D:Wave
The Quantum Computing Company™

Measuring Performance of the Leap Constrained Quadratic Model Solver

TECHNICAL REPORT

2022-11-02

Overview

In this technical report we provide an overview of the performance of the Constrained Quadratic Model solver available as part of D-Wave's hybrid solver service. We define a variety of problems based on widely used problem types and benchmark the most recent update of the Constrained Quadratic Model Solver against previous versions.

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

14-1065A-A
D-Wave Technical Report Series

Notice and Disclaimer

1 Introduction

D-Wave's hybrid solver service (HSS) contains a portfolio of heuristic solvers that leverage both quantum and classical solution approaches to solve optimization problems much larger than can fit on Advantage™ quantum processors. The quantum processing unit (QPU) natively solves quadratic unconstrained binary optimization problems over the Pegasus graph topology [1], while the portfolio of HSS solvers provide interface support for applications well outside that native problem formulation. This interface reduces, and sometimes completely eliminates, the need for users to translate their application problems into a formulation that matches the quantum architecture.

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 Binary Quadratic Model (BQM) and Discrete Quadratic Model (DQM) solvers read unconstrained quadratic problems defined on binary variables (that is, taking two values), and on discrete variables (taking multiple values), respectively. The Constrained Quadratic Model (CQM) Solver adds the capability of specifying linear and quadratic constraints for the quadratic model. Moreover, this solver accepts problems defined on binary, integer and, as of May 2022, real variables.[1] 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.

In this report we focus on understanding the performance of the CQM solver on a wide variety of constrained quadratic problems. As D-Wave continues to update the CQM solver, including algorithmic improvements and increasing support for more problem types, the benchmarking framework used in this report can be used to quantify the impact of these changes.

This report presents an overview of performance of the Constrained Quadratic Model solver in the following sections:

- Section 2 surveys the varieties of problem types that serve as industry-standard benchmarks.

[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.

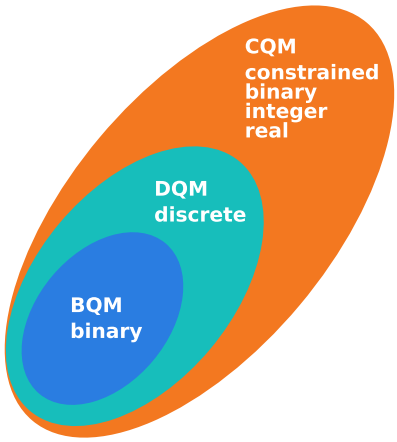

Figure 1: The hierarchy of models available in Ocean. The Binary Quadratic Model and Discrete Quadratic Model are subsets of the more general Constrained Quadratic Model. Figure originally appeared in [2].

- Section 3 presents details about the releases of the CQM solver which are compared in this benchmark.
- Section 4 presents the results of the benchmark.

The hybrid solver service is cloud-based and offered by subscription via the Leap™ web portal; see [3, 4] to learn more about Leap and the hybrid solver service.

2 Problem Classes for Understanding Performance

Constrained quadratic models are a large class of models which can contain binary, discrete, integer, and real variables. The most general version of the CQM formulation can include interactions between two variables of any kind. In addition to specifying an objective function to be optimized, the model can include several kinds of constraints which must be satisfied. These constraints themselves take the form of quadratic models. The CQM model supports formulation of equality and inequality constraints, which may be linear or quadratic as well as hard (weighted) or soft (weighted).

Constraints can be defined on binary, integer, or continuous variables.[2] Samples returned by the CQM solver are called *feasible* if they satisfy the constraints provided; otherwise, they are called *infeasible*.[3]

Understanding performance on all CQMs can be difficult because this problem class is very diverse. Instead, we can break it up into a menagerie of smaller interrelated problem classes that resemble specific applications types. We can use a rough hierarchy of generality to categorize these kinds of problems, as illustrated in Figure 2.

Binary Problems Binary quadratic problems are often used in feature selection, satisfying boolean expressions, and quantum simulation. Binary quadratic problems are closest to the native model supported by the quantum computer. In addition, a CQM with binary variables can be used to represent a model with discrete variables. Given a discrete variable x, which takes values in $\{D_1, \ldots, D_n\}$, we can encode the state of x as a set of n booleans d_i which are equal to 1 if $x = D_i$. This is not sufficient, as x can only take one state at a time, and so we have to add the constraint that $\sum D_i = 1$. Therefore, we also use problems with discrete variables, such as graph coloring, and give them binary-only formulations.

Integer Problems Integer problems, for which variables can be assigned multiple ordered values like $0, 1, 2, 3, \ldots$ are widely used in discrete optimization. Here we use "integer problem" to include problems defined on integer and/or binary variables, but not on real (continuous) variables. Many problems are natively integer, for example, financial problems that involve purchasing whole units. Integer problems are often also useful for problems with discrete units of time or space, such as in scheduling problems where events are scheduled in 15-minute intervals. These problems are naturally integer and not discrete because time and space are ordered, whereas discrete variables encoded as binary are unordered. Often formulations for problems, like bin packing, can be done using either integer or real variables; however, the choice of which formulation is more appropriate is determined by the specifics of a given application. The integer versions of these problems are created by discretizing the continuous variables, like time or space.

Mixed Integer Problems Mixed integer problems, which can contain a combination of binary, integer, and real variables, are the most general types used in our tests. Models containing real variables are typically found when the values to be assigned to nodes represent variables that are naturally continuous, like locations in space, time, and money (when allowing for subdividing dollars).

Sourcing We source many of these problems from several discrete programming libraries [6–10], which often feature a variety of variable types, degree, and sizes of problems. Some libraries, like MINLPLib [6, 7] have a variety of application-specific problems which have been collected into benchmarks. Other libraries are generated based on a single application type, such as graph coloring [10]. All instances were sourced as `lp` or `mps` files and then converted into a solver-compatible format using the Ocean SDK [4].

We have also developed input generators for satisfiability and circuit satisfiability (used for factoring) inputs, which are mainly formulated as binary quadratic inputs. One benefit of writing problem generators is the ability to test performance over a variety of input parameters and formulation strategies, including performance on equivalent formulations. For example we can construct a boolean satisfiability problem $(x_1 \wedge x_2)$ as an unconstrained CQM $Obj = -x_1 \cdot x_2$ or as a constrained cqm with $Obj = 0$ and the constraint $x_1 \cdot x_2 = 1$. We can do the same procedure for factoring problems expressed as multiplication circuits, since these are a kind of satisfiability problem.

To achieve a similar diversity of equivalent formulations for problems drawn from benchmarking libraries, we create equivalent formulations by randomly applying a "flip" to a constraint $x \geq y \Rightarrow -x \leq -y$, or scaling constraints by random positive numbers $x \geq y \Rightarrow a \cdot x \geq a \cdot y, a > 0$. These augmentations to existing li-

[2] Not every possible formulation has historically been supported by the solver, with real variables being introduced in May of 2022. Moreover, the solver does not support CQMs where there is a real variable in a quadratic interaction. Similarly, problems involving soft constraints can only be solved in versions of the solver introduced as of November of 2022.

[3] Because the CQM solver is a heuristic solver, it may return a mixture of feasible and infeasible samples, none of which are guaranteed to be optimal.

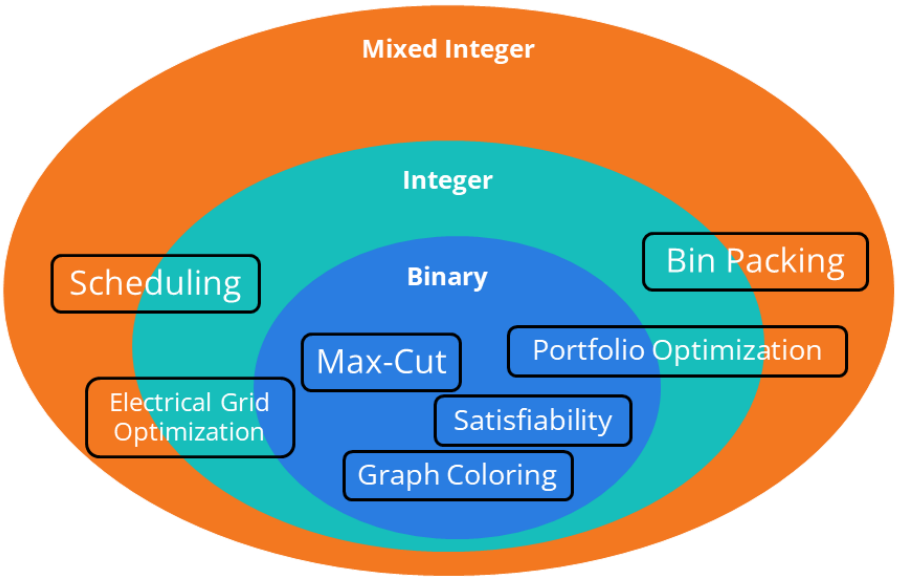

Figure 2: For the various problem classes that were tested, a non-exhaustive set of example application areas as they would be formulated in the CQM solver. Often problems can be formulated several ways and the details of the formulation are application and performance specific. To see some of these examples in practice visit [5].

braries help provide an even more diverse set of problems with which to characterize the performance of the solver.

3 Updates to the CQM Solver

To characterize the impact of updates to the CQM solver available on Leap, we choose particular updates to use in our tests.[4]

1. As of December 2021: This release includes a few minor updates from the initial October 2021 introduction of the CQM solver. This version of the CQM solver accepts problems with binary and integer variables and one or more constraints. In contrast to earlier releases of the BQM and DQM solvers, which require the user to formulate constraints as penalty terms in the objective function, this version of the CQM solver supports direct expression of equality and inequality constraints on variables.
2. As of May 2022: This release enables the CQM solver to expand the accepted models to include CQMs with real-valued variables (also known as "continuous variables").
3. As of November 2022: This release enables the CQM solvers to accept CQMs with weighted (i.e. "soft") constraints. A constraint can be either "hard" or "soft." Soft constraints are weighted in relative importance to the objective, other constraints, or both, and thus can be violated to achieve an overall good solution; whereas hard constraints must be satisfied. Previously, all constraints were hard.

In addition to these modeling features, each release includes performance updates. Because the CQM solver continues to evolve, certain problem classes can be solved in newer updates that were previously impossible. Figure 3 characterizes the ability of the solver to read certain input categories for each update.

[4] To better understand where these updates to the solver exist in the product timeline, see [**ReleaseNotesDWave**].

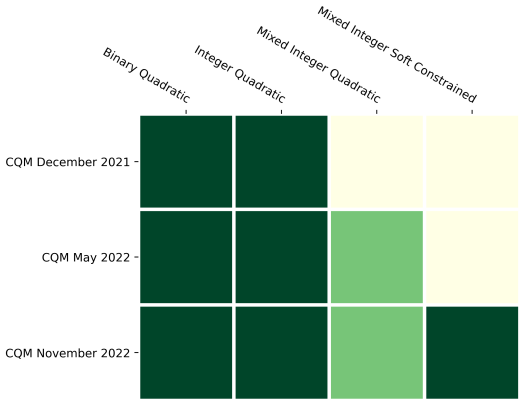

Figure 3: The taxonomy of problems being tested, alongside the ability of particular updates of the CQM solver to take in particular problem classes. Dark green indicates the given update can solve all problems of that class; light green indicates the CQM solver can solve problems of that class with some exceptions; and yellow indicates incompatibility with that update of the CQM solver.

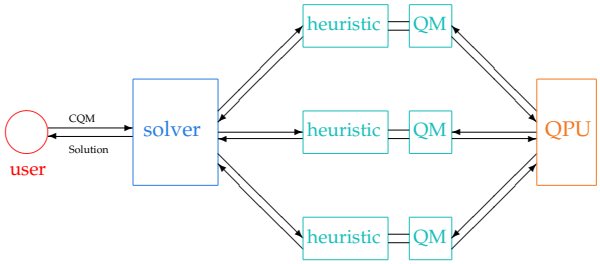

Figure 4: Structure of the CQM solver in HSS. The front end (blue) reads an input Q, and, optionally, a time limit T. The CQM solver invokes some number of heuristic solvers (threads) that run on classical CPUs and GPUs (teal) and search for good-quality solutions to Q. Each heuristic solver contains 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. This figure originally appeared in [2].

4 Performance of the CQM Solver

Every release of the CQM solver is based on the same hybrid quantum-classical workflow, as shown in Figure 4. The solver has a classical front end that reads an input Q and (optionally) a time limit T.[5] It then invokes one or more hybrid heuristic solvers (computation threads) to search for good-quality solutions to Q.

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 backend Advantage QPU. Responses 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.

In a production environment, heuristic solvers run in parallel on state-of-the-art CPU and/or GPU platforms. These tests were carried out using a "laboratory" version of the CQM solver for each update tested. These versions run on less performant classical hardware, but allow us to maintain older updates which are no longer available in production, and to carefully control the hardware for comparisons. In contrast, the HSS production solvers available to the public are deployed for scalable use in the cloud. Since the hybrid framework shown in Figure 4 is heavily dependent on the performance and scale of hardware, the results of this section may differ somewhat from those observed in deployed systems, though we generally expect the latter to be more efficient.

4.1 Methodology

Each solver was containerized and run with the contemporaneous Ocean software releases where possible, with small fixes to allow older updates of the solver to run. The solvers were all given 300 seconds (5 minutes) to solve each problem using the same hardware resources and access to the QPU. Each solver returned an algorithmically determined number of samples, and

[5] If no time limit is provided by the user, a default time that depends on input size is used.

the objective value and feasibility of each sample was calculated.

In this section we compare the different updates of the solver using a win-loss criterion. For each problem, the solver which has the lowest feasible objective value on any sample is given a "win", with ties being counted as a win for both solvers. If none of the solvers have a feasible objective value, meaning all of the samples for that problem across all solvers were infeasible, then the solver with the lowest overall objective value is counted as a win. This approach aligns with "solution to time framework" adopted by several publicly available repositories of benchmarks for quadratic solvers [e.g. 11].

4.2 Results

Figure 5 summarizes the results of running this experiment on the different updates of the CQM solver under the parameters outlined above.

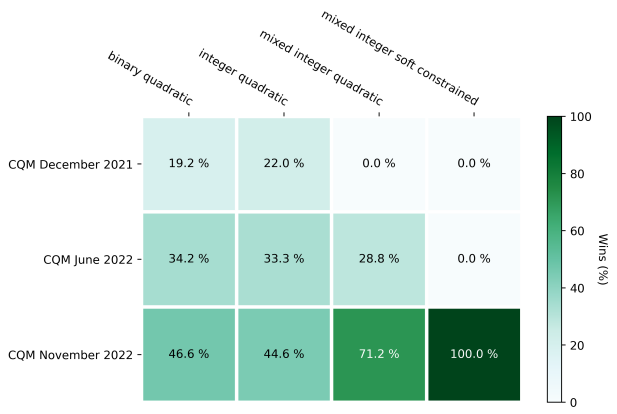

Figure 5: The percentage of "wins" for each update of the solver, allowing for ties to be counted for both solvers. A win is characterized by a lower feasible objective value on the same problem instance given the same hardware and time limit. If no feasible objective value is found, then the win is given to the solver with the lowest overall objective value.

Of the 657 binary quadratic problems, the November 2022 release won 46.6% of problems. The second and third best solver on binary quadratic problems were May 2022 (34.2%) and December 2021 (19.2%) respectively. Of the 708 integer quadratic problems, the November 2022 release won 44.6% of problems. The second and third best solver on integer quadratic problems were May 2022 (33.3%) and December 2021 (22.0%) respectively. Of the 735 mixed integer quadratic problems, the November 2022 release won 71.2% of problems. The second best solver on mixed integer quadratic problems was May 2022 (28.8%), and no other solver won or was able to return solutions. Of the 70 mixed integer soft constrained problems, the November 2022 release won 100.0% of problems. No other solver was able to win or return solutions.

5 Conclusion

To understand the performance of algorithms at the rapidly advancing frontier of hybrid optimization, it is necessary to benchmark on a diverse set of problems. This report puts forward a taxonomy of problems which represent various real-world and theoretical benchmarks for D-Wave's Constrained Quadratic Model solver. Using this framework, we characterize the overall performance of the most recent release of this solver against several previous updates to the solver.

References

1 C. McGeoch and P. Farré, "Advantage Processor Overview," D-Wave Technical Report Series (2022).

2 "Hybrid Solvers for Quadratic Optimization," D-Wave Whitepaper Series (2022).

3 *D-Wave Leap*, `https://cloud.dwavesys.com/leap`.

4 *D-Wave Ocean Software Documentation*, `https://docs.ocean.dwavesys.com/`.

5 *D-Wave Systems Examples*, GitHub, `https://github.com/dwave-examples`.

6 M. R. Bussieck, A. S. Drud, and A. Meeraus, "MINLPLib—A Collection of Test Models for Mixed-Integer Nonlinear Programming," INFORMS Journal on Computing **15**, 114–119 (2003).

7 S. Vigerske, *MINLPLib: A Library of Mixed-Integer and Continuous Nonlinear Programming Instances*, (Oct. 14, 2022) `https://www.minlplib.org/` (visited on 10/22/2022).

8 A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, et al., "MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library," Mathematical Pro-

gramming Computation, 10.1007/s12532-020-00194-3 (2021).

[9] F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, et al., "QPLIB: a library of quadratic programming instances," Math. Prog. Comp. **11**, 237–265 (2019).

[10] S. Gualandi and M. Chiarandini, *Vertex Coloring - Graph Coloring Benchmarks*, https://sites.google.com/site/graphcoloring/vertex-coloring (visited on 10/22/2022).

[11] H. D. Mittelmann, *Decison Tree for Optimization Software*, http://plato.asu.edu/guide.html (visited on 10/22/2022).