The Quantum Computing Company™

Hybrid Solver for Constrained Quadratic Models

WHITEPAPER

Summary

This whitepaper presents an overview introduction to the constrained quadratic model (CQM) solver, the newest addition to D-Wave's hybrid solver service (HSS). The CQM solver provides a convenient notation for modeling constraints, which greatly expands the variety of optimization problems that fall within scope of the HSS. A small performance comparison of the BQM, DQM, and CQM solvers in the HSS highlights the importance of choosing the right tool for the job at hand. Like all solvers in the HSS, CQM integrates both quantum and classical solution methods, leveraging the best features of both to achieve faster convergence to better solutions on some problems.

1 Introduction

D-Wave's hybrid solver service (HSS) contains a portfolio of heuristic solvers that leverage quantum and classical methods to solve problems much larger than can fit on Advantage™ quantum systems.

To date, the HSS has held solvers for two problem types: the binary quadratic model (BQM) solver for problems defined on binary values (0,1); and the discrete quadratic model (DQM) solver for problems on nonbinary values (such as red, orange, yellow, green).

This report describes the newest member of the HSS, a solver for constrained quadratic models (CQMs): this solver can represent problems defined on binary and integer variables, and it offers a convenient notation for expressing problem constraints. Here is an overview.

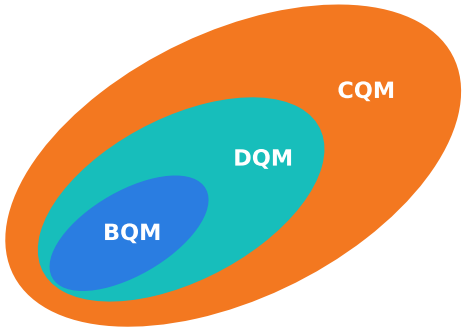


Figure 1: Increasing scope of solvers in the HSS.

- As illustrated in Figure 1, the CQM solver is a continuation of D-Wave's efforts to expand the space of optimization problems that fall within scope of the HSS. Section 2 illustrates the types of applications that can be formulated using each solver.
- Section 3 presents a small performance comparison highlighting the benefits of choosing the right tool for the job at hand. In our tests, the BQM solver performed best on unconstrained binary problems. The DQM and CQM solvers were more efficient on nonbinary problems, with CQM dominating on constrained nonbinary problems.
- Like its companions in the HSS, the CQM solver implements a classical front-end that works in tandem with a quantum back-end, an Advantage quantum system. Section 4 describes this arrangement and shows how the hybrid workflow can ex-

hibit superior performance compared to a purely classical workflow.

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

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

2 Quadratic Models for Real-World Problems

The binary quadratic formulation used by the BQM solver (which is native to all D-Wave quantum processors) arises naturally in many application areas; a simple example is presented in the next section.

More generally, however, many problems must first be translated to this binary formulation for direct solution on the quantum hardware. Researchers have built up a large "cookbook" of suitable problem translations: see [4] to learn about more than 250 applications that have been demonstrated to run on D-Wave platforms. In principle, any NP-hard problem can be efficiently translated to a BQM; but in practice it can be challenging to come up with an efficient translation, in the sense that solver performance can depend critically on translation quality.

The DQM and CQM solvers are part of an outreach effort by D-Wave developers to expand the scope of problems that can be modeled directly within the HSS, without requiring further translation to BQMs. This effort both lowers barriers to use and can sometimes produce better performance. To illustrate this point, the types of problem formulations — called models — that each solver supports are illustrated below.

Binary Quadratic Models We start with a simple BQM represented by a graph G, as shown in Figure 2 (a). The nodes of G are variables, and the edges represent relationships between pairs of variables. This is a *binary* problem because the nodes can be assigned one of two values, in this example either 0 (teal) or 1 (orange).

A BQM input has real-valued weights, called *biases*, that may be attached to the nodes and edges of G. An edge bias expresses a preference for certain value assignments on its endpoint nodes. In Figure 2 (a), a solid edge indicates a preference for *same* values (0,0) or (1,1) on the endpoints, and a dotted edge indicates a preference for *different* values (1,0) or (0,1). In this example, the length of an edge corresponds to a stronger (shorter) or weaker (longer) preference. For simplicity we assume here that all node biases are set to zero, indicating no preference.

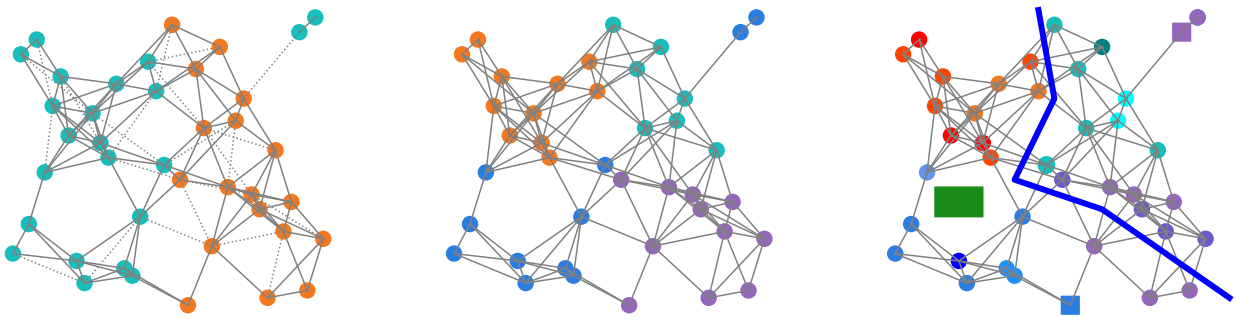
In a solution that assigns values to nodes, we say an edge is *satisfied* if its endpoints match its preference, otherwise the edge is *frustrated*. Each possible solution has a quality score S, equal to the (positive) sum of biases on satisfied nodes and edges, plus the (negative) sum of biases on frustrated nodes and edges. The computational task is to find an assignment of binary values to nodes that maximizes this quality score.

For example, suppose the edge biases represent friendly (solid) and hostile (dotted) relationships among citizens of Verona in the 1300s. The computational problem is to assign every citizen to one of two social groups — the Montagues (0, teal) or the Capulets (1, orange) — to maximize the score S as defined above. Figure 2 (a) shows one possible assignment of citizens to these two groups.

Note that it may not be possible to find an assignment that satisfies every edge in the graph. For example, Juliet is friendly with both Romeo and her father, but Romeo and Lord Capulet have a hostile relationship: no assignment of binary values to this triangle can satisfy all three edges. For another example, Mercutio has a hostile relationship with both groups,[1] so some edges must be frustrated no matter which group is assigned to him.

In the field of social network analysis, the number of frustrated edges in an optimal solution is a measure of *structural imbalance* in the network. Structural imbalance is associated with the potential for unrest and clashes within a given social group: see [5] to learn more.

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



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

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

Discrete Quadratic Models In DQM formulations, nodes can be assigned many values, not just two. For example, suppose that Figure 2 (b) represents a social network modeling four social groups located in Verona and Florence: Capulet (orange), Montague (teal), Medici (purple), and Albizzi (blue).

The four values are called *cases*. The DQM solver supports models in which lists of allowed cases can be attached to each node, and allowed pairs of cases can be attached to each edge. Restrictions of these lists can be used to define rules about forbidden assignments or combinations, for example:

1. The two nodes at upper right can only be assigned to Medici or Albizzi.

2. No edge can have the combination (Capulet, Medici) on its endpoints.

Rules of this type, called *constraints*, are used to define *infeasible* solutions, which are considered invalid and must be discarded. Note that the solution in Figure 2 (b) contains one violation of constraint 2, and is therefore infeasible.

In unconstrained problems, all solutions are feasible; in constrained problems some solutions are considered infeasible. The computational problem is to find the optimal feasible solution as defined by the quality score S.

Constrained Quadratic Problems The CQM solver can model problems defined on binary and integer values. Furthermore, the CQM solver provides direct support for expressing a variety of constraints, including constraints that are expressed arithmetically. This represents a significant expansion of modeling power over the BQM and DQM solvers.

For example, suppose Figure 2 (c) describes a city containing a river (blue line) and a duomo (green square). The social network also contains two square nodes that are not directly connected. Integer values assigned to the nodes correspond to lighter and darker shades within each color group: Capulet (oranges [1..10]), Montague (teals [11...20]), Medici (purples [21..30]), and Albizzi (blues [31..40]), perhaps representing seniority levels within a group.

Here are some examples of constraints — involving both sums of nodes and edges, and sums of integer values assigned to nodes and edges — that can be modeled using the CQM solver.

1. The square nodes must be assigned to different groups, with total value greater than 30.

2. The values of the five nodes surrounding the duomo must sum to exactly 150.

3. The total value assigned to each color group must be with one of its average (5.5, 15.5, 25.5, 35.5).

4. Exactly nine edges across the river must have same-colored endpoints.

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 designed to solve toy problems, but rather to tackle constrained optimization problems of industrially-relevant complexity and size. Table 1 summarizes the features, problem types, and inputs supported by each solver, with annotations and details in the caption.

3 Performance Comparison

In addition to greater 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?

For an apples-to-apples performance comparison, we use problems that can be easily translated to run on all three solvers. As a general rule, translating a problem "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 be prohibitively complicated.

To address this issue we have selected three problems that are simple enough to allow easy translation both upstream and downstream: note that this means the problems selected 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 Max Cut and QUBO inputs [6]. These unconstrained binary inputs represent a variety of application domains and have sizes $N \in [1200, 2500]$.
- The DQM problem set consists of 15 inputs from the DIMACS Graph Coloring problem repository [7]. The graph coloring problem is to assign a set of 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 colors used. These inputs come from a variety of applications and have sizes $N \in [74, 561]$.
- CQM problems consist of 15 randomly generated inputs for the traveling salesperson problem (TSP). The TSP problem is to assign "visit times" 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 time slot is assigned to exactly one node. These inputs have sizes $N \in [35, 63]$, and uniform edge weights in range $[1, 2N]$.

As a side-note, these problems illustrate a general observation that increasing the range of values assigned to variables (from binary to discrete to integer) tends to produce more compact problem representations, so that fewer variables are needed to express problems of similar complexity.

In all tests we record the best solution returned by each solver within a five-minute time limit; 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.

For each input, let S_{best} be the best solution 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. Let the error distance $\Delta(x, y)$ denote the positive distance 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 3 shows the results. The y-axis shows relative errors for each inputs, and the x-axis marks results for three solvers in each of three input categories. Each boxplot summarizes the distribution of 15 data points, corresponding to relative errors observed over 15 inputs for this problem and this solver. The area between box endpoints corresponds to the middle 50 percent of the distribution, horizontal lines within the boxes correspond to medians, and lines and individual points outside the boxes show 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) shows best performance, with

	BQM	DQM	CQM
Objective Function	linear & quadratic	linear & quadratic	linear & quadratic
Variable Type **Max Values per Variable**	binary 2	discrete 65,000	binary & integer 2, $\pm 2^{53}$ [1]
Constraint Representation	via penalties	case restriction [2] via penalties	variable bounds linear & quadratic equality linear & quadratic inequality via penalties
Max Variables [3]	1 million	5,000	5,000 (nonbulk) [4] 500,000 (bulk)
Max Constraints	–	–	100,000
Max Biases [5]	200 million	3 billion (nonbulk) 5 billion (bulk)	750 million (nonbulk) 2 billion (bulk)

Table 1: Solvers in the HSS portfolio provide support for ever-broader categories of problems. Notes: [1] Integers are represented as `dimod.INTEGER` variable 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] The user can select regular (nonbulk) service for low latency and high throughput on smaller jobs that need real-time response, or bulk service for higher latency and lower throughput on larger jobs that can tolerate start-up delays. [5] 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.

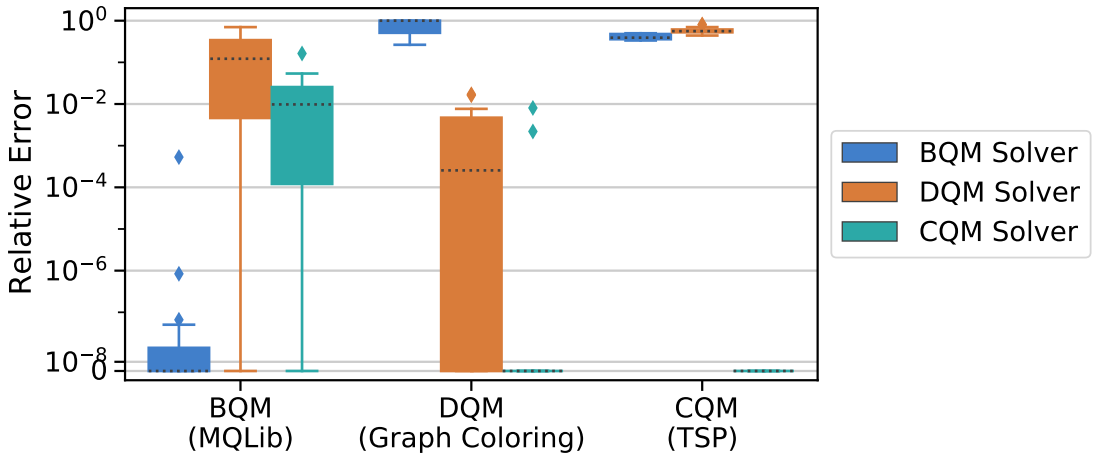


Figure 3: Performance of BQM vs DQM vs CQM on three problem sets. MQLib problems are naturally expressed as BQMs, graph coloring problems are naturally expressed as DQMs, and TSP problems are naturally expressed as CQMs.

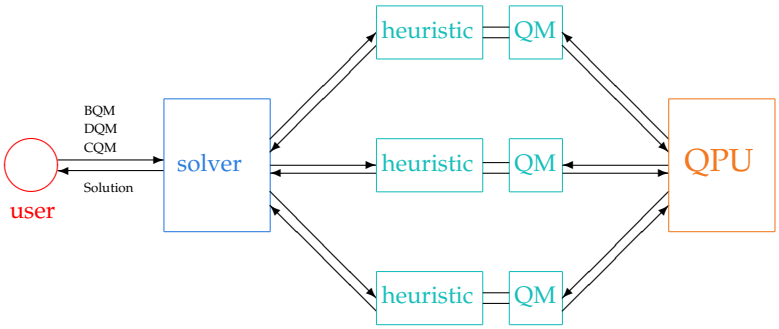


Figure 4: Structure of a hybrid solver in HSS. The front end (blue) reads an input Q and optionally a time limit T. It invokes some number of heuristic solvers (threads) that run on classical CPUs and GPUs (teal) and search for good-quality solutions to Q. Each heuristic solver contains a quantum module (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 (e.g.) removes duplicates and forwards a subset of solutions to the user.

median relative error $R = 0$, meaning BQM found best solutions on over half its inputs. The DQM (orange) and CQM (teal) solvers are able to solve at least a few inputs well ($R = 0$), but median relative errors are near $R = 0.1$ and $R = 0.01$, respectively.

- **Graph Coloring**. The three center boxes compare performance on graph coloring problems which are naturally expressed as DQMs. On these discrete 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, which incorporates new code speedups for both discrete and constrained problems, shows best performance, returning $R = 0$ on all but two inputs.

- **TSP.** The three rightmost bars compare solver performance on TSP inputs. Here 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.

This test shows the importance of choosing the right solver for the task at hand. The BQM solver shows best performance on unconstrained binary problems, while DQM and (especially) CQM outperform it on discrete and integer problems. The CQM solver is the best choice for both 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 4. Each solver has a classical front end that reads an input Q and (optionally) a time limit T.[2] 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 (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 im-

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

provements 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 5, from tests using the CQM solver.

Internal versions of HSS solvers[3] 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 classical 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.

In all panels we observe a general property of optimization heuristics, that solution quality tends to improve as computation time grows. We also 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 de-

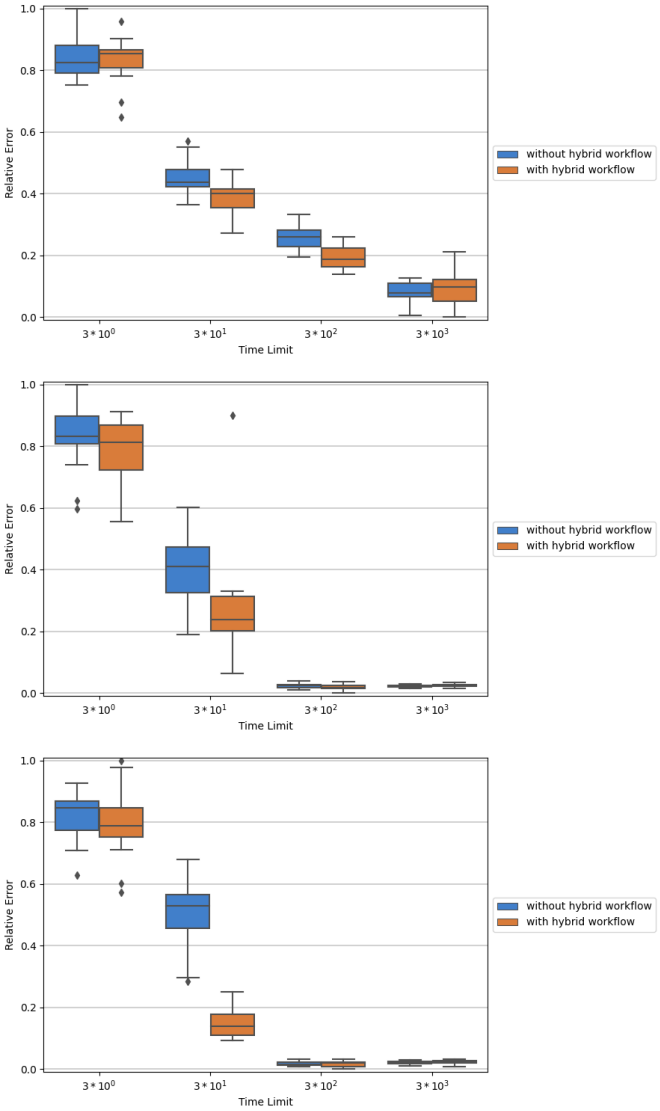


Figure 5: Hybrid workflows sampled at different time limits T can exhibit *hybrid acceleration*, converging to better solutions faster than purely classical workflows.

[3] 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; as well, 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.

signed 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 incorporates 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.