The Quantum Computing Company™

Hybrid Solver for Discrete Quadratic Models

WHITEPAPER

Summary

The discrete quadratic model (DQM) solver, a new member of D-Wave's hybrid solver service (HSS), is introduced. In contrast to binary quadratic model (BQM) solvers that read problems defined on two output values (0,1), the DQM solver can read problems defined on discrete sets of output values, such as $(A \ldots Z)$.

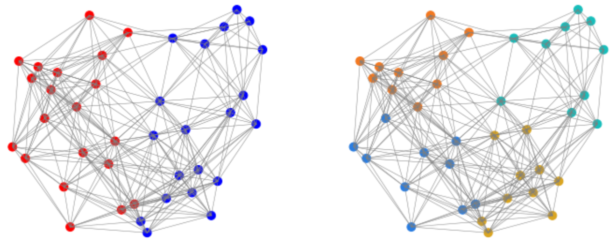

Figure 1: Left: A solution to a binary graph partitioning problem with $k = 2$. Right: a solution to a discrete graph partitioning problem with $k = 4$.

1 Introduction

D-Wave's hybrid solver service (HSS) contains a collection of heuristic solvers that combine quantum and classical methods to solve problems much larger than can fit on current-generation Advantage™ quantum computers. To date, all of the solvers in HSS work on *binary* problems, with variables that can be assigned one of two output values, e.g. $\{0,1\}$. Inputs for these solvers use standard format known as a binary quadratic model (BQM).

This report introduces a new member of HSS, a portfolio solver for *discrete* quadratic models[1] (DQMs). DQMs generalize BQMs by allowing variables to be assigned more than two values.

Figure 1 illustrates the difference using a graph partitioning problem: given a graph G and an integer k, partition the nodes of G into k equal-sized groups to minimize the total weight of edges between different groups.[2] The graph on the left shows a binary version of the problem with $k = 2$; the graph on the right shows a discrete version with $k = 4$.

Several real-world applications based on graph partitioning have been implemented on D-Wave QPUs and hybrid solvers, including: modeling molecular structures [1, 2], analyzing social networks [3, 4], and developing statistical clustering methods [5].

In these and similar cases, discrete problems have been formulated as BQM inputs for the quadratic unconstrained binary optimization (QUBO) problem, for example using a string of k binary variables to represent one discrete variable taking k values, such as 0001, 0010, 0100, 1000 to represent $k = 4$ (additional constraints are needed to ensure that invalid patterns such as 0110 are suppressed). The DQM formulation removes this ne-

[1] For search engine purposes the official name of this solver is `hybrid_discrete_quadratic_model_version1`.

[2] We assume here that the weight on each edge is proportional to its length in the figure, and that the number of nodes is a multiple of k. The example solutions shown are not necessarily optimal.

cessity: instead, DQM solvers are designed to read inputs for the quadratic unconstrained discrete optimization problem (QUDO), and to directly explore the discrete solution space, with invalid bit patterns omitted. Furthermore:

1. DQM solvers can read discrete problems with up to 5,000 nodes and 10,000 groups per node, up to a maximum of 2 billion total input weights. Section 2 shows how DQM problems are formulated.
2. Like all hybrid solvers in HSS, DQM solvers have the capability to leverage queries to the Advantage quantum processing unit (QPU), to accelerate convergence to good-quality solutions. Section 3 demonstrates this phenomenon.

We believe these benefits of greater convenience in problem formulation, high input capacity, and hybrid acceleration will further broaden the scope of usability for the hybrid solvers in HSS.

The next section presents a simple example to illustrate how any QUDO input can be formulated as a DQM. Section 3 gives an overview of hybrid solver structure and demonstrates that DQM solvers can exhibit hybrid acceleration.

2 Formulating a DQM

Given a graph G containing n nodes and m edges, the *graph coloring* problem is to find a *valid* assignment of colors to nodes that uses a minimum number of different colors: in a valid coloring, no two nodes sharing an edge can have the same color. Real world applications of graph coloring include:

- **Radio frequency assignment**. The colors correspond to radio frequency channels, and the problem is to assign a minimum number of channels to mobile transmitters, so that any pair of transmitters within a certain threshold distance are assigned different channels.
- **Scheduling** . The colors correspond to gates in an airline terminal. The problem is to assign a minimum number of gates to scheduled flights such that no two flights on the ground at the same time are assigned the same gate.

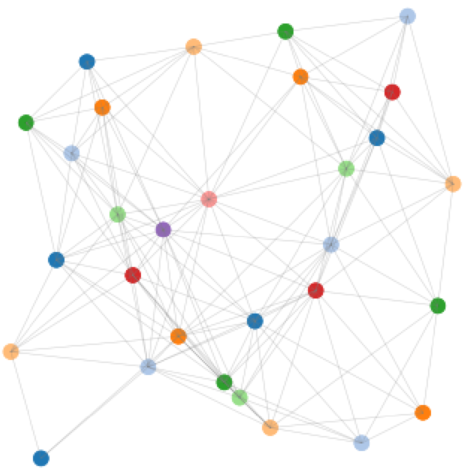

Figure 2: A graph coloring problem with $n = 30$ nodes and $m = 186$ edges. This solution uses $c = 8$ colors.

- **Timetabling**. The colors correspond to final exam time slots: the problem is to assign exam times to courses such that any pair of courses with at least one student attending both must be assigned different times.

Graph coloring problems have been implemented on D-Wave QPUs and hybrid solvers [6–9]; however, as with graph partitioning, this previous work has been based on BQMs.

Figure 2 shows an example solution for a graph G with $n = 30$ nodes and $m = 186$ edges. The node degree d is the highest number of edges per node: here $d = 17$, with the orange node near the center. Let c denote the number of different colors used in the solution. Although it is always possible to color a graph using $c = d + 1$ colors, the figure shows a better solution using just $c = 8$ colors. (Finding an even better coloring is left as an exercise for the reader.)

This problem can be formulated as a DQM using tools in D-Wave's Ocean developers' toolkit. Starting with an instantiated `dqm` object, our example input is constructed as follows (see [**dw-dqm-docs**, **dw-dqm**] for details and more examples).

Nodes. Use `dqm.add_variables` to create the $n = 30$ nodes.

Cases. The color values that can be assigned to nodes (i.e., variables) are called *cases*. Since $c \leq 18$ we represent colors using integers $[1 \ldots 18]$. There are $540 = 30 \times 18$ possible combinations of nodes and colors. (The DQM format allows cases to be customized to individual variables, e.g. restricting node x to just $[red, green, blue]$, but this option is not pursued here.)

Edges. Use `dqm.add_quadratics` to specify the edges and cases. The number of `quadratic` terms in the problem is the total number of edges m, multiplied by number of possible combinations of cases on its two endpoints.

Biases. A *bias* is a numerical value assigned to a node or edge in the DQM; all biases are set to 0 by default but can be modified as needed.

Some biases come from the problem definition, such as the edge weights in the graph partitioning example of Section 1. The basic graph coloring problem does not have weights, but variations might: for example, each node x could be weighted according to its density d_x, and the goal could be to find a coloring to minimize the weighted sum of color assignments $W = \sum_x d_x c$ (this variation is not further pursued).

In addition, a bias can be a numerical penalty assigned to certain combinations of cases, nodes and edges, to discourage their appearance in output solutions. Since HSS solvers look for minimum-cost solutions, penalties should be positive numbers. Graph coloring requires two types of biases:

- Since a valid coloring requires that no two nodes joined by the same edge have the same color, we assign *quadratic* biases to discourage this event. Suppose the penalty is P (considerations in choosing penalty values are discussed below). Use `set_quadratic` to assign p_{quad} to quadratic terms corresponding to a violation of this constraint; there is one such term for each edge (x, y) and each color c.
- To minimize the total number of different colors assigned to nodes, we use *linear* biases on nodes to discourage assignments of higher-numbered colors and thus encourage solutions with a concentration of lower-numbered colors. For each node n and each color c, use `set_linear` to assign a penalty such as $p_{lin} = c^2$ to the combination (n, c).

Max number of variables n:	5000
Max number of cases per variable:	10,000
Max biases:	2 billion

Table 1: Input parameter bounds for current-generation DQMs.

Once formulated, the DQM input can be submitted to the DQM solver in HSS. Table 1 shows size and parameter limits for current-generation DQM solvers. The total input size corresponds to the total number of linear plus quadratic biases in the problem: in addition to these individual limits, the total input size cannot exceed two billion biases.

Choosing penalties. The choice of suitable penalties and penalty functions when formulating optimization problems for heuristic solvers is a perennial (i.e unsolved) topic in optimization research [12, 13]. A few guidelines are very briefly discussed here.

First note that the quadratic biases are used to enforce a *hard constraint*, so-called because solutions that violate this constraint are invalid and must be discarded. The linear biases enforces a *soft constraint*, in the sense that fewer colors are preferred but the number of colors used does not determine validity. As a general rule, penalties for hard constraints should be set higher than those for soft constraints, to ensure that output solutions are valid.

Penalties on the soft constraints could be set lower or higher, for example using $p_{lin} = c$ or $p_{lin} = c^3$. Setting stronger penalties can accelerate the discovery of better-quality solutions; but on the other hand, setting penalties too high tends to break the solution space into disconnected regions and inhibit exploration over the full space of possibilities. General advice about penalty-setting for DQMs and BQMs may be found in D-Wave documentation and the Leap Community discussion boards [**dw-examples**, 15].

An alternative approach in this case is to omit the soft constraints and instead use iterative tests with varying values of c. That is, instead of building just one input that encourages few colors in the output, create a series of inputs with hard limits $c \in [3, 4, \ldots d + 1]$ and send them to the DQM solver one-by-one, until a valid coloring is found. This removes the need to search for a suitable penalty function but adds more time to test

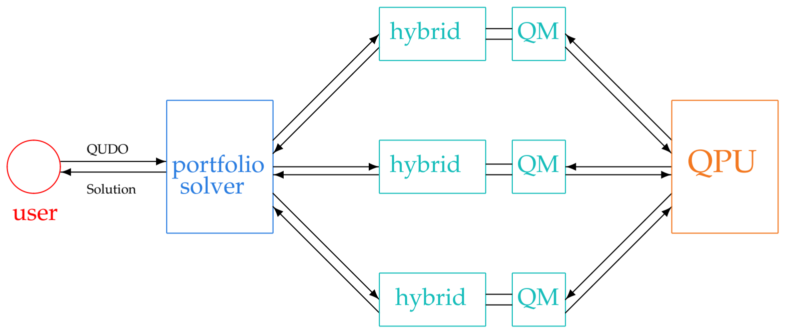

Figure 3: Structure of a portfolio solver in HSS. The portfolio front end (blue) reads an input Q formulated as a quadratic unconstrained discrete optimization problem (QUDO), and optionally at time limit T. It invokes some number of hybrid solvers running on classical CPUs and GPUs (teal), to find solutions to Q. The hybrid solvers contain a quantum module (QM) that formulates and sends quantum queries to a D-Wave quantum processor (orange). At time limit T, the hybrid solvers send their results to the portfolio front end, which forwards solutions to the user.

different values of c.

3 DQM solver features

Once formulated, a DQM input can be submitted to the DQM portfolio solver in HSS. This section presents a brief summary of structure and solver workflows for all HSS solvers, and shows how the hybrid approach can outperform a purely classical version of the same solver; for more about general properties of HSS solvers see [16].

Figure 3 shows the overall structure of HSS. HSS contains a collection of *portfolio solvers*, which read user inputs and (optionally) a time limit,[3] and then invoke one or more hybrid solvers to work in finding good quality solutions to the input. Current versions of HSS contain two types of portfolio solvers, which accept either binary inputs or discrete inputs.

The discrete hybrid solvers in the HSS portfolio run in parallel on a collection of CPU and/or GPU platforms provided by Amazon Web Services (AWS). Each hybrid solver contains an implementation of a classical heuristic that explores the discrete solution space defined by the input, together with a quantum module (QM) that formulates some number of *quantum queries* that are sent to an Advantage QPU. Replies are used to guide the classical heuristic toward more promising areas of the search space. Each hybrid solver sends its best solutions to the portfolio solver before the time limit is reached, and the portfolio front end forwards the solutions to the user.

Panel	Linear Biases	Quadratic Biases
top	18,336	226,560
center	29,072	618,728
bottom	56,100	162,800

Table 2: Number of DQM biases for the three inputs in Figure 4.

Hybrid acceleration

All of the hybrid solvers in the HSS contain a QM that sends quantum queries to a D-Wave QPU standing online, currently either a D-Wave 2000Q or an Advantage system. The QM receives replies and formulates those replies into suggestions for the classical solver component about promising regions of the search space to explore next. (Note that technically the QM may also be considered hybrid because it uses classical code to carry out such tasks as formulating queries, and selecting replies to be mapped to their heuristic components.)

This arrangement makes possible a phenomenon we call *hybrid acceleration*, illustrated in Figure 4. Internal versions of HSS solvers can operate in two modes, called workflows: in the *hybrid* workflow (blue) the QM module is active; in the *heuristic* workflow (orange) the QM module is disabled and the classical heuristic works alone.[4]

The three panels show results for three input graphs

[3]A default lower bound time limit depending on input size may be used instead.

[4]For reasons of efficiency and interface simplicity, choice of workflow mode is not supported in HSS solvers available to the public.

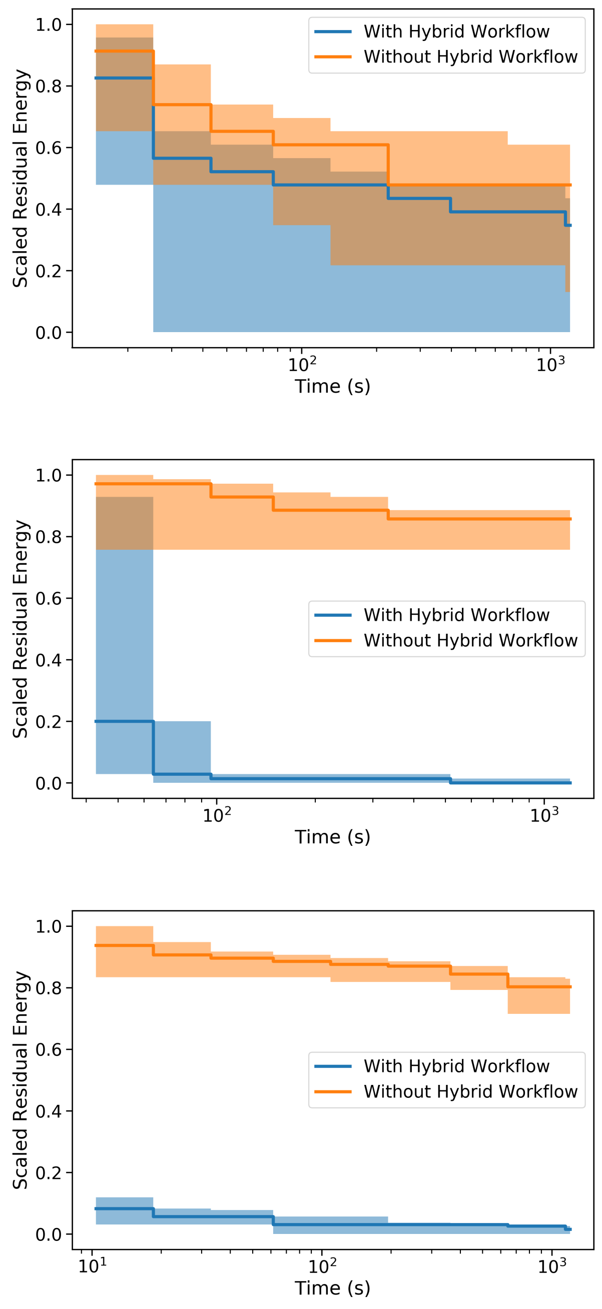

Figure 4: Burndown charts

of sizes shown in Table 2. The y axis marks solution quality returned by each workflow, as measured by problem biases (i.e. penalties), scaled to the largest and smallest values observed in the presented data.

The x axis in each panel shows a range of computation times, from the default lower limit determined by input size, to an upper limit of 20 minutes (1200 seconds). For each input, solution quality is sampled at several points t within these time limits, using three independent trials for each workflow.

For each t and each trial, the DQM solver returns a sample of s solutions, where s is determined by internal solver configuration, typically $s \in [1, \ldots, 50]$. We filter the full sample by recording the five best solutions, for a total of 15 sampled solutions for each t; then at each t we record the best 15 solutions found so far. The orange and blue lines show how median solution quality improves over time, and the shaded regions show the range of solution quality found in these filtered samples.

In each panel we can observe a general property of optimization heuristics, that solution quality tends to improve over time. We also observe that the hybrid workflow (blue), with the QM activated, shows faster convergence to better solutions than the heuristic workflow (orange) with the QM disabled. We call this phenomenon *hybrid acceleration*.

Like all solvers in the HSS portfolio, the DQM solver is designed in such a way that the QPU always has a chance to contribute to the solution search. This does not necessarily imply that hybrid acceleration occurs with every input sent to the QPU: some inputs are easy enough to be solved classically without needing a quantum boost, and some inputs may contain complexities that cannot be captured by the QM. As well, the nature and timing of this acceleration varies from input to input: for example the top panel shows relatively small differences in solution quality over the range of measured runtimes, while the bottom panel shows that acceleration has taken place sometime before the lower-bound time limit set for this problem.

4 Conclusions

This report introduces a new solver in D-Wave's hybrid solver portfolio, which reads discrete rather than binary inputs, formulated as DQMs. A small example showing how to formulate DQMs is presented, together with an overview of solver structure and a demonstration that the DQM solver can exhibit hybrid acceleration: that is, the Advantage QPU can make positive contributions to the quality of solutions obtained.

References

[1] Djidjev et al., "Graph partitioning using quantum annealing on the D-Wave system," PMES'17 (2017).

[2] H. Ushijima-Mwesigwa et al., "Graph partitioning using quantum annealing on the D-Wave system," arXiv:1705.03082.

[3] C. Negre et al., "Detecting multiple communities using quantum annealing on the D-Wave System," arXiv:1901.09756.

[4] J. Ambrosiano et al., *Using the D-Wave 2X quantum computer to explore the formulation of global terrorist networks* (LANL Technical Report LA-UR-17-23946, 2017).

[5] F. Neukart et al., "Quantum assisted cluster analysis," arXiv:1803.02886.

[6] A. Wieckowsky et al., "Disorder-assisted graph coloring on quantum annealers," Phys. Rev. A **100** (2019).

[7] M. Dineen et al., "Finding the chromatic sums of graphs using a D-Wave quantum annealer," The Journal of Supercomputing **75**, 4811–4828 (2019).

[8] Z. Tabi et al., "Quantum optimization for the graph coloring problem with space-efficient embedding," arXiv:2009.0731 (2020).

[9] S. Srivastava and V. Sungaraghavan, "Box algorithm for the solution of differential equations on a quantum annealer," Phys. Rev. A **99** (2019).

[12] K. Bryan and Y. Shibberu, "Penalty functions and constrained optimization," visit `http://www.bioinfo.org.cn/~dbu/AlgorithmCourses/Lectures/PenaltyFunction.pdf` and `https://www.cs.cinvestav.mx/~constraint/papers/yeniay05.pdf`.

[13] O. Yeniay, "Penalty function methods for constrained optimization methods with genetic algorithms," Mathematical and Computational Applications **10**, 45–56 (2005).

[15] "D-wave leap community," `https://support.dwavesys.com/hc/en-us/community/topics`.

[16] C. McGeoch, P. Farré, and W. Bernoudy, *D-Wave Hybrid Solver Service plus Advantage: Technology Update* (D-Wave TR 14-1084A-A, 2020).