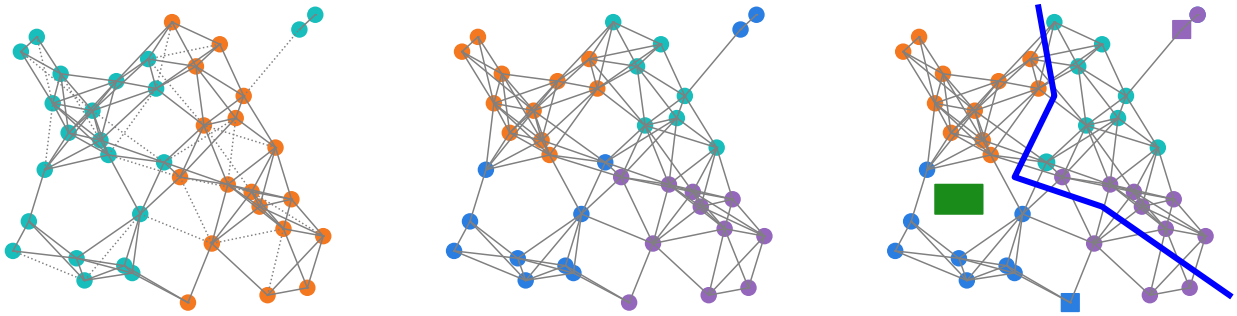
Binary Quadratic Models Graph G in Figure 2 (a) shows a simple BQM problem known as MAXCUT. The nodes of G are variables and the edges represent interactions between pairs of variables. A solution to the problem corresponds to an assignment of values (in this case colors) to the nodes of G. This is a *binary* problem because only two values 0 (teal) or 1 (orange) can be assigned to the nodes.

The edges of G are assigned numbers, called *biases*, that express preferences for certain value combinations on endpoint nodes. In this example, a solid edge has negative bias, expressing preference for *same* values (0,0) or (1,1), and a dotted edge has positive bias, expressing preference for *different* values (1,0) or (0,1). Assume that the length of an edge indicates the magnitude of the bias and the strength of the preference. (BQMs also support assignment of biases to nodes, but MAXCUT does not use this feature.)

Each possible solution has a quality score S, computed according to how well the assignment satisfies the preferences expressed by biases. This is a *quadratic* problem because calculation of S incorporates edge and node biases, whereas a *linear* problem only considers node biases. The MAXCUT problem is, given a graph G and its biases, to find an assignment of binary values to nodes that maximizes S.

Variations on this abstract problem arise in many real-world application areas, such as:

- **VLSI circuit design.** Nodes represent circuit components and edge biases represent preferences that components be located on "same" or "different" design layers. An optimal MAXCUT solution assigns components to two layers (0 or 1) in a way that minimizes the cost of wires needed to connect components within and between the layers.



(a) BQM (b) DQM (c) CQM

Figure 2: (a) A solution to a BQM problem with binary variables taking two values (teal and orange). (b) A solution to a DQM problem with discrete variables taking four values (orange, teal, purple, blue). (c) A solution to a CQM problem with constraints, defined on integer variables assigned to gradations of colors.

- **Portfolio allocation.** Nodes represent financial assets available for purchase. Node biases represent expected returns, and edge biases represent price correlations (positive or negative) between asset pairs. A robust portfolio minimizes risk by using diversification to maximize negative correlations within a group of selected assets. An optimal solution to this problem divides the assets into two groups ("select" and "omit'), to maximize return and minimize risk in the selected set.
- **Social network analysis.** Nodes represent people, and edge biases represent friendly and hostile encounters between them. An optimal solution to the "community detection" problem assigns people to two groups to maximizes a score measuring friendly relationships within groups, and hostile relationships between members of different groups.

To elaborate on the third example, suppose the edge biases represent friendly (solid) and hostile (dotted) encounters among citizens of twelfth-century Verona. The computational problem is to assign citizens to the groups Montague (0, teal) and Capulet (1, orange), to maximize the MAXCUT score S. Figure 2 (a) shows one possible solution.

Researchers in social network analysis study the number of "frustrated" edges in an optimal solution — that is, hostile encounters within a group or friendly encounters between members of different groups. For example, Juliet is friendly with both Romeo and her father, but Romeo and Lord Capulet have a hostile relationship: any assignment must frustrate at least one edge of this triangle. In another example, Mercutio has a hostile relationship with both groups,[2] so some edges must be frustrated no matter which group he is in. A high number of frustrated edges in an optimal solution is a sign of *structural imbalance*, which is associated with increased potential for clashes, violence, and perhaps even tragedy.

Discrete and Integer Models Suppose now that Figures 2(b) and 2(c) represent solutions to a network analysis problem involving four social groups in Verona and Florence: Capulet=orange=1, Montague=teal=2, Medici=purple=3, and Albizzi=blue=4. DQMs and CQMs may be defined directly on discrete and integer variables which allows the user to circumvent both problems. In this context, *discrete* refers to categorical values such as colors or surnames, whereas *integer* refers to numerical values.

Although it is technically possible to formulate this problem as a BQM, techniques for doing so would in-

[2] "A plague o' both your houses!"

volve replacing each node in the original graph with four binary nodes (one for each possible value assignment), creating a four-fold increase in problem size. It would also require rewriting the objective function to ensure that at exactly one of each binary combination (node, value) is selected. Using the DQM or CQM solvers allows the user to circumvent the problem of expanding input size and the inconvenience of problem reformulation.

Models with Constraints In this example problem the notational differences between DQM and CQM formulations are small: in DQM the four values are called *cases*, and the input would contain lists of valid cases that can be assigned to each node; in CQM the input would specify valid ranges of integers $[0 \dots 3]$ for each node.

The primary difference between the CQM solver and its companions in HSS is that CQM offers a rich language for expressing *constraints* — that is, rules about what constitutes a *feasible* (i.e., valid) solution to the problem. In constrast, all solutions to (unconstrained) BQMs and DQMs are considered feasible. As before, while it is technically possible to incorporate constraints in objective functions for BQM and DQM formulations, the constraint language of CQM is much more convenient to use. In addition, the direct approach gives the CQM solver a performance edge by allowing it to recognize and avoid infeasible regions of the solution space. Furthermore, representation of more realistic models can greatly improve the practical value of solutions found by the CQM solver.

To illustrate this point, suppose that Figure 2 (c) describes city features including a river (blue line), a duomo (large green square), and two palazzi (square nodes). With integer variable it becomes possible to express constraints involving (linear) sums of node and edge weights, (quadratic) sums of products of nodes and edge weights, and sums of node *values*. Rules such as the following can be expressed using this interface:

1. The two palazzi must be assigned to two different families.
2. The five nodes surrounding the duomo cannot all be from the same family.
3. Every family must be assigned to at least 8 and no more than 12 nodes.
4. No more than half the edges across the river can have endpoints assigned to different families.

Problems with Integer and Real Variables As of April 2022, the CQM solver supports representation of *continuous* models defined on real-valued variables as well as integers and binaries. Models containing real variables are typically found when the values to be assigned to nodes represent locations in space or time. Since the MAXCUT/social networking problem is not of this category, we switch to a new problem known as Job Shop Scheduling (JSS), shown in Figure 3 to illustrate this new CQM feature.

An input to a JSS problem consists of a list of jobs to be performed; there are five jobs (gold, orange, green, blue, teal). Each job is divided into a sequence of tasks (colored blocks numbered 0, 1, 2, 3, 4), of varying durations (indicated by block widths). Each task is performed using a specific machine (A, B, C, D, E) in the shop.

There is one variable per task, and the values assigned to tasks are their start times. The optimization problem is to assign a time to each task so as to minimize the total *makespan* – the time between the start of the first task and the finish of the last task — while obeying two constraints:

1. Within a job, each task i must finish before its successor task $i + 1$ can begin. In Figure 3, it is easy to verify that orange tasks obey this constraint, as do the tasks of other colors.
2. A machine can perform only one task at a time: in the figure, it is easy to see that no machine is assigned tasks that overlap in time.

As with MAXCUT, variations on JSS may be found in many real-world applications, for example:

- **Equipment maintenance scheduling.** The jobs correspond to corporate assets (trucks, helicopters, generators), each scheduled to undergo a series of maintenance operations. The machines correspond to separate facilities containing specialized equipment for these operations. An optimal solution schedules all task times so as to minimize total maintenance time.

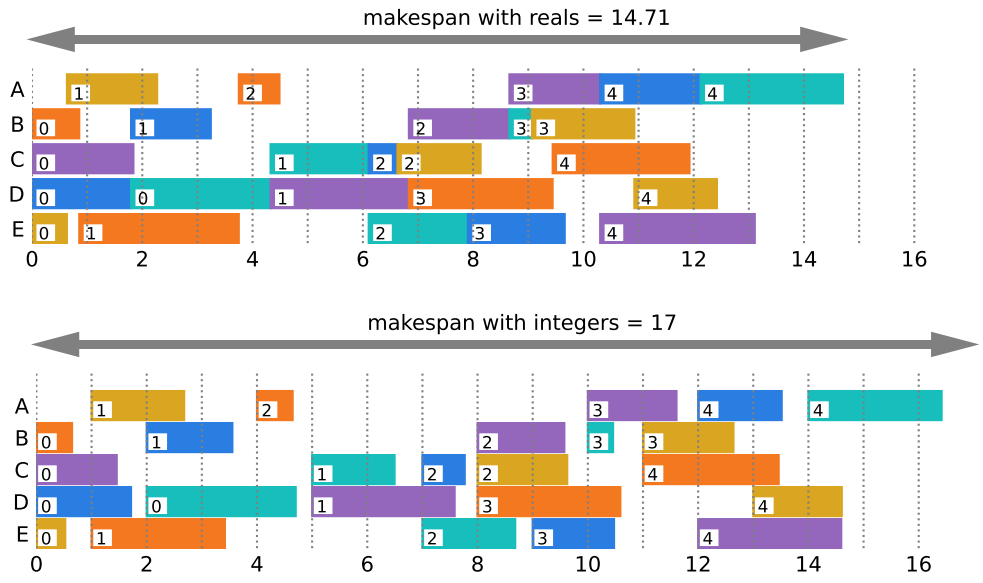


Figure 3: Top: A solution to a JSS problem defined on real variables. Bottom: A solution to the same problem, with real variables replaced by integer variables. Makespan has increased 15 percent.

- **Work crew scheduling.** The "jobs" are construction sites, each consisting of certain tasks (HVAC, plumbing, flooring, paint, etc.) to be performed, in a specified order. The "machines" are specialized work crews that travel from site to site (one site per day) to perform the tasks. The optimal schedule assigns days to work crews, to minimize the time to complete construction at all sites.

- **Airport scheduling.** Arrival of an aircraft at an airport consists of a sequence of steps requiring exclusive use of certain airport resources: approach on path A, land on runway B, taxi across runway intersections C, D, and so forth. The jobs are the aircraft, the tasks are the arrival steps, and the machines are the airport resources and/ or ground crew members necessary to each step. An optimal schedule assigns times to each step to minimize the total time required for all incoming flights to arrive at their gates.

The top panel of Figure 3 shows a solution to a JSS problem defined on real variables (top), and shows a solution to the same problem defined on integers (bottom). In the top version, a task can start immediately after its predecessor ends. In the bottom version, time is divided into discrete intervals, say one hour each (shown on the bottom), and tasks are assigned to the start of each interval. A comparison of the top and bottom solutions shows that requiring each task to start at the top of the hour creates wasted time whenever a task finishes early. In this example, the use of integers instead of reals increases makespan by about 16 percent, from 14.71 to 17.

As a general rule, problems that are naturally defined in terms of real values are best solved using continuous models. At the time of this writing, the CQM solver supports a broader set of integer constraints than real constraints; formulation as an integer model may be the only available option at present. However, development of the CQM solver progresses rapidly and the variety of supported constraint types is expected to increase in future versions.

The toy problems described in this section are intended to illustrate the progressively more powerful modeling capabilities of the DQM and CQM solvers compared to the original BQM solver. However, the HSS is not de-

	BQM	DQM	CQM
Objective Function	linear & quadratic	linear & quadratic	linear & quadratic
Variable Type	binary	discrete	binary, integer, real
Max Values per Variable	2	65,000	2, $\pm 2^{53}$ [1]
Constraint Representation	via penalties	case restriction [2] via penalties	variable bounds integer linear & quadratic equality integer linear & quadratic inequality real linear equality & inequality via penalties
Max Variables [3]	1 million	5,000	500,000
Max Constraints	–	–	100,000
Max Biases [4]	200 million	5 billion	2 billion

Table 1: Solvers in the HSS portfolio provide support for ever-broader categories of problems. Notes: [1] Variables are represented as `dimod.BINARY`, `dimod.INTEGER`, and `dimod.REAL` types. [2] The BQM solver uses case restriction for constraints involving forbidden combinations of values assigned to variables or pairs of variables. [3] In BQM and CQM solvers, the maximum number of variables is also limited by the maximum number of biases; see the documentation for details. [4] For BQM and CQM solvers the number of biases is the number of nonzero weights on all nodes and edges of the input graph; for DQM this is the number of all nonzero weights on all cases assigned to nodes and edges.

signed to solve toy problems, but rather to tackle constrained quadratic optimization problems of industrially relevant complexity and size. Table 1 summarizes the features, problem types, and inputs supported by each solver.

3 Performance Comparison

In addition to ease-of-use and modeling power, the performance of CQM in comparison to its companions BQM and DQM is also of interest: which is the best choice for tackling the job at hand?

An apples-to-apples performance comparison requires problems that can be translated to run on all three solvers. As a general rule, translating "downstream" from BQM to DQM to CQM is straightforward, since their variable domains are increasingly general. However, reformulating problems "upstream" from CQM to DQM to BQM can sometimes be prohibitively complicated.

We have selected three problems that are simple enough to allow easy translation both upstream and downstream: for this reason the selected problems do not fully exercise the expressive power of the CQM solver.

Each problem in our test set is most naturally represented by one specific HSS solver as follows:

- The BQM problem set comprises 15 inputs from the MQLib problem repository of MAXCUT and QUBO inputs [6]. These unconstrained binary inputs represent a variety of application domains and contain $N \in [1200 \ldots 2500]$ variables.
- The DQM problem set consists of 15 inputs from the DIMACS Graph Coloring problem repository [7]. The graph coloring problem is to assign colors to nodes of a graph, so that no two edge endpoints have the same color, in a way that minimizes the total number of different colors used. These inputs come from a variety of applications and have sizes $N \in [74 \ldots 561]$.
- CQM problems consist of 15 randomly generated inputs for the traveling salesperson problem (TSP). The TSP problem is to assign a "visit index" (first, second, etc.) to nodes in a graph, to minimize the total weight of edges between successively visited nodes, under the constraints that each node is visited exactly once and that each index is assigned exactly once. These inputs have sizes $N \in [35 \ldots 63]$, and uniform edge weights in $[1, 2N]$.

As a side note, these problems illustrate a general prop-

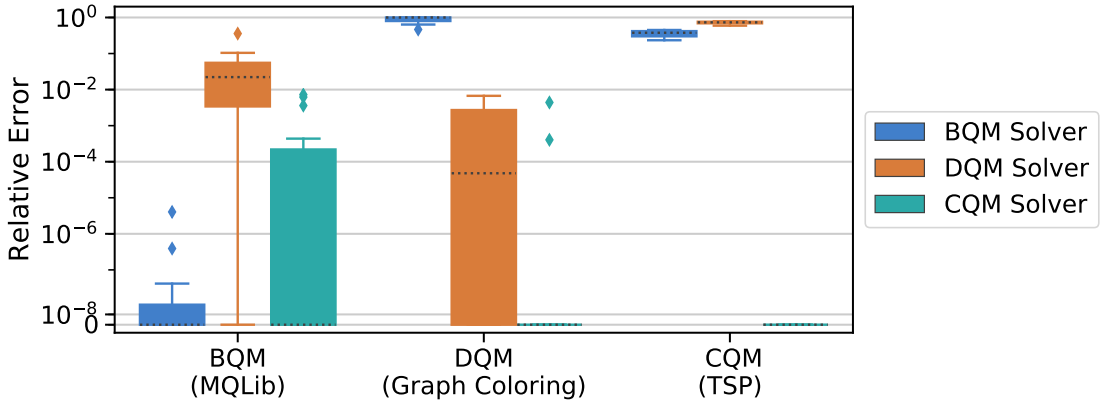


Figure 4: Performance of BQM vs DQM vs CQM on three problem sets. MQLib problems are naturally expressed as BQMs; graph coloring problems are naturally expressed as DQMs; and TSP problems are naturally expressed as CQMs.

erty: increasing the variable complexity from binary to discrete to integer in a formulation tends to decrease the number of variables needed to represent the problem. In this case, TSP inputs must be kept small enough to ensure that their binary representation fits within the BQM size limits shown in Table 1.

As currently deployed, the BQM solver always returns a single solution, while the DQM and CQM solvers may return multiple solutions, depending on input properties and internal configurations. In these tests we record the best-quality solution returned with a five-minute time limit.

For each input: let S_{best} be the best quality score found among all solvers; let S_{worst} be the worst score over all; and let S denote the best score found by a given solver on this input. The error distance $\Delta(x, y)$ denotes the positive difference between two scores, accounting for possible sign differences. The *relative error* is the scaled error distance, $R = \Delta(S_{best}, S)/\Delta(S_{best}, S_{worst})$.

Figure 4 shows the results. The y-axis marks relative errors for each input set (lower is better), and the x-axis shows results for three solvers in each of three input categories. Each boxplot summarizes the distribution of 15 measurements of relative error, one for each input. The area between box endpoints corresponds to the middle 50 percent of the distribution, horizontal lines within the boxes are medians, and lines and individual points outside the boxes show the distribution tails and outliers. Here are some observations.

- **MQLib**. The three left boxes compare solver performance on MQLib inputs, which are most naturally expressed as BQMs. Not surprisingly the BQM solver (blue) performs best, with median $R = 0$, meaning BQM found best solutions on over half its inputs. The DQM (orange) and CQM (teal) solvers are able to match this performance in some cases, with CQM outperforming DQM in terms of worst-case performance.

- **Graph Coloring**. The three center boxes compare performance on graph coloring problems which are naturally expressed as DQMs. On these problems the BQM solver shows worst performance of the three. The DQM solver is able to find good results in some problems, but the more recent CQM solver performs best overall. This outcome reflects an ongoing effort by CQM developers to detect DQM structures and apply newly developed strategies not available in DQM. We expect that CQM will eventually replace the DQM solver in HSS.

- **TSP.** The three rightmost bars compare solver performance on TSP inputs. We see significant performance improvements from the CQM solver. Although all three solvers were able to find feasible solutions to these problems, the ability to directly represent constraints means that the CQM solver does a better job of avoiding time-consuming exploration of the infeasible problem space.

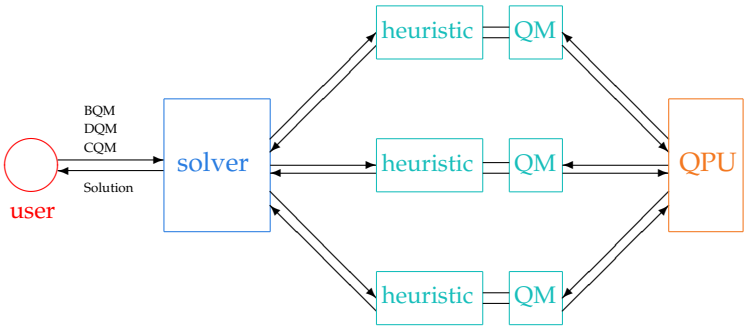


Figure 5: Structure of a hybrid solver in HSS. The front end (blue) reads an input Q and optionally a time limit T. It invokes some number of heuristic solvers (threads) that run on classical CPUs and GPUs (teal) and search for good-quality solutions to Q. Each heuristic solver contains a quantum module (QM) that formulates and sends quantum queries to a D-Wave QPU (orange); QPU responses to these queries may be used to guide the heuristic search or to improve the quality of a current pool of solutions. Before time limit T, the heuristic solvers send their results to the portfolio front end, which, for example, removes duplicates and forwards a subset of solutions to the user.

This test shows the importance of choosing the right solver for the task at hand. The BQM solver shows the best performance on unconstrained binary problems, while DQM and CQM outperform it on discrete and integer problems. CQM outperforms DQM on the discrete problems tested here, and is expected eventually to outperform DQM on all discrete problems. The CQM solver is the best choice for representing and solving constrained optimization problems.

4 Hybrid Workflows

Every solver in the HSS portfolio incorporates a hybrid quantum-classical workflow, as shown in Figure 5. Each solver has a classical front end that reads an input Q and (optionally) a time limit T.[3] It then invokes one or more hybrid heuristic solvers (computation threads) to search for good-quality solutions to Q.

The heuristic solvers run in parallel on state-of-the-art CPU and/or GPU platforms provided by Amazon Web Services[4] (AWS). Each contains a classical *heuristic module* that explores the solution space, and a *quantum module (QM)*, which formulates *quantum queries* that are sent to a back-end Advantage QPU. Replies from the QPU are used to guide the heuristic module toward more promising areas of the search space, or to find improvements to existing solutions. Each heuristic sends its best solutions to the front end before the time limit is reached, and the front end forwards best results to the user.

Accelerating Convergence to Better-Quality Solutions

This arrangement of classical and quantum solvers working in tandem makes possible a phenomenon illustrated in Figure 6, from tests using the CQM solver.

Internal versions of HSS solvers[5] can operate in two modes, called workflows: in the *hybrid* workflow (orange) the QM module is active; in the *heuristic* workflow (blue) the QM module is disabled and the classi-

[3] If none is provided by the user, a default time that depends on input size is used.

[4] Amazon Web Services is a trademark of Amazon Technologies, Inc.

[5] These tests were carried out using a "laboratory" version of the CQM solver, which runs single-threaded on a single platform. In contrast, the HSS production solvers available to the public are deployed for multi-threaded use in the cloud. For reasons of efficiency they do not offer the heuristic workflow option to users; in addition, accurate runtime control and reporting is problematic. For these reasons, the results of this section may differ somewhat from those observed in deployed systems, though we expect that latter to be generally more efficient.

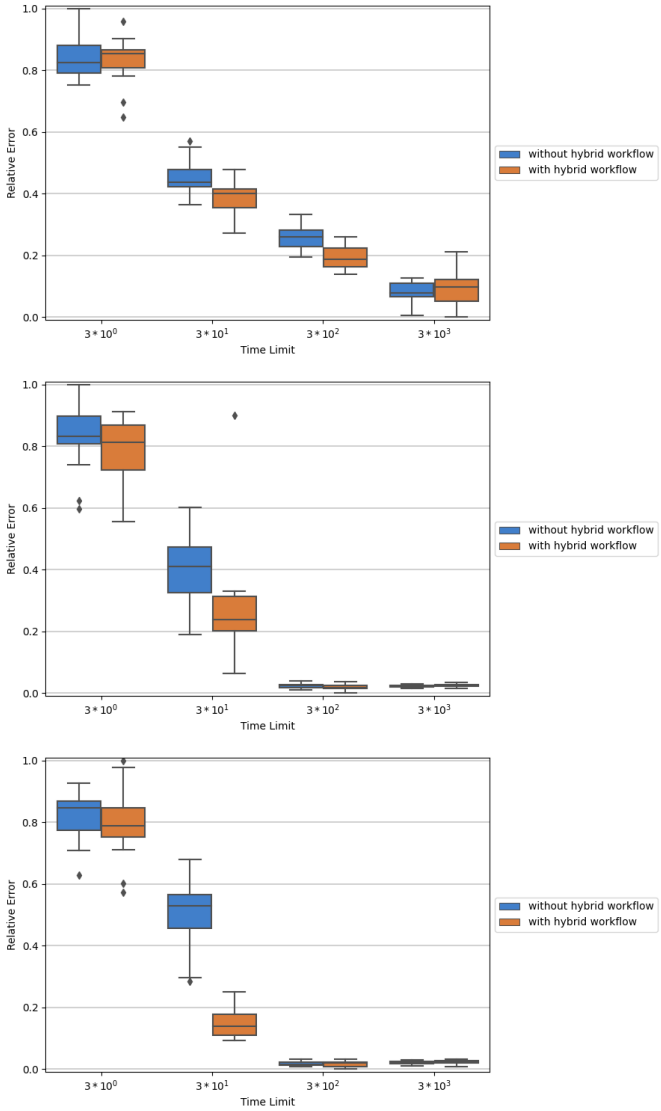


Figure 6: Hybrid workflows sampled at different time limits T can exhibit *hybrid acceleration*, converging to better solutions faster than purely classical workflows.

cal heuristic works alone. Note that for reasons of efficiency, the heuristic workflow option is not available in HSS solvers that are deployed for public use.

The three panels correspond to three individual inputs. The y-axis corresponds to relative errors observed, at time limits T between 3 seconds and 3000 seconds (50 minutes), marked on the x-axis. The boxplots summarize sampled relative errors over 75 independent trials for each workflow and each time limit. Note that the hybrid workflow converges more quickly to better results, returning better solutions at some time limits. We call this phenomenon *hybrid acceleration*.

The nature of hybrid acceleration varies from input to input. For example the top panel shows small but steady differences in solution quality over the range of sampled times, while the bottom panel shows significant acceleration around $T = 30$ seconds, that disappears by $T = 300$ seconds. As a general rule, hybrid acceleration cannot be observed at higher T, when both workflows have had enough time to converge to the same (presumably optimal) solutions.

Like its companions in the HSS, the CQM solver is designed in such a way that the QPU always has a chance to accelerate convergence in this way. This does not necessarily mean that acceleration always occurs: some inputs are easy enough to be solved heuristically without needing a quantum boost, and some inputs may have complex structures that resist acceleration.

5 Remarks

This report introduces a new CQM solver for constrained quadratic models to D-Wave's hybrid solver portfolio. The CQM solver can model problems defined on integer variables, and offers an easy-to-use interface that supports direct representation of problem constraints. These capabilities bring a much larger region of the constrained optimization problem space within scope of the HSS.

Because CQM is able to represent constraints explicitly, it tends to be more efficient than its companions at finding good-quality feasible solutions to constrained problems. However, unconstrained binary problems can be more efficiently solved by the BQM solver.

Like its companions in HSS, the CQM solver incor-

porates queries to an Advantage quantum processor working as a back-end query processor. This combination of classical and quantum solution methods working in tandem can exhibit a phenomenon known as *hybrid acceleration*, whereby the hybrid workflow can find better solutions faster than a purely classical workflow.

References

[1] Visit *cloud.dwavesys.com/leap*.

[2] Visit *docs.ocean.dwavesys.com*. Search terms: Using Leap's Hybrid Solvers.

[3] Visit *docs.ocean.dwavesys.com*. Search terms: Ocean Software: Ocean documentation: dwave-hybrid.

[4] Visit *dwavesys.com/learn/featured-applications*.

[5] Visit *docs.ocean.dwavesys.com*. Search terms: structural imbalance.

[6] Dunning et al., What works best when? A systematic evaluation of heuristics for Max-Cut and QUBO, *Informs Journal on Computing*, 15 Oct. 2018. Inputs may be downloaded from *github.com/MQLib/MQLib*.

[7] D. S. Johnson and M. A. Trick, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, Oct. 11, 1993. Inputs may be downloaded from mat.tepper.cmu.edu/COLOR/instances.html.