Embedding Inequality Constraints for Quantum Annealing Optimization

Tomáš Vyskočil, Scott Pakin, and Hristo N. Djidjev

Los Alamos National Laboratory

Abstract. Quantum annealing is a model for quantum computing that is aimed at solving hard optimization problems by representing them as quadratic unconstrained binary optimization (QUBO) problems. Although many NP-hard problems can easily be formulated as binary-variable problems with a quadratic objective function, such formulations typically also include constraints, which are not allowed in a QUBO. Hence, such constraints are usually incorporated in the objective function as additive penalty terms. While there is substantial previous work on implementing linear equality constraints, the case of inequality constraints has not much been studied. In this paper, we propose a new approach for formulating and embedding inequality constraints as penalties and describe early implementation results.

1 Introduction

Quantum annealing is a hardware analogue of simulated annealing [11], a well-known black-box approach for solving optimization problems. (By "black-box" we mean that no assumptions, e.g., differentiability, can be made of the function to optimize.) However, quantum annealing takes advantage of quantum effects—in particular *quantum tunneling*—to converge with higher probability than simulated annealing given the same annealing schedule [10]. Quantum annealers have been designed to look for solutions of a specific type of binary optimization problem as discussed next.

1.1 Quadratic unconstrained binary optimization

Current quantum annealers, such as the D-Wave 2000Q [8], minimize only quadratic pseudo-Boolean functions. That is, they heuristically solve a *quadratic unconstrained binary optimization* problem (QUBO),

$$\arg\min_{x} \overbrace{\left(\sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{i,j} x_i x_j\right)}^{\mathrm{Obj}(\boldsymbol{x})} \tag{1}$$

for $\boldsymbol{x} \in \mathbb{B}^n$ (where $\mathbb{B}$ indicates the set $\{0,1\}$) given $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^{n \times n}$. We write "heuristically solve" because, just like simulated annealing, global optimality

is not guaranteed; a quantum annealer takes only a best-effort approach. Finding the true global minimum is an NP-hard problem [1] so any hardware that can accelerate solutions—even if only by a polynomial amount—can be useful in practice for reducing time to solution.

Although Equation (1) may seem restrictive in expressiveness, there are in fact a large set of problems with a known mapping into that form [13]. A particular challenge in expressing an optimization problem as a QUBO is that a QUBO by definition does not allow constraints. However, a constraint can be expressed in terms of *penalties*, $a_i x_i$ and $b_{i,j} x_i x_j$ terms that can be included in Equation (1) and that evaluate to zero when the constraint is honored and to a positive number when the constraint is violated. For example, one can constrain $x_1 = x_2$ by adding a penalty term of $x_1 + x_2 - 2x_1 x_2$ to the QUBO. In this case, $\text{Obj}(0,0) = \text{Obj}(1,1) = 0$, but $\text{Obj}(0,1) = \text{Obj}(1,0) = 1$. A quantum annealer would therefore favor the $x_1 = x_2$ cases when computing $\arg\min_x \text{Obj}(\boldsymbol{x})$. Kochenberger et al. list a handful of other such penalties in their survey of QUBO problems [12].

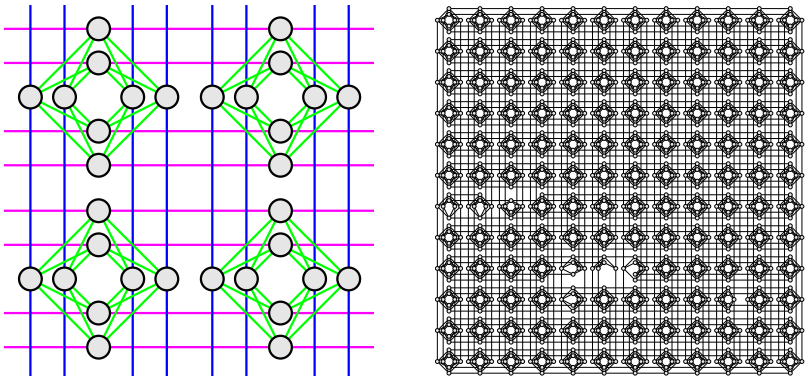
1.2 Physical limitations

Equation (1) describes a *logical* optimization problem. In practice, quantum-annealing hardware imposes a number of additional limitations on what can be expressed. Some of the key differences between the logical and physical problems are as follows:

- The number of variables (n) is limited to the number of qubits (quantum bits) provided by the quantum processing unit (QPU). In the case of a D-Wave 2000Q, $n \leq 2048$.
- The $\boldsymbol{a}$ and $\boldsymbol{b}$ coefficients have neither infinite range nor infinite precision. In the case of a D-Wave 2000Q, each $a_i \in [-2.0, 2.0]$, and each $b_{i,j} \in [-1.0, 1.0]$, with approximately 5–6 bits of precision.
- Only a small subset of the $b_{i,j}$ coefficients are allowed to be nonzero. In the case of a D-Wave 2000Q, there can be at most six nonzero $b_{i,j}$ coefficients for a given i due to the hardware's physical topology.

To elaborate on that final difference, a D-Wave 2000Q employs a particular physical topology called a *Chimera graph* [4]. A nonzero a_i coefficient can appear at any vertex in the graph, but a nonzero $b_{i,j}$ coefficient can appear only where an edge is present in the graph. Figure 1 illustrates a Chimera graph. A Chimera graph is composed of complete, 8-vertex bipartite graphs (i.e., $K_{4,4}$) called *unit cells* (Figure 1(a)). Each vertex in the first partition connects to its peer to the north and its peer to the south. Each vertex in the second partition connects to its peer to the east and its peer to the west. Consequently, a Chimera graph is, excluding vertices on the boundaries, a degree-six graph, which is quite sparse.

Due to manufacturing and calibration imperfections, some vertices and edges are inevitably absent from the hardware's Chimera graph. For instance, Figure 1(b) presents the physical topology of Ising, a D-Wave system installed at



(a) Blow-up of a 2×2 grid of unit cells

(b) Chimera graph implemented by LANL's D-Wave system, Ising

Fig. 1. Illustration of the Chimera-graph topology

Los Alamos National Laboratory (LANL). Ising is missing 11 vertices and 62 edges relative to a complete C12 Chimera graph. (*C12* denotes a 12×12 grid of unit cells.)

1.3 Problem embedding

We revisit the notion of penalty terms introduced in Section 1.1 in light of the hardware limitations described in Section 1.2. Consider implementing as a QUBO the constraint $(\sum_{i=1}^{n} x_i) = k$ for some $0 \leq k \leq n$ and with $x_i \in \mathbb{B}$. One would typically accomplish this by adding a "squared error" expression,

$$M\left((\textstyle\sum_{i=1}^{n} x_i) - k\right)^2 \tag{2}$$

to the quadratic pseudo-Boolean function to be minimized, for a sufficiently large M. For example, constraining exactly two of three Boolean variables to be 1 implies setting $n = 3$ and $k = 2$ and deriving $\mathrm{Obj}(\boldsymbol{x})$ as

$$\begin{aligned} M\left((\textstyle\sum_{i=1}^{3} x_i) - 2\right)^2 &= M(x_1^2 + 2x_1x_2 + 2x_1x_3 - 4x_1 \\ &\quad + x_2^2 + 2x_2x_3 - 4x_2 + x_3^2 - 4x_3 + 4) \\ &= M(-3x_1 - 3x_2 - 3x_3 + 2x_1x_2 + 2x_1x_3 + 2x_2x_3 + 4), \end{aligned}$$

recalling that $x_i^2 = x_i$ when $x_i \in \mathbb{B}$. Furthermore, the constant 4 can be removed from the penalty because constant terms do not affect the solution to Equation (1).

While the "squared error" approach is generally applicable, it suffers from a severe shortcoming in practice: For n variables, it requires n^2 quadratic terms. In a graph representation, this implies all-to-all connectivity—a feature lacking in contemporary hardware. A workaround is to *minor-embed* [5] the problem

graph into the hardware graph. This implies replacing individual variables with multiple variables and *chaining* them together—constraining them to have the same value using the $x_1 = x_2$ approach presented in Section 1.1. For a complete graph, minor embedding incurs a substantial (quadratic) cost in the variable count. Specifically, a C_m Chimera graph can represent problems of no more than $4m + 1$ variables if all $(4m + 1)^2$ quadratic terms are nonzero [3]. For example, even with 1141 vertices, the C12 subgraph shown in Figure 1(b) is limited to $n = 49$ when applying the Equation (2) constraint.

1.4 Our contribution

In this paper we introduce a novel approach to represent inequality constraints of the form $\sum_{i=1}^{n} x_i \leq k$ using roughly kn variables. Our approach is based on a set of "gadgets" that map directly to a unit cell and that spatially tile the Chimera graph in a manner that enforces the desired constraint.

The rest of the paper is structured as follows. In Section 2 we discuss related efforts to map constraints to QUBOs. Section 3 explains the basics of our approach. The methods used to implement our approach are detailed in Section 4. We present in Section 5 some empirical analysis of the efficacy of our techniques on actual quantum-annealing hardware. Finally, we draw some conclusions from our work in Section 6.

2 Related Work

Much work has been done in mapping individual computational problems to QUBOs. Kochenberger et al. survey a set of these dating back to the early 1970s [12]. (Note that Kochenberger et al. use the term *unconstrained binary quadratic programming* or *UBQP* instead of QUBO. The two are synonymous.) Baharona [1] describes how a large set of problems can be mapped to the form of an Ising-model Hamiltonian function, which is structurally identical to Equation (1), but solves for $\boldsymbol{x} \in \{-1, +1\}^N$ rather than $\boldsymbol{x} \in \{0, 1\}^N$. A simple linear transformation converts between the two.

Perhaps the most closely related work to what we are proposing is Bian et al.'s investigation into mapping constraint satisfaction problems (CSPs) to a D-Wave system while carefully considering the embedding on the Chimera graph [2]. One difference is our focus on scalability that allows constraints of any number of variables to be embedded and that we are able to handle inequalities. We have also applied an approach similar to the one described in Sections 3 and 4 to equality constraints of the type $\sum_{i=1}^{n} x_i = 1$ [15] and later generalized this approach [14].

3 Optimization based approach

In this section we define the problem we intend to solve and present an outline of our approach.

3.1 Formulating a constraint embedding as an optimization problem

We first overview the properties of the standard penalty-based approach and then the approach adopted in this paper that uses optimization and extra binary variables.

Implementation of a constraint as a penalty Although most NP-hard optimization problems admit simple formulations as optimization problems with quadratic binary objective functions, such formulations typically contain at least one constraint and are therefore not in the QUBO form represented by Equation (1). In order to convert optimization problems containing constraints into QUBOs, each constraint is usually converted to a penalty that is included in the objective function as an additive term. We call $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ *feasible* if it satisfies the constraint, and *infeasible*, otherwise.

Given a minimization problem with objective $\mathrm{Obj}(\boldsymbol{x})$ and an equality constraint $C(\boldsymbol{x}) = 0$, the penalty method transforms the constraint into the quadratic term $Q(\boldsymbol{x}) = (C(\boldsymbol{x}))^2$ and changes the objective to $\mathrm{Obj}'(\boldsymbol{x}) = \mathrm{Obj}(\boldsymbol{x}) + MQ(\boldsymbol{x})$ for a large positive constant M. When $\boldsymbol{x}$ is feasible, i.e., if $C(\boldsymbol{x}) = 0$, then $Q(\boldsymbol{x}) = 0$ and the penalty term does not change the value of the objective, i.e., $\mathrm{Obj}'(\boldsymbol{x}) = \mathrm{Obj}(\boldsymbol{x})$. But if $\boldsymbol{x}$ is infeasible, then $Q(\boldsymbol{x}) \geq \gamma$ and $\mathrm{Obj}'(\boldsymbol{x}) \geq \mathrm{Obj}(\boldsymbol{x}) + M\gamma$, where γ is the minimum nonzero value of $Q(\boldsymbol{x})$. Hence, for the penalty method, the following properties of the penalty function are essential:

(i) If $\boldsymbol{x}$ is feasible, then $Q(\boldsymbol{x}) = 0$;
(ii) if $\boldsymbol{x}$ is infeasible, then $Q(\boldsymbol{x}) \geq \gamma$,

and the parameter γ, called *gap*, should be as large as possible. Having a large gap is important because it enables using smaller M values, making the resulting optimization problem more stable numerically and thereby increasing the accuracy of the solution.

As discussed in the introduction, implementing a linear constraint as a penalty comes with significant drawbacks. But implementing a linear *inequality* constraint such as

$$\sum_{i=1}^{n} x_i \leq k \tag{3}$$

is even harder because the squared-error approach does not apply. One way to deal with such inequalities is to introduce a new variable $z \geq 0$ such that $(\sum_{i=1}^{n} x_i) + z = k$ and then to apply to the resulting equality constraint the penalty method described above. However, z does not necessarily take binary values in a feasible solution and can be as large as k, so it needs to be represented in a binary form by introducing $\lceil \log k \rceil$ binary variables with possibly large ($\approx k$) QUBO coefficients, further complicating the resulting QUBO. Hence, we are looking in this paper at the problem of finding a scalable and qubit-efficient way for embedding inequality-type constraints in quantum annealers such as those by D-Wave Systems.

Using ancillary variables One approach, previously examined by multiple researchers [2, 15], is to include m ancillary variables t_i and use the additional degrees of freedom to establish a larger "gap" between feasible and infeasible solutions and/or reduce the number of quadratic coefficients needed. Formally, let $\boldsymbol{t} = \{t_1, \ldots, t_m\}$. We want to define a QUBO $Q(\boldsymbol{x}, \boldsymbol{t})$ that satisfies the following analogues of properties (i) and (ii) above:

(i′) If $\boldsymbol{x}$ is feasible, then $\min_{\boldsymbol{t}} Q(\boldsymbol{x}, \boldsymbol{t}) = 0$;
(ii′) if $\boldsymbol{x}$ is infeasible, then $\min_{\boldsymbol{t}} Q(\boldsymbol{x}, \boldsymbol{t}) \geq \gamma$,

and that also maximizes γ.

Although these conditions look very similar, (ii′) can be implemented as a linear programming constraint with respect to the real variables a_i and $b_{i,j}$ from Equation (1), while (i′) is not linear and implementing it requires, e.g., the introduction of additional binary variables, resulting in a much more difficult problem. Moreover, the total number of constraints is 2^{n+m} as there is a constraint for each $\boldsymbol{x} \in \mathbb{B}^n$ and each $\boldsymbol{t} \in \mathbb{B}^m$. The resulting mixed-integer programming (MIP) problem with exponential number of constraints cannot be solved in reasonable time for more than 3–4 unit cells of the Chimera graph (i.e., $n + m$ is no more than 30–40). Clearly, this approach cannot be applied to current D-Wave systems with up to 2048 qubits and especially not to future generations with much higher numbers of qubits.

3.2 Our two-level approach

The main idea of our proposed method is to develop a scalable and modular design by using a *two-level* approach that replaces solving an optimization problem for the entire Chimera graph with solving several similar types of optimization problems, each limited to a single unit cell of the Chimera graph.

Specifically, on the higher level, we design specifications for several types of cell behavior, determined by the properties of the QUBOs defined in these cells, which we call *gadgets*. If the gadgets are arranged in a specific pattern to cover the entire Chimera graph or portions of it, then they produce a QUBO that implements the constraint of interest.

On the lower level, we show that gadgets with the specified properties can actually be designed by solving a corresponding optimization problem. In the next section, we provide details on the implementation of that approach.

4 Implementation

4.1 Gadgets

As mentioned above, gadgets are QUBO quadratic forms defined on a single Chimera-graph cell. We will arrange these gadgets to cover a rectangular portion R of the Chimera graph. We define three types of gadgets depending on where the corresponding cells will be positioned in R:

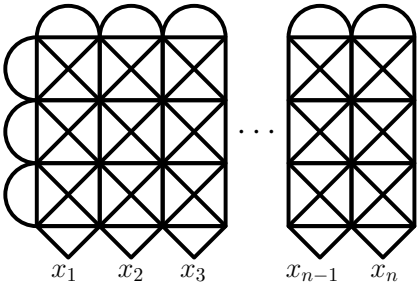


Fig. 2. Arranging the gadgets to solve constraint Equation (3) in a $k \times n$ region R.

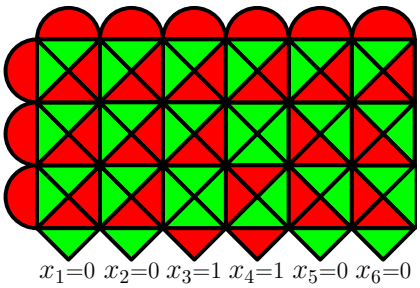


Fig. 3. Optimal tiling for the inequality $\sum_{i=1}^{6} x_i \leq 3$.

- *Internal gadget*, denoted by ⊠, for cells (mostly) in the interior of R;
- *Problem gadget*, denoted by ▽, for cells on the bottom boundary of R, each of which represents a problem variable from $\boldsymbol{x}$; and
- *Boundary gadget*, for cells on the left and top boundaries of region R. Depending on the orientation of this gadget, it will be denoted by either ◖ or ◡.

4.2 Arranging the gadgets to cover the Chimera graph

As illustrated by Figure 2, the gadgets cover a rectangular region of the Chimera graph, with problem gadgets on the bottom, boundary gadgets on the top and left boundaries, and internal gadgets on the rest. Moreover, problem variables $\boldsymbol{x}$ are positioned in the problem gadgets, one per gadget. We refer to this QUBO formulation as $Qb(R)$. In the next subsections we will show that $Qb(R)$ is a correct implementation of the constraint specified by Equation (3) and estimate the gap.

4.3 Tiles

In this section we define *tiles*, which are used to easily identify when an assignment to the variables of QUBO problems of a specific kind is an optimal one, merely by examining their types and colors. Specifically, if all the tiles are of "good" type (as defined below) and the adjacent tiles' neighboring sides are colored with the opposite color, then the assignment is an optimal one. Next, we give a more formal description and analysis.

While a gadget refers to the QUBO formulation of a cell, i.e., to the values given to the coefficients $\boldsymbol{a}$ and $\boldsymbol{b}$ in the cell, a *tile* is defined both by the underlining gadget as well as by the values of the cell's variables, i.e., by $\boldsymbol{x}$ and $\boldsymbol{t}$. The combination of the tile types of all the cells of R is called a *tiling* of R. In order to estimate the value of a given variable assignment, i.e., a tiling of R, we categorize the tiles as *good*, meaning that the value α of the cell's QUBO

is the smallest possible for any cell, and *bad*, meaning that the corresponding QUBO value exceeds that minimum by at least a parameter $\gamma > 0$, which we call a *gap*. There are 8 types of good tiles illustrated in Figure 4. The colors indicate values of the interface variables, i.e., variables that are connected by an active (i.e., with nonzero bias) coupler to a variable in a neighboring cell. (In fact, in order to maximize the gap, such couplers are given the value -1, the smallest value allowed by the hardware.) Specifically, red corresponds to an interface variable value of 1, and green corresponds to 0. Crucially, the colors are used to characterize the interactions between neighboring tiles as described next.

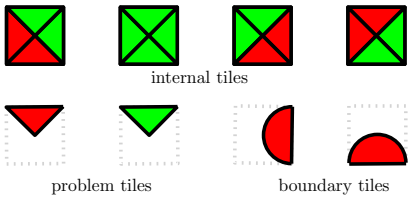


Fig. 4. The set of good tiles.

Similarly to the good and bad tile types, we define *good* interactions when the colors on both sides of the common border are different, and *bad*, if they are the same. If the interaction is good, then the value of the quadratic term corresponding to the active coupler connecting the tiles will be negative, generating a reward (i.e., reducing the value of the QUBO, which is beneficial for a minimization problem), and if it is bad, then that value is positive, generating a penalty (increasing the value of the QUBO). We require that the difference between a reward and the smallest penalty for good/bad interactions be γ, the same as the gap between good and bad tiles.

The tiling of R is called good, if all the tiles and tile interactions are good. The main property of the construction defined in Section 4.2 is that there exists a good tiling of R if and only if the values of the problem variables $\boldsymbol{x}$ satisfy the constraint Equation (3). From this property, it follows that if the constraint is not satisfied, then there is at least one bad tile or connection, and therefore, the gap for $Qb(R)$ between variables satisfying and not satisfying Equation (3) is γ.

We will illustrate why this property holds by an example, the formal proof is similar to the one given in our prior report [14].

Figure 3 shows a good tiling for the constraint $\sum_{i=1}^{6} x_i \leq 3$ implemented using gadgets as illustrated in Figure 2. Note that all boundary tiles have to be red in a good tiling, since this is the only tile type available for a boundary gadget. Furthermore, the colors of the problem tiles encode the variables $\boldsymbol{x}$; in our case,  for $x_1 = x_2 = x_5 = x_6 = 0$ and  for $x_3 = x_4 = 1$. We show that the types of the remaining (internal) tiles are uniquely determined in a good tiling. Notice that each internal tile consists of four triangles, and we are looking at the color of the top one. The main property is that that the number of red colored top triangles decreases by one when we go row by row up from the bottom to the top, until reaching zero. Specifically, the first row from the bottom, consisting of problem tiles, has two red triangles , on position 3 and 4, corresponding to the two $+1$ $\boldsymbol{x}$ values. The second row has only one good way to be tiled: going from left to right, we can only tile it by a sequence of tiles, followed by one tile

above the leftmost problem red tile, one tile above the other problem red tile, and ending with a sequence of tiles. As a result, that row has on its top boundary one fewer red top-triangle than the previous one, since the x_3 and x_4 tiles were replaced by one (green top) and one (red top).

Similarly, the remaining red top-triangle tile is eliminated by another in the next row. In general, if there are l red problem tiles in the bottom row, in the next i rows there are exactly $l - i$ red top-triangle tiles (all of type) for $1 \leq i \leq l$, assuming $l \leq k$.

The remaining $k - l$ rows can only be tiled by sequences of internal tiles with green top-triangles. (In our example $k - l = 1$, but it is clear that one can put any number of rows consisting of internal tiles on top of each other.) The red boundary tiles on the last row guarantee that $l \leq k$ in a good tiling, since the row below it should have all red-top tiles being eliminated.

4.4 Solving the optimization problems

In the previous section we showed that if tiles with properties as described (i.e. for each gadget, the QUBO values of good and bad tiles for this gadget have a gap γ) then the QUBO formulation described in Section 4.2 implements the constraint Equation (3) with gap γ. In order to show that such tiles do exist, for each gadget, we need to solve an optimization problem similar to Problem (i′)-(ii′), but restricted to a single cell. Despite being a mixed-integer programming (MIP) problem, it is of small size (16 binary variables and 16 constraints), and easily solvable by modern solvers (in a fraction of a second by Gurobi [9]).

5 Analysis

We implemented the tiling scheme described in Section 3 using Python and D-Wave Systems's Ocean libraries [6]. We now present an empirical evaluation of this implementation.

As our initial test, we evaluate the $\{k = 3, n = 8\}$ case (i.e., 3 of 8 bits set to 1), comparing measurements taken on Google/NASA/USRA's D-Wave 2000Q quantum annealer with those acquired from a classical simulated-annealing algorithm (implemented in Ocean). We begin with a comparison of correctness. Recall that all annealers are probabilistic and that correct output, corresponding to finding a global minimum, is not guaranteed.

Figure 5 plots a histogram of the results of 1,000,000 anneals. The horizontal axis corresponds to the count of variables that were measured as 1 when the anneal completed, and the vertical axis indicates the number of observations of that count. Ideally, there should be a single bar of height 1,000,000 at position 3 on the x axis. The runs performed using simulated annealing have the correct mode of 3, observed on 35% of the anneals. Disappointingly, the mode of the D-Wave measurements is 5, not 3. 42% of the anneals set five variables to 1, while only 6% set three variables to 1. However, the correctness improved when we

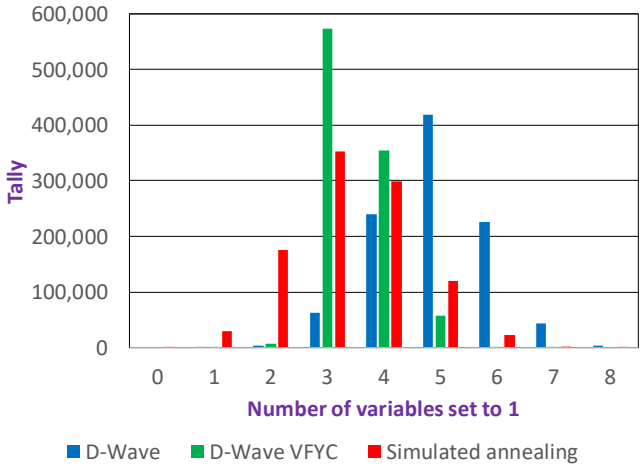


Fig. 5. Comparison of quantum annealing to simulated annealing for $\{k = 3, n = 8\}$

ran the D-Wave in virtual full-yield Chimera (VFYC) mode, which includes a classical post-processing step that nudges near-optimal solutions towards optimal solutions. This case performed better than both the raw D-Wave runs and the simulated-annealing runs, observing the correct mode and seeing 58% of the anneals setting three variables to 1. We do not yet know why the raw D-Wave performed so poorly on this test but are currently investigating the issue.

We now turn our attention to execution time. Table 1 lists the time in seconds required to perform 10,000 anneals, averaged over 10 trials. The annealing time for a D-Wave 2000Q is user-specified and ranges from 1µs to 2000µs. We retained the default annealing time of 20µs for our experiments. Simulated annealing was run on a workstation containing two 3.0 GHz Intel Broadwell processors (E5-2687WV4) and 64 GB of DRAM. As Table 1 indicates, the D-Wave is over five times faster than Ocean's implementation of simulated annealing running on a workstation. However, classical overheads—time to transfer the QUBO 1000 miles/1600 km (and back) over an SSL-encrypted network connection, time for the job scheduler to schedule the execution, time for the D-Wave to compute the analog waveforms corresponding to the digital QUBO, etc. [7]—dominate the time spent in quantum annealing. Specifically, at 10,000 anneals of 20µs apiece, only 0.2s of the 17.2s or 19.1s listed in the table (just over 1%) are spent in annealing proper.

Table 1. Execution time for 10,000 anneals, averaged over 10 trials

Annealer	Time (s)	(±%)
D-Wave	17.2 ± 1.7	(9.9%)
D-Wave VFYC	19.1 ± 2.6	(13.5%)
Simulated annealing	101.5 ± 1.1	(1.1%)

6 Conclusions

We developed an optimization approach for embedding penalty constraints of linear inequality type for quantum annealing and applied it to the Chimera graph topology used by the current D-Wave systems. Our approach enables inequality constraints of the type shown in Equation (3) to be implemented as a QUBO of $O(nk)$ variables. Such QUBOs can be embedded into a Chimera graph without chains (i.e., the QUBO structure is consistent with the Chimera-graph topology). The time for finding such an embedding is independent of the number of variables and depends only on the number of qubits in a cell. The experimental analysis shows that solving the resulting QUBO problem with a classical solver or with D-Wave plus classical postprocessing is accurate, but solutions with a purely quantum solver lack sufficient accuracy. In our future research, we plan to investigate the sources of such inaccuracy and will be looking for ways to increase the robustness.

We note that the construction from Figure 2 can also be used to solve interval constraints of the type $k_1 \leq \sum_{i=1}^{n} x_i \leq k_2$. To see that, notice that the problem variables are on the bottom boundary of the region. We can therefore create a new construction by merging the current one from Figure 2 with parameter k set to k_2 and a mirror construction with parameter k set to $n-k_1$ and reusing the same problem tiles. Clearly, the top half will implement the inequality $\sum_{i=1}^{n} x_i \leq k_2$, and we want the bottom one to implement the inequality $k_1 \leq \sum_{i=1}^{n} x_i$. To achieve that, we need to make the lower portion "count" the green ($x_i = -1$) tiles rather than the red ones, which can be done by a slight modification of the tiles in the bottom half (treating red color as green and vice versa). As a result, the bottom half will implement the inequality $\sum_{i=1}^{n}(1 - x_i) \leq n - k_2$, which is equivalent to $n - \sum_{i=1}^{n} x_i \leq n - k_2$ or $\sum_{i=1}^{n} x_i \geq k_2$.

Acknowledgments

Research presented in this article was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project numbers 20180267ER and 20190065DR. This work was also supported by the U.S. Department of Energy through Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy (contract no. 89233218CNA000001).

References

1. Francisco Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.
2. Zhengbing Bian, Fabian Chudak, Robert Israel, Brad Lackey, William G. Macready, and Aidan Roy. Discrete optimization using quantum annealing on sparse Ising models. *Frontiers in Physics*, 2:56:1–56:10, 2014.

3. Tomas Boothby, Andrew D. King, and Aidan Roy. Fast clique minor generation in Chimera qubit connectivity graphs. *Quantum Information Processing*, 15(1):495–508, January 2016.
4. Paul I. Bunyk, Emile M. Hoskinson, Mark W. Johnson, Elena Tolkacheva, Fabio Altomare, Andrew J. Berkley, Richard Harris, Jeremy P. Hilton, Trevor Lanting, Anthony J. Przybysz, and Jed Whittaker. Architectural considerations in the design of a superconducting quantum annealing processor. *IEEE Transactions on Applied Superconductivity*, 24(4):1–10, August 2014.
5. Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, October 2008.
6. D-Wave Systems, Inc. D-Wave's Ocean Software. URL: `https://ocean.dwavesys.com/`.
7. D-Wave Systems, Inc. Measuring computation time on D-Wave systems. User Manual 09-1107A-G, February 2, 2018.
8. Elizabeth Gibney. D-Wave upgrade: How scientists are using the world's most controversial quantum computer. *Nature*, 541(7638):447–448, January 26, 2017.
9. Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2015.
10. Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. *Physical Review E*, 58:5355–5363, November 1998.
11. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
12. Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: A survey. *Journal of Combinatorial Optimization*, 28(1):58–81, July 2014.
13. Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5:1–5:15, 2014.
14. Tomas Vyskocil and Hristo Djidjev. Optimization approach to constraint embedding for quantum annealers. Technical Report LA-UR-18-30971, Los Alamos National Laboratory, 2018.
15. Tomas Vyskocil and Hristo Djidjev. Simple constraint embedding for quantum annealers. In *International Conference on Rebooting Computing (to appear)*, 2018.