frontiers in PHYSICS

ORIGINAL RESEARCH ARTICLE
published: 18 September 2014
doi: 10.3389/fphy.2014.00056

Discrete optimization using quantum annealing on sparse Ising models

Zhengbing Bian[1], Fabian Chudak[1], Robert Israel[1], Brad Lackey[2], William G. Macready[1]* and Aidan Roy[1]

[1] *D-Wave Systems, Burnaby, BC, Canada*
[2] *Department of Computer Science, Joint Center for Quantum Information and Computer Science, University of Maryland, College Park, MD, USA*

Edited by:
Jacob Biamonte, ISI Foundation, Italy

Reviewed by:
Mauro Faccin, ISI Foundation, Italy
Maria Kieferova, Slovak Academy of Sciences, Slovakia
Andrew Lucas, Harvard University, USA
Travis S. Humble, Oak Ridge National Laboratory, USA

****Correspondence:***
William G. Macready, D-Wave Systems, 3033 Beta Ave, Burnaby, BC V5G 4M9, Canada
e-mail: wgm@dwavesys.com

This paper discusses techniques for solving discrete optimization problems using quantum annealing. Practical issues likely to affect the computation include precision limitations, finite temperature, bounded energy range, sparse connectivity, and small numbers of qubits. To address these concerns we propose a way of finding energy representations with large classical gaps between ground and first excited states, efficient algorithms for mapping non-compatible Ising models into the hardware, and the use of decomposition methods for problems that are too large to fit in hardware. We validate the approach by describing experiments with D-Wave quantum hardware for low density parity check decoding with up to 1000 variables.

Keywords: sparse Ising model, quantum annealing, discrete optimization, penalty functions

1. INTRODUCTION

D-Wave Systems manufactures a device [1–4] that uses quantum annealing (QA) to minimize the dimensionless energy of an Ising model

$$E(\boldsymbol{s}|\boldsymbol{h},\boldsymbol{J}) = \sum_{i\in V(\mathcal{G})} h_i s_i + \sum_{(i,j)\in E(\mathcal{G})} J_{i,j}\, s_i s_j. \qquad (1)$$

Here we have spin variables $s_i \in \{-1, 1\}$ indexed by the vertices $V(\mathcal{G})$ of a graph $\mathcal{G}$ fixed by the device with allowed pairwise interactions given by the edges $E(\mathcal{G})$ of this graph, and where the h_i and $J_{i,j}$ dimensionless coefficients are real-valued.

QA was proposed in Finilla et al. [5] and Kadowaki and Nishimori [6] as a method to optimize discrete energy functions. More recently, similar ideas were generalized to full quantum computation [7, 8]. Here we explore practical implementation of QA. The quantum annealing process minimizes the Ising energy by evolving the ground state of an initial Hamiltonian $\boldsymbol{H}_0 = \sum_{i\in V(\mathcal{G})} \boldsymbol{\sigma}_i^x$ to the ground state of a problem Hamiltonian $\boldsymbol{H}_P = \sum_{i\in V(\mathcal{G})} h_i\boldsymbol{\sigma}_i^z + \sum_{(i,j)\in E(\mathcal{G})} J_{i,j}\,\boldsymbol{\sigma}_i^z \otimes \boldsymbol{\sigma}_j^z$. The ground state of $\boldsymbol{H}_0$ is a superposition state in which all spin configurations are equally likely, while at the end of the process the spin configurations with smallest energy with respect to $\boldsymbol{H}_P$ are most likely to be measured. The efficiency of QA is determined in part by the energy gap separating ground and excited states during evolution. However, different representations of the same optimization problem may give different quantum gaps, and it is very difficult to know this gap in advance.

In this report, we focus not on problem representations giving rise to larger quantum gaps, but on representations ameliorating the limitations imposed by current experimental hardware. In particular, we observe that the following issues are detrimental for solving real world problems [9]:

1. *Limited precision/control error.* Physical devices impose limitations on the precision to which the programmable parameters $\boldsymbol{h}, \boldsymbol{J}$ can be specified. Moreover, since the Ising model is only an approximation to the underlying physics there may be systematic errors causing a discrepancy between programmed $\boldsymbol{h}, \boldsymbol{J}$ and the effective Ising approximation. To maximize the performance of QA in hardware we seek problem representations that are insensitive to these control errors.
2. *Limited energy range/finite temperature.* Technological limitations restrict the range of energy scales of $\boldsymbol{h}, \boldsymbol{J}$ relative to the thermal energy k_BT. For problems having few ground states and exponentially many excited states within k_BT a limited range makes the optimization challenging.
3. *Sparse connectivity.* It is difficult in hardware to realize all pairwise couplings as the number of couplings grows as n^2, for an n-qubit system. Thus, QA hardware offering large numbers of variables is likely to offer only sparse connectivity. **Figure 1** shows an example of the connectivity graph $\mathcal{G}$ for recent hardware supplied by D-Wave Systems. Optimization problems of practical interest require coupling significantly different from $\mathcal{G}$.
4. *Small numbers of qubits.* While Ising problems may be difficult to minimize even at scales of several hundred variables, real-world problems are often significantly larger. Moreover, the translation of any optimization problem into sparse Ising form will almost always require additional ancillary variables thereby increasing the size of the problem.

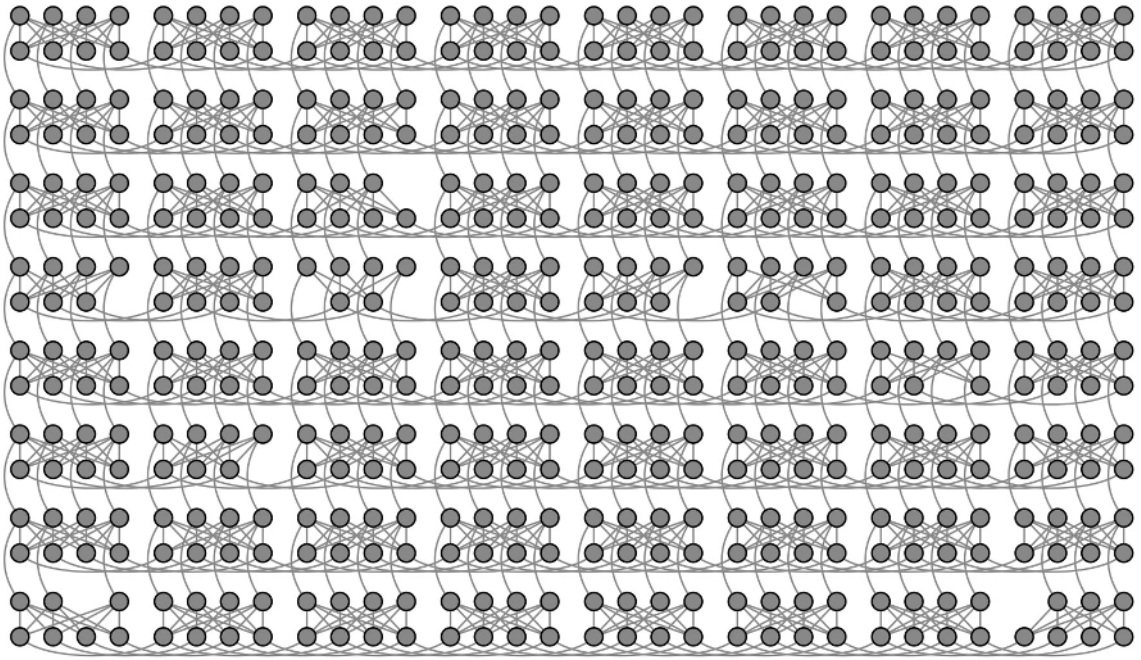


FIGURE 1 | D-Wave's recent generation Chimera connectivity graph $\mathcal{G}$. Vertices indicate spin-valued variables represented by programmable qubits (h_i's), and edges indicate programmable couplers ($J_{i,j}$'s). $\mathcal{G}$ is a lattice of $K_{4,4}$ unit cells where missing qubits are the result of fabrication defects.

The detrimental impact of hardware limitations (1) and (2) may be mitigated by finding problem representations in which there is a large (classical) gap between Ising ground and excited states. Notice that this does not guarantee that quantum gaps get any larger, though it seems unlikely they get worse. To deal with issue (3) we introduce additional variables to mediate arbitrary connectivity. For instance, ferromagnetically coupled spins can act as "wires" transporting long-range interactions; in general, the extra variables can take a more subtle role as ancillary variables. Finally, for issue (4), we demonstrate methods by which large problems may be solved through decomposition into smaller subproblems.

In addressing these practical limitations we focus on solving Constraint Satisfaction Problems (CSPs) involving a finite set of binary variables. That is, we have a set of variables $s_i \in \{-1, 1\}$, $i = 1 \ldots n$, and a set of constraints, each corresponding to a non-empty subset $\boldsymbol{F}_j$ of $\{-1, 1\}^n$ (constituting the configurations allowed by the constraint), and we wish to find $\boldsymbol{s} \in \bigcap_j \boldsymbol{F}_j$. To represent such a problem using an Ising model, we construct an Ising-model penalty function for each constraint $\boldsymbol{F}_j$. We write a penalty function as $Pen_{F_j}(\boldsymbol{z}) = Pen_{F_j}(\boldsymbol{s}, \boldsymbol{a})$ where $\boldsymbol{s}$ are the "decision" variables upon which $\boldsymbol{F}_j$ depends and $\boldsymbol{a}$ are ancillary variables. This function should be represented in the Ising form (1) (with an additional constant offset for convenience) and must satisfy:

$$\min_{\boldsymbol{a}} Pen_{F_j}(\boldsymbol{s}, \boldsymbol{a}) \begin{cases} = 0 & \text{if } \boldsymbol{s} \in \boldsymbol{F}_j \\ \geq g & \text{if } \boldsymbol{s} \notin \boldsymbol{F}_j \end{cases} \tag{2}$$

where $g > 0$. We call g the *gap* of the penalty function. While a decision variable may be involved in many different constraints, the sets of ancillary variables for different constraints are disjoint. In general, larger gaps make it more likely that the hardware will find a solution satisfying the constraints and offers protection against noise and precision limitations.

We add the penalty functions for each constraint to obtain an Ising model for our system of decision and ancillary variables. For any configuration of the decision variables that satisfies all the constraints, there will be some setting of the ancillary variables that makes all the penalty functions 0. On the other hand, any configuration of decision variables violating some constraint will have energy $\geq g$. Thus, a ground state of the system solves the CSP if a solution exists.

We also address Constrained Optimization Problems (COPs), where in addition to the constraints $\boldsymbol{F}_j$ there is an objective to be minimized over the feasible configurations. This can be accomplished by adding more terms to the Hamiltonian expressing the objective in Ising form. The objective should be scaled so that it does not overcome the penalty functions, causing the appearance of ground states that do not satisfy the constraints. In this case also, a larger gap g allows for better scaling of the objective.

This report shows how CSPs and COPs can be represented to ameliorate the hardware limitations listed above. The techniques we present are applicable to any hardware device offering only sparse pairwise variable interaction. In Section 2.1, we consider how to construct a penalty function for a given constraint with the largest possible gap, subject to bounds on the h's and J's imposed by the hardware. We supply a novel algorithm which exploits the sparsity of the hardware graph $\mathcal{G}$. In Section 2.2, we consider how to fit a collection of penalty functions (and the corresponding graphs) onto a hardware graph of sparse connectivity. Generally,

each variable participates in several constraints, necessitating the use of embedded chains of vertices of the hardware graph representing a single variable. We present a new heuristic embedding algorithm which scales well to large problems sizes. In Section 2.3, we consider how to deal with problems that are too large to be embedded in the hardware graph. We split the problem into subproblems small enough to embed, and coordinate solutions of the subproblems. We explore two coordination approaches: Belief Propagation (BP) and Dual Decomposition. Section 3 presents experimental results on using the D-Wave hardware to decode LDPC (low-density parity-check) codes [10, see Ch. 47]. Here, each constraint is a parity check on a small set of bits, and the objective is to minimize the Hamming distance to a received vector subject to satisfying all parity checks. Using the techniques presented in this paper on hardware very similar to **Figure 1**, we were able to decode problems with up to 1000 variables that were not correctly decoded by the standard BP decoding algorithm.

2. METHODS

2.1. MAPPING PROBLEMS TO ISING MODELS

This section provides an algorithm to find Ising representations of constraints with large classical gaps. To emphasize the linear dependence of the Ising energy on its parameters we write Equation (1) as[1]

$$Pen(\boldsymbol{z}|\boldsymbol{\theta}) = \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{z}) \rangle$$

where $\boldsymbol{\theta} = \left(\theta_0, (\theta_i)_{i \in V(\mathcal{G})}, (\theta_{i,j})_{(i,j) \in E(\mathcal{G})}\right)$ and $\boldsymbol{\phi}(\boldsymbol{z}) = \left(1, (z_i)_{i \in V(\mathcal{G})}, (z_i z_j)_{(i,j) \in E(\mathcal{G})}\right)$. Here θ_i are the local fields h_i, $\theta_{i,j}$ are the couplings $J_{i,j}$, and θ_0 represents a constant energy offset. When $\boldsymbol{z} = (\boldsymbol{s}, \boldsymbol{a})$, we implicitly identify certain nodes in $V(\mathcal{G})$ as decision variables $\boldsymbol{s}$, and other nodes as ancillary variables $\boldsymbol{a}$. We assume there are n decision variables and n_a ancillary variables, and that the assignment of these variables to nodes (qubits) is given. The hardware's lower and upper bounds on the parameters are denoted by $\underline{\boldsymbol{\theta}}$ and $\overline{\boldsymbol{\theta}}$ respectively (e.g., for D-Wave hardware $h_i \in [-2, 2]$ and $J_{i,j} \in [-1, 1]$).

For the purpose of this section we will assume that the number of variables, $n + n_a$, is not too large, say less than 40. We will also require that the subgraph induced by the ancillary variables, $\mathcal{G}_{\boldsymbol{a}}$, has low *treewidth*. This assumption allows us to efficiently enumerate the smallest k energy states of any Ising model in $\mathcal{G}_{\boldsymbol{a}}$ for, say, $k \leq 10000$. The treewidth of the Chimera graph in **Figure 1** that consists of $N \times N$ tiles of complete bipartite graphs $K_{4,4}$ is $4N$. In our experiments, we used up to 4×4 Chimera tiles, so the tree width of $\mathcal{G}_{\boldsymbol{a}}$ was at most 16.

We maximize the penalty gap g separating $\boldsymbol{s} \in \boldsymbol{F}$ from $\boldsymbol{s} \notin \boldsymbol{F}$ subject to the bounds on $\boldsymbol{\theta}$. This criterion gives rise to the following constrained optimization problem

$$\max_{g,\boldsymbol{\theta}} \; g \tag{3}$$

$$\text{subject to} \;\; \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}) \rangle \geq 0 \qquad \forall \boldsymbol{s} \in \boldsymbol{F}, \forall \boldsymbol{a} \tag{4}$$

$$\langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}) \rangle \geq g \qquad \forall \boldsymbol{s} \notin \boldsymbol{F}, \forall \boldsymbol{a} \tag{5}$$

$$\exists \boldsymbol{a} : \;\; \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}) \rangle = 0 \qquad \forall \boldsymbol{s} \in \boldsymbol{F} \tag{6}$$

$$\underline{\boldsymbol{\theta}} \leq \boldsymbol{\theta} \leq \overline{\boldsymbol{\theta}}.$$

Constraint (5) separates all $\boldsymbol{s} \notin \boldsymbol{F}$, while constraints (4) and (6) make sure that the minimum penalty for $\boldsymbol{s} \in \boldsymbol{F}$ is 0 [note that constraints (4)–(6) ensure (2)]. Here, constraint (6), involving the disjunction over $\boldsymbol{a}$, makes this problem difficult since we do not know which of the ancillae settings gives zero energy for a particular feasible $\boldsymbol{s}$, i.e., what is $\boldsymbol{a}^\star(\boldsymbol{s})$ at which $\langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}^\star(\boldsymbol{s})) \rangle = 0$? However, we note that solving for $\boldsymbol{\theta}$ given $\boldsymbol{a}^\star(\boldsymbol{s})$, for $\boldsymbol{s} \in \boldsymbol{F}$, is a linear programme, and can be made relatively scalable in spite of the exponential number of constraints (the low treewidth assumption makes cut generation tractable [11]). Using binary variables and linear constraints, it is straightforward to transform this optimization problem into a mixed integer linear programme (MILP). Commercial MILP solvers can typically solve this problem if $|V(\mathcal{G})|$ is no larger than about 10. For larger problems these solvers may be ineffective, partly because the linear programme relaxations are typically weak. We propose a method which scales better than a MILP.

To address the disjunctive constraint it is easiest to encode $\boldsymbol{a}^\star(\boldsymbol{s})$ by introducing $n_a|\boldsymbol{F}|$ Boolean-valued (0/1) optimization variables $\beta(\boldsymbol{s}, i)$ for $\boldsymbol{s} \in \boldsymbol{F}$ defined so that $2\beta(\boldsymbol{s}, i) - 1 = a_i^\star(\boldsymbol{s})$. The shorthand $\boldsymbol{\beta}(\boldsymbol{s})$ is used to indicate the vector of length n_a whose i-th component is $\beta(\boldsymbol{s}, i)$. Rather than directly maximizing g we have found empirically that it is faster to fix a g value and solve the following feasibility problem, FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$), to identify $\boldsymbol{\theta}$ and $\boldsymbol{\beta} \equiv \{\boldsymbol{\beta}(\boldsymbol{s}) \mid \boldsymbol{s} \in \boldsymbol{F}\}$:

FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$): find $\boldsymbol{\theta}, \boldsymbol{\beta}$ such that

$$\begin{cases} \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}) \rangle \geq 0 & \forall \boldsymbol{s} \in \boldsymbol{F}, \forall \boldsymbol{a} \\ \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}) \rangle \geq g & \forall \boldsymbol{s} \notin \boldsymbol{F}, \forall \boldsymbol{a} \\ 2\boldsymbol{\beta}(\boldsymbol{s}) - \mathbf{1} = \boldsymbol{a} \implies \langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a}) \rangle = 0 & \forall \boldsymbol{s} \in \boldsymbol{F}, \forall \boldsymbol{a} \\ \underline{\boldsymbol{\theta}} \leq \boldsymbol{\theta} \leq \overline{\boldsymbol{\theta}}. \end{cases}$$

Separately, we find the largest g for which FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$) is satisfiable. Notice that in this formulation we *ground* constraint (6) by considering all possible ancilla settings $\boldsymbol{a}$. We solve FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$) with Satisfiability Modulo Theory (SMT) solvers. SMT [12] generalizes Boolean satisfiability by allowing some Boolean variables to represent equality/inequality predicates over additional continuous variables. As mentioned earlier, for any given setting of Boolean variables, finding the continuous variables $\boldsymbol{\theta}$ is a linear programme. This linear programme can be solved using a variant of the simplex method [13] that can efficiently propagate new constraints and infer "nogoods" on the $\boldsymbol{\beta}$ variables. A number of solvers for SMT are available, and experiments here rely on the MathSat solver [14].

A naive representation of FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$) requires 2^{n+n_a} inequality constraints, $2^{n_a}|\boldsymbol{F}|$ implication constraints and $|V(\mathcal{G})| + |E(\mathcal{G})|$ bound constraints. However, we can significantly reduce the number of constraints by exploiting the fact that $\mathcal{G}_{\boldsymbol{a}}$ has low treewidth. This allows us to solve any Ising model in $\mathcal{G}_{\boldsymbol{a}}$ using dynamic programming, or more specifically, variable

[1] We indicate the energy as *Pen*, instead of *E*, to remind the reader this energy function represents a penalty function.

elimination (VE). VE exploits the observation that summation distributes over minimization, i.e., $\min(a+b, a+c) = a + \min(b, c)$. Thus, to minimize a sum of local interactions we pick an ordering in which to minimize (eliminate) one a_i variable at a time, and push minimizations as far to the right in the summation as possible. If we record these minimizations in tables then they can be reused in different contexts to save recomputation. Bucket elimination [15] is a convenient way to structure this memoization, and can be encoded nicely within our constraint model.

More concretely, we solve an Ising model in $\mathcal{G}_{\boldsymbol{a}}$ by storing partial computations and spin states in tables as follows. Suppose that ancilla variables are eliminated in the order $a_{n_a}, a_{n_a-1}, \ldots, a_1$. Let $\mathcal{V}_{n_a}$ be the set of ancilla variables adjacent to a_{n_a} in $\mathcal{G}_{\boldsymbol{a}}$. Clearly, for each setting of all variables in $\mathcal{V}_{n_a}$ we can deduce the optimal value of a_{n_a}. This information is collected in $\mathcal{T}_{n_a}$, a table of size $2^{|\mathcal{V}_{n_a}|}$. This table is assigned to the variable of largest index in $\mathcal{V}_{n_a}$ (alternatively, one can think that the variables in $\mathcal{V}_{n_a}$ become induced neighbors of each other). In general, $\mathcal{V}_i$ consists of the neighbors of variable a_i together with the variables on the tables assigned to a_i but excluding a_i itself (alternatively all its neighbors and induced neighbors). We then create $\mathcal{T}_i$, a table of size $2^{|\mathcal{V}_i|}$, for all possible values of variables in $\mathcal{V}_i$. Notice that for each setting of the variables in $\mathcal{V}_i$ we can calculate the smallest contribution to the objective of variable a_i by adding linear and quadratic local terms to values from previously generated tables. This information is stored in $\mathcal{T}_i$ and assigned to the variable of largest index in $\mathcal{V}_i$ (alternatively, the variables in $\mathcal{V}_i$ become induced neighbors of each other). In this way, after all variables have been eliminated we end up with one or more tables with a single entry (i.e., the corresponding $\mathcal{V}_i$ is empty). The sum over the values stored in these singleton tables is the value of the optimal solution of the Ising model. Notice that the storage necessary for the tables is $O(\sum_{i=1}^{n_a} 2^{|\mathcal{V}_i|})$. For our work here, the important observation is that one can easily find a variable elimination order for which $\max_i |\mathcal{V}_i|$ is the treewidth of $\mathcal{G}_{\boldsymbol{a}}$ using Gogate and Dechter [16]. In what follows, we assume that ancilla variables are ordered using this ordering.

In our case, the constraints are not as simple as just solving an Ising model in $\mathcal{G}_{\boldsymbol{a}}$, since some of the coefficients are themselves variables (entries of $\boldsymbol{\theta}$). Nevertheless, in this parametric Ising model, each table entry can be replaced by a continuous variable $m_i(\boldsymbol{v}_i|\boldsymbol{s})$ (a *message* conveying all required information from previously eliminated variables), indexed by the decision variable setting $\boldsymbol{s}$ and a setting $\boldsymbol{v}_i$ of variables in $\mathcal{V}_i$. Thus, the constraints on $\boldsymbol{s}$ Equations (4)–(6) become

$$\sum_{i\,|\,\mathcal{V}_i=\emptyset} m_i(\emptyset\,|\,\boldsymbol{s}) \geq g \quad \forall \boldsymbol{s} \notin \boldsymbol{F} \quad \text{and}$$
$$\sum_{i\,|\,\mathcal{V}_i=\emptyset} m_i(\emptyset\,|\,\boldsymbol{s}) = 0 \quad \forall \boldsymbol{s} \in \boldsymbol{F}\ . \tag{7}$$

The purpose of message $m_i(\boldsymbol{v}_i|\boldsymbol{s})$ is to eliminate the ancilla setting of a_i. Since in this case the Ising model is defined by variables $\boldsymbol{\theta}$, we do not know which setting of a_i makes the contribution to the Ising model smallest. However, we can upper bound $m_i(\boldsymbol{v}_i|\boldsymbol{s})$ using the two possible values of a_i, imposing two inequalities on $m_i(\boldsymbol{v}_i|\boldsymbol{s})$. This suffices for the case when $\boldsymbol{s} \notin \boldsymbol{F}$ because the messages will be themselves a lower bound on the value of the Ising model and we only require this value to be at least g in (7). When $\boldsymbol{s} \in \boldsymbol{F}$, we must make sure that the message takes the exact minimum value for the parametric Ising model. We impose these constraints by making sure that when a_i corresponds to $\beta(\boldsymbol{s}, i)$, we lower bound the message so it takes the correct value.

2.1.1. Implementation concerns

For some problems many of the constraints arising from variable elimination are redundant, and many message variables can be shown to be equal. Eliminating such redundancies can dramatically shrink the size of FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$). Further consolidation can be obtained by exploiting automorphisms of $\mathcal{G}$, and gauge symmetries of the Ising energy $\langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{z})\rangle$. Lastly, we note that the order in which constraints are presented to the SMT solver can significantly impact solving time. We have found that running multiple SMT solver instances each with a random shuffling of constraints often yields at least one solution quickly.

2.1.2. Scalability

FEAS($\boldsymbol{\theta}, \boldsymbol{\beta}$) can be very difficult to solve (particularly at larger g). Currently, we are limited to problems defined on graphs $\mathcal{G}$ with at most 30–50 nodes, and up to $|\boldsymbol{F}| = 1000$ feasible configurations. To scale to larger sizes requires heuristics which give suboptimal gaps. One approach to better scaling is through graph embedding. With embedding a problem is reduced to pairwise interactions without regard for the connectivity this reduction may generate. The connectivity is then mimicked in hardware with strong ferromagnetic interactions that slave qubits together: techniques for doing so are discussed in Section 2.2.

2.1.3. Examples

Consider the 3-bit parity check equation, that is, $\boldsymbol{F}$ is the set of four feasible solutions consisting of spin triplets with an even number of positive spins. Realizing this parity constraint as an Ising model requires at least one ancillary bit. The bound constraints are $h_i \in [-2, 2]$ for each i and $J_{ij} \in [-1, 1]$ for each edge (i, j). We assume first that $\mathcal{G}$ is the complete graph on 4 nodes, K_4. Then it is straightforward to verify that $(s_1 + s_2 + s_3 - 2a + 1)^2/4$ defines an optimal penalty function with gap 1. The same gap can be achieved if the hardware graph is the complete bipartite graph $K_{3,3}$ and we identify two qubits on the right side of the partition with two on the left side using ferromagnetic couplings. The first two decision variables are mapped to these two pairs of coupled qubits, so that, $s_1 = a_1$ and $s_2 = a_2$, while the two remaining qubits are s_3 and an additional ancilla a_3. In this case, the Ising model is

$$0.5(s_1 + s_2 + s_3) - a_3 + s_1(-a_1 + 0.5a_2 - a_3)$$
$$+s_2(-a_2 - a_3) + s_3(0.5a_1 + 0.5a_2 - a_3).$$

However, using the MILP or SMT model we can obtain a gap of 2 by placing the decision qubits s_1, s_2, s_3 on one side of $K_{3,3}$, and all ancillary qubits a_1, a_2, a_3 on the other:

$$-a_1 + a_2 - a_3 + s_1(a_1 + a_2 + a_3)$$
$$+s_2(-a_1 + a_2 + a_3) + s_3(a_1 + a_2 - a_3).$$

As a more complex example consider the constraint $\sum_{i=1}^{8} s_i = -6$ (perhaps indicating one of 8 objects). This can be represented with the penalty $\left(\sum_{1 \le i \le 8} s_i + 6\right)^2$ which couples all 8 variables. Embedding this penalty using the methods of Section 2.2.3 in the hardware graph of **Figure 1** requires 24 qubits and gives a gap of 2/3.

However, using the SMT model we can accommodate the same constraint with only 16 qubits, and a gap of 4. The Ising models for these two penalties are shown in **Figure 2**.

2.2. MAPPING ISING MODELS TO HARDWARE

Using the techniques in Section 2.1, we realize the constraints $\{\boldsymbol{F}_j \mid 1 \le j \le m\}$ of a CSP as penalty Ising models $\{Pen_{F_j}(\boldsymbol{s}, \boldsymbol{a}) \mid 1 \le j \le m\}$, where each Pen_{F_j} is defined on a small subgraph $\mathcal{G}_j$ of the hardware graph $\mathcal{G}$. By choosing the subgraphs to be disjoint (or possibly intersecting at decision variables), we can solve all constraints simultaneously on the hardware. However, most variables s_i will appear in multiple constraints, and we require that all instances of a variable take the same value. To do this, we identify several connected qubits with the same variable, and apply a strong ferromagnetic connection between those qubits during the annealing process, ensuring that they obtain the same spin. A connected set of qubits representing the same variable is known as a *chain*. Notice that the variables in a chain connecting two variables which must assume the same value are ancillary variables enforcing the equality constraint. Chains are simply penalty functions enforcing equality constraints.

The problem of embedding the Ising model of a CSP into $\mathcal{G}$ then consists of two parts: (1) choosing a placement of constraints onto disjoint subgraphs of $\mathcal{G}$, and (2) routing chains to represent variables that appear in multiple constraints. This "place and route" model of embedding has been used with great success and scalability in VLSI physical design [17, 18], where circuits with millions of variables may be mapped onto a single chip. Some of the techniques presented here have analogies in the VLSI literature. In this section, we describe techniques for placement and routing, as well as more general methods to map Ising models of arbitrary structure to $\mathcal{G}$.

2.2.1. Placement

We consider graphs $\mathcal{G}$ consisting of a repeating pattern of unit cells, and assume that constraints (like parity check on 3 variables) can be represented within a single unit cell. In mapping constraints to unit cells within $\mathcal{G}$, we try to place constraints that share variables close to one another: a good choice of placement will make the routing process more tractable. The techniques presented here can be generalized, but for simplicity we assume that the hardware graph $\mathcal{G}$ consists of an $N \times N$ grid of $K_{4,4}$ unit cells. Approaches to placement include:

- *Quadratic assignment*: Define a flow $A_{j,j'}$ between two constraints $\boldsymbol{F}_j$ and $\boldsymbol{F}_{j'}$ to be the number of variables they have in common. Define a distance $B_{u,u'}$ between two unit cells u and u' in $\mathcal{G}$ to be the length of the shortest path between them. Assuming two instances of a variable must be joined together by a path to create a chain, the quadratic assignment problem QAP(A, B) identifies a mapping from constraints to cells that roughly minimizes the sum of the chain lengths. (See [19] for background on QAP). Note that we do not require an optimal solution to the QAP problem in order to find a valid embedding; an approximate solution suffices.
- *Simulated annealing*: Simulated annealing updates an assignment of constraints to cells by swapping pairs of constraints. Swaps are chosen randomly and accepted or rejected as a function of the change in some cost function (the cost function

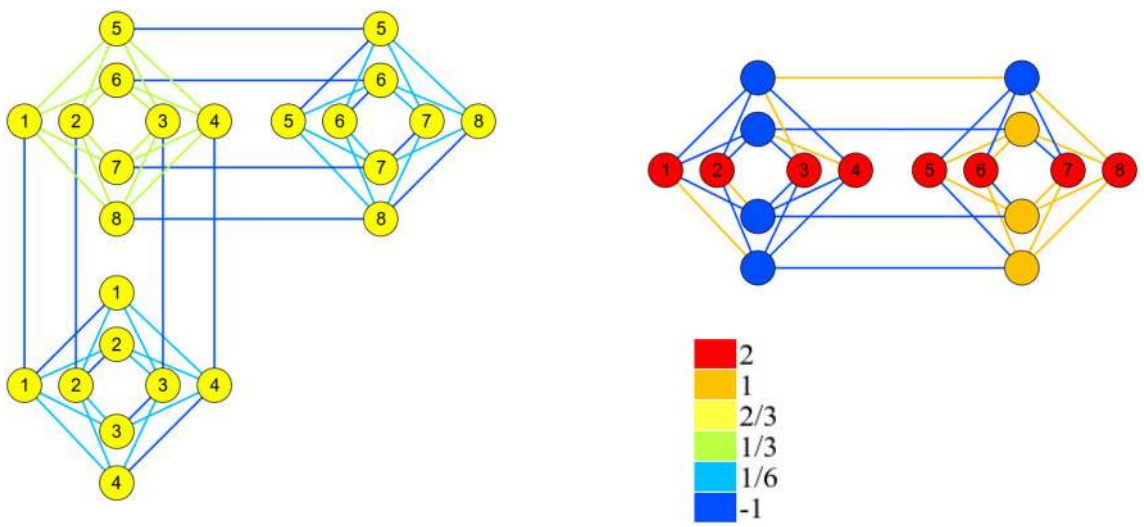


FIGURE 2 | Penalty functions for the constraint $\sum_{i=1}^{8} s_i = -6$. The left graph represents the penalty function that arises from embedding a K_8, The right graph shows the penalty found by the SMT solver. θ_i values are indicated by node colors, and $\theta_{i,j}$ values by edge colors. The node labeled i corresponds to variable s_i, and unlabeled nodes are ancillae.

used in QAP is one example). Provided that each constraint only shares variables with a small number of other constraints, changes in cost function can be evaluated quickly.
- *Recursive placement*: As the hardware graph gets larger, both simulated annealing and quadratic assignment become computationally prohibitive. The cost can be reduced by recursively splitting the problem into pieces and then combining the solutions. First we partition the constraints into two regions such that the number of variables shared between regions is minimized, Then we partition the unit cells of $\mathcal{G}$ in two regions such that the number of edges between them is minimized. We continue to partition until the regions are small enough to be computationally tractable. Techniques for partitioning are discussed in Section 2.3.

2.2.2. Routing

Once constraints, and therefore variables, have been assigned to disjoint subgraphs of $\mathcal{G}$, different instances of a variable must be joined together. The routing problem is formulated as follows: given $\mathcal{G}$ and a collection of disjoint terminal sets $\{T_i \subseteq V(\mathcal{G}) \mid i = 1 \ldots m\}$ representing the qubits on which variable s_i has already been placed, find a collection of disjoint chains $\{S_i\}_{i=1}^m$ such that S_i contains T_i. The performance of the D-Wave hardware is dependent on the choice of chains, so we would also like to minimize either the maximum size of any chain or the total number of qubits used. This routing problem differs from traditional VLSI routing only in the choice of objective function and the hardware graph $\mathcal{G}$, so several VLSI methods apply with some modification:

- *Multicommodity flow*: The problem of finding a tree S_1 in $\mathcal{G}$ of minimal size containing a given terminal set T_1 is known as the Steiner Tree problem on graphs and is NP-hard. For a given $T_1 = \{z_1, z_2, \ldots, z_{k_1}\}$, we can model a Steiner Tree for T_1 as a network flow. Root vertex z_1 is a source with $k_1 - 1$ units of commodity, while each of $\{z_2, \ldots, z_{k_1}\}$ is a sink requiring one unit of commodity. All other vertices of $\mathcal{G}$ have a net-flow of zero. Then, a Steiner Tree S_1 for T_1 exists if and only if there is a feasible flow in this network. By adding binary variables that indicate whether or not a vertex of $\mathcal{G}$ is in S_1, we can easily formulate this problem as a mixed integer linear programme (MILP).
  The general case, when $m > 1$, can be formulated as a MILP using m commodities, one for each terminal set (see [20, Ch. 3.6] or [21] for background on multicommodity flows). We add the constraint that each vertex of $\mathcal{G}$ can be in at most one Steiner tree, and require that the flow of commodity i can only be routed through vertices in Steiner tree S_i.
- *Greedy Steiner Trees*: We heuristically select a variable order and then greedily choose a Steiner Tree for each s_i from the subgraph $\mathcal{F}$ of unused qubits in $\mathcal{G}$. Several approximation algorithms for the Steiner tree problem are known, but perhaps the simplest is Kou et al. [22]: define a weighted complete graph $\mathcal{K}$ on $V(\mathcal{F})$, where the weight of an edge z_1z_2 is the shortest-path distance between z_1 and z_2 in $\mathcal{F}$. Choose a minimum spanning tree in $\mathcal{K}$, and then assign s_i to every vertex in $\mathcal{F}$ on the paths representing edges in the minimum spanning tree. This creates a chain for s_i.
- *Rip-up routing*: Rip-up routing [18, 23, 24] is a variation of the greedy Steiner Tree algorithm that temporarily allows chains of variables to overlap. For each variable s_i, we maintain a chain S_i such that $T_i \subseteq S_i$. Initially S_i is set to T_i, but after the first iteration of the algorithm each S_i is connected, and by the end of the algorithm the chains are (hopefully) disjoint. The algorithm iteratively updates each chain S_i as follows:

  1. For each vertex z in $\mathcal{G}$, define a vertex weight $wt(z) = \alpha^{|\{j \neq i : z \in S_j\}|}$, for some fixed $\alpha > 1$.
  2. Using a Steiner Tree approximation algorithm, choose an approximately minimal vertex-weighted Steiner Tree S_i for the terminal set T_i.

  The algorithm terminates when all S_i are disjoint or no improvements are found. By setting the weight of a vertex proportional to the number of variables it represents, the algorithm encourages chains to form on unused vertices whenever possible.

Multicommodity flow is an exact algorithm and therefore most successful when it is tractable (up to roughly 500 qubits). On the other hand Steiner tree approximation algorithms which are polynomial time can be used at much larger scales, and rip-up routing is usually more effective than greedy routing as it is less likely to get trapped in suboptimal local minima.

2.2.3. Global embedding techniques

For certain optimization problems it may be difficult or suboptimal to map individual constraints $\boldsymbol{F}_j$ to Ising models with the connectivity structure of $\mathcal{G}$. In these cases, we may instead map each $\boldsymbol{F}_j$ to an Ising model $Pen_{F_j}(\boldsymbol{z})$ of arbitrary structure (possibly using ancillary variables), and then attempt to map $Pen(\boldsymbol{z}) = \sum_j Pen_{F_j}(\boldsymbol{z})$ to $\mathcal{G}$ directly. Because of the limited qubit connectivity, we will again require chains to represent variables. The techniques described here attempt to map a problem graph $\mathcal{P}$ (in which z_i and z_j are adjacent if they have a non-zero interaction in $Pen(\boldsymbol{z})$) to $\mathcal{G}$ without assuming $\mathcal{P}$ or $\mathcal{G}$ have any particular structure.

Constructing chains such that if two variables are adjacent in $\mathcal{P}$ then there is at least one edge between their chains in $\mathcal{G}$ is known as the minor-embedding problem [25]. Minor-embedding is NP-hard; the best known algorithm has running time $O(2^{(2k+1)\log k}|V(\mathcal{P})|^{2k}2^{2|V(\mathcal{P})|^2}|V(\mathcal{P})|)$, where k is the branchwidth of $\mathcal{G}$ [26]. While there are deep theoretical results about minors in the theory developed by Robertson and Seymour [27], none of the known exact algorithms are even remotely practical for the scale of problems we are interested in. Nevertheless, heuristics can be effective provided that $\mathcal{P}$ is not too large compared to $\mathcal{G}$. When $\mathcal{G}$ is the Chimera graph in **Figure 1**, one strategy is to treat $\mathcal{P}$ as a subgraph of a fixed complete graph. The ideal Chimera graph with $8N^2$ qubits was designed to have a minor-embedding of a complete graph on $4N + 1$ variables [9, 25], and [28] provides algorithms for embedding complete graphs when qubits are missing. The heuristics described below, while slower, can embed much larger problems and also have more flexibility than constraint-based techniques. These heuristics may

also be used to improve chain lengths of an embedding, regardless of how that embedding was found.

2.2.4. Shortest-path-based chains

This algorithm uses efficient shortest-path computations to construct a chain for each variable, based on the locations of its neighbors' chains. Chains are temporarily allowed to overlap, and each qubit is assigned a weight based on how many variables are currently assigned to it. Suppose we want to find a chain S_0 for variable s_0, and s_0 is adjacent to $s_1, \ldots, s_k$ which already have chains $S_1, \ldots, S_k$ respectively. By computing the weighted shortest path from each S_i to every other qubit in $\mathcal{G}$, we can select a "root" qubit z^* for S_0 which minimizes the weight of the qubits needed to create a connection between s_0 and each of $s_1, \ldots, s_k$. We then take the union of the shortest paths from z^* to each S_j as the chain for S_0. The details of this algorithm can be found in Cai et al. [in preparation].

2.2.5. Simulated annealing

An alternative approach uses simulated annealing to attempt to improve partial embeddings. A partial embedding is an assignment of a chain S_i to each variable $s_i \in V(\mathcal{P})$ such that all S_i are disjoint. An edge $s_i s_j$ of $\mathcal{P}$ is unfulfilled if there is no edge in $\mathcal{G}$ joining S_i and S_j. The partial embedding is assigned a score, consisting of the sum of the distances between S_i and S_j for unfulfilled edges $s_i s_j$ plus a small positive constant times the sum of the squares of the cardinalities of all chains. We try to minimize the score by considering moves of the following types:

1. Given qubit z that is not in any chain, but is adjacent to a chain S_i, adjoin z to S_i.
2. Given qubit z in chain S_i, remove z from S_i (if $S_i \backslash \{z\}$ is still a valid chain) and either leave it unassigned or adjoin it to a neighboring chain S_j.
3. Given two qubits in different chains, $z_1 \in S_i$ and $z_2 \in S_j$ respectively, if z_1 is adjacent to S_j and z_2 is adjacent to S_i, and $S_i \backslash \{z_1\}$ and $S_j \backslash \{z_2\}$ are still valid chains, switch z_1 from S_i to S_j and z_2 from S_j to S_i.

Simulated annealing often produces better results than shortest-path-based chains but takes much longer.

2.3. SOLVING LARGER PROBLEMS

The tools in Sections 2.1 and 2.2 allow for mapping of arbitrary CSPs to Ising models with connectivity constraints. However, it may be difficult to fit a given problem in the current D-Wave hardware due to its limited number of qubits. In this section we outline approaches to solving large problems by repeatedly calling QA hardware to optimize smaller subproblems.

One way of dealing with a large problem is to decompose it into *regions*. Smaller regional subproblems are then solved, and each region sends some form of feedback to its neighboring regions, which in turn modifies each regional subproblem. The process is repeated until all regions agree on shared variables or some stopping criteria is triggered.

Each region of a CSP is, essentially, a smaller CSP. When we partition a CSP into regions, each region should be as large as possible subject to being embeddable in the hardware graph. Partitions should be chosen so that regions have as few variables in common as possible to minimize the communication between regions.

To decompose the CSP in this way we may use graph and hypergraph partitioning techniques. One mechanism is the following: define a node for each constraint of the CSP and a weighted edge between two nodes counting the variables their constraints share. Then a min-cut balanced partition of the graph will produce similar regions with a minimal number of pairs of shared variables. The min-cut balanced partitioning problem is NP-hard in general, but the Kernighan-Lin algorithm [29] performs well in practice. It starts with a random partition and iteratively swaps sequences of vertices between regions based on improvements to the edge-weights. The software package Metis [30] has implemented a "multi-level" version of Kernighan-Lin that is scalable to tens of thousands of nodes. Since embedding in the hardware graph puts bounds on the number of variables, for regional decomposition to be effective the problem must exhibit some sparsity.

Decomposing the Ising model for the original CSP into $\mathcal{R}$ regions produces similar regional subproblems; a region R will have couplings $\hat{h}^{(R)}$ and $\hat{J}^{(R)}$ derived from (and often equal to) those of (1), yielding a problem

$$\min_{\boldsymbol{z} \in \{-1,1\}^{n_R}} \left(\sum_{i \in V^{(R)}} \hat{h}_i^{(R)} z_i + \sum_{(i,j) \in E^{(R)}} \hat{J}_{i,j}^{(R)} z_i z_j \right), \tag{8}$$

on a graph $\mathcal{G}^{(R)} = (V^{(R)}, E^{(R)})$ (where $n_R = |V^{(R)}|$). We assume the size and sparsity structure of this problem, defined by $G^{(R)}$, is compatible with the hardware. Next, we consider two approaches to coordinate the solutions of these regional subproblems produced by the decomposition. In our implementations, we chose the regions so that $h_i^{(R)}$ splits h_i evenly across regions R containing qubit i, but each edge $ij \in E(\mathcal{G})$ belongs to exactly one region so that $J_{i,j}^{(R)} = J_{i,j}$. During the run of the algorithms, for each region R, only the linear terms were updated.

2.3.1. Belief propagation

Belief propagation (BP) [31] is an algorithm in which messages are passed between regions and variables. Messages represent beliefs about the minimal energy conditional on each possible value of a variable. In min-sum BP, messages from variables $(i, j, \ldots)$ to regions $(R, S, \ldots)$ are updated using the formula

$$\mu_{i \to R}(z_i) := \sum_{S \in N(i) \backslash R} \mu_{S \to i}(z_i),$$

where $N(i)$ is the set of regions containing i. In the reverse direction, messages are updated as

$$\mu_{R \to i}(z_i) := \min_{\boldsymbol{z}_{N(R) \backslash z_i}} \left\{ \sum_{j \in V^{(R)}} h_j^{(R)} z_j + \sum_{(j,k) \in E^{(R)}} J_{j,k}^{(R)} z_j z_k + \sum_{j \in N(R) \backslash i} \mu_{j \to R}(s_j) \right\}, \tag{9}$$

where similarly $N(R)$ is the set of variables in R.

Specifically, $\mu_{i\to R}(z_i)$ represents the aggregate of beliefs about z_i from all regions excluding R, while $\mu_{R\to i}(z_i)$ is the minimum energy for R aggregating beliefs about all variables in R excluding i. These messages are iteratively passed between regions and variables until messages converge or another criterion is met when messages do not converge. BP is known to perform well for certain CSPs, and is a standard decoding algorithm for LDPC [32].

Two advantages of using hardware-sized regions, over performing BP in which each region is a single constraint, are (1) we can internalize within regions many of the short cycles of variable interactions, which are known to cause failure of BP, and (2) we need only pass messages for variables that appear in multiple regions, as variables appearing in a single region may be resolved after the BP algorithm terminates. Using min-cut heuristics to construct regions augments these benefits.

2.3.2. Dual decomposition

Dual decomposition (DD) uses Lagrangian relaxation methods, a standard approach for dealing with large scale optimization problems [11, 33, 34]. The Lagrangian relaxation of an Ising model (1) consists of a concave optimization problem of the form $\max_{\boldsymbol{\lambda}\in\Lambda} L(\boldsymbol{\lambda})$ where

$$L(\boldsymbol{\lambda}) = \sum_{R=1}^{\mathcal{R}} \min_{\boldsymbol{z}^{(R)}} \left[\left\langle \boldsymbol{h}^{(R)} + \boldsymbol{\lambda}^{(R)}, \boldsymbol{z}^{(R)} \right\rangle + \left\langle \boldsymbol{z}^{(R)}, \boldsymbol{J}^{(R)} \boldsymbol{z}^{(R)} \right\rangle \right],$$

where for region R, $\boldsymbol{h}^{(R)} = (h_i^{(R)})_{i\in V(\mathcal{G}^{(R)})}$, $\boldsymbol{J}^{(R)} = (J_{i,j}^{(R)})_{(i,j)\in E(\mathcal{G}^{(R)})}$. Valid multipliers $\boldsymbol{\lambda}$ must lie within $\Lambda = \{\boldsymbol{\lambda} \mid \sum_{R=1}^{\mathcal{R}} \boldsymbol{\lambda}^{(R)} = \mathbf{0}\}$. As before the Ising models defined by $\boldsymbol{h}^{(R)}$ and $\boldsymbol{J}^{(R)}$ have sparsity structure $\mathcal{G}^{(R)}$ determined once after the partitioning, and the multipliers $\boldsymbol{\lambda}$ do not change the sparsity structure of $\boldsymbol{J}^{(R)}$. Notice that evaluating $L(\boldsymbol{\lambda})$ (and thus computing a supergradient) decomposes into $\mathcal{R}$ independent regional subproblems of the form (8). The concave optimization problem that defines the relaxation can be solved using subgradient optimization methods (e.g., [33]).

The Lagrangian relaxation is a lower bound on the optimal value of the original Ising model. In particular, if after solving the Lagrangian function $L(\boldsymbol{\lambda})$ all the solutions $\boldsymbol{z}^{(R)}$ agree at the regional boundaries, we have indeed solved the original problem. If the solutions $\boldsymbol{z}^{(R)}$ differ at regional boundaries, a simple heuristic to try to derive a solution to the CSP is to use regional majority vote or randomized rounding to determine the values of the spins at the boundaries followed by a straightforward greedy descent procedure.

The Lagrangian relaxation can also be used to compute lower bounds in a branch-and-bound approach.

2.3.3. Large-neighborhood local search

Greedy descent or local search is an iterative optimization method that moves from one spin configuration $\boldsymbol{z}$ to another by flipping the spin that reduces the objective value of the Ising model the most. A straightforward variation of local search [35] is large-neighborhood local search (LNLS) (see [36]), in which many spins may be flipped simultaneously. In each iteration of LNLS, we select a subset of the spins $\tilde{\boldsymbol{z}}$ that are allowed to change (the rest are fixed), and we optimize the subproblem over $\tilde{\boldsymbol{z}}$ in hardware so as to minimize the overall objective value of the Ising model.

In contrast with the decomposition methods in Sections 2.3.1 and 2.3.2, LNLS does not rely on a single partitioning of the original problem: a new subproblem may be selected at each iteration. However, we must ensure that subproblems are embeddable in hardware. For instance, if we have an embedding of a complete graph K_m, any subset of m variables can be optimized while keeping all the other spins fixed. An important consideration is the selection of the variables that are fixed at each iteration. One heuristic is to pick a variable at random and grow a breadth-first search tree to a fixed number of variables.

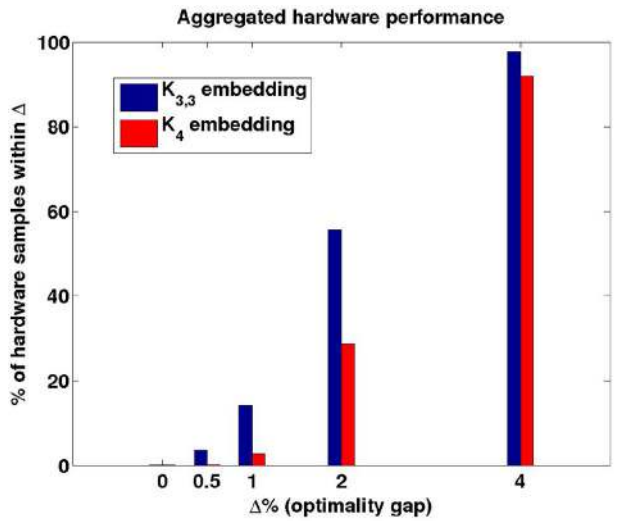
3. RESULTS

In this section, we report results of our experiments using D-Wave hardware for solving LDPC problems. In our tests, an instance of LDPC consists of a parity check matrix $\boldsymbol{G} \in \{0,1\}^{r\times n}$, and a bit string (message) $\boldsymbol{y} \in \{0,1\}^n$. A codeword $\boldsymbol{x}$ is any bit string whose parity check vector, $\boldsymbol{Gx}$, is $\mathbf{0}$ modulo 2. The goal is to *decode* message $\boldsymbol{y}$ by finding the codeword $\boldsymbol{x}$ whose Hamming distance to $\boldsymbol{y}$ is as small as possible. The parity check matrix $\boldsymbol{G}$ is sparse. We chose $\boldsymbol{G}$ so that n was between 100 and 1000, $r \approx 0.70n$, the number of non-zeros on each row was between 3 and 5, and the number of non-zeros in each column was between 2 and 4. The message $\boldsymbol{y}$ was chosen by first picking a random codeword $\boldsymbol{x}$, and then flipping pn random bits, where p is the *error rate*. We tested instances for which $p \in \{8\%, 10\%, 12\%, 14\%\}$.

The standard decoding algorithm for LDPC uses belief propagation as described in Section 2.3.1, with each region defined to be a single parity check. We generated our instances at random, and, to get an indication of the potential usefulness of the hardware, considered only instances that had a unique optimal solution and for which standard belief propagation did not converge within 1000 iterations. Optimal solutions used to baseline hardware performance were obtained by dynamic programming applied to regional subproblems [16]. While dynamic programming is faster than hardware solution at these size subproblems, dynamic programming will not scale to much larger subproblems due to prohibitive memory requirements.

We solved the instances using D-Wave hardware (such as the one in **Figure 1**, for which $h_i \in [-2, 2]$ and $J_{i,j} \in [-1, 1]$) and the decomposition strategies of Section 2.3. Each region consisted of up to 20 parity checks from $\boldsymbol{G}$ and was mapped to the hardware in the following way. First we added ancillary variables so that each parity check involved exactly 3 variables[2]. Each of these checks was then placed in a cell using the techniques of Section 2.2.1. Inside a cell we used the two embeddings described in the example of Section 2.1.3, one with gap 2 ($K_{3,3}$) and one with gap 1 (K_4). We observed that the problems submitted to the hardware using $K_{3,3}$-based embeddings had a significantly improved success probability as compared to the problems that used K_4-based embeddings (see **Figure 3**). This in turn allowed for higher precision for the subproblems

[2]Embedding 3-bit checks which fit within a unit cell and offer free qubits for routing is more efficient than embedding the penalties of 4- or 5-bit checks.



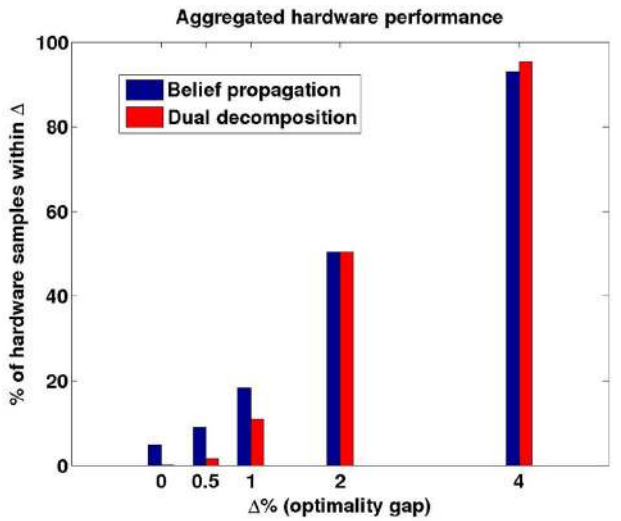


FIGURE 3 | Hardware performance aggregated over all regional subproblems. Δ is the relative error of a solution returned by the hardware compared to the true optimal value of the Ising model. **Left**: comparing $K_{3,3}$-based embedding with K_4-based embedding. **Right**: comparing Belief Propagation and Dual Decomposition.

submitted to the hardware as required by the decomposition algorithms. Thus, in the following experiments we used $K_{3,3}$-based embeddings.

We implemented both region-based Belief Propagation (BP) and Dual Decomposition (DD). We optimized each regional subproblem using up to 10 D-Wave hardware calls, in order to adjust the strength of the ferromagnetic couplings used to identify chains. Each hardware call took 10,000 samples with an annealing time of 20 μs (see [37] for more details on the hardware's operating specifications). In the case of DD, we used a standard subgradient algorithm once, and used a simple randomized rounding procedure and local search to try to produce codewords from the subproblems generated throughout the subgradient optimization.

We generated 18 random instances as described above. BP managed to solve all the instances, while DD solved 14 of 18, encountering difficulties for error rates $p = 12\%$ for instances with more than 600 bits. We attribute the higher reliability of BP to the fact that the problems sent to the hardware had lower precision than DD. On the other hand, typically DD required fewer hardware calls to converge. Among all the hardware samples we collected, 95% of them were within 4% of optimum (see **Figure 3**).

4. CONCLUSION

We have outlined a general approach for coping with intrinsic issues related to the practical use of quantum annealing. To address these issues we proposed methods for finding Ising problem representations that have a large classical gap between ground states and first excited states, practical methods for embedding Ising models that are not compatible with the hardware graph, and decomposition methods to solve problems that are larger than the hardware. As an application of our techniques, we described how we implemented LDPC decoding problems in D-Wave hardware. Our approach has enabled us to solve LDPC decoding problems of up to 1000 variables. The current hardware implementation of QA tested here is roughly as fast as an efficient implementation of simulated annealing, but these results offer the promise of hybrid quantum/classical algorithms that surpass purely classical solution as QA hardware matures.

As future work, we would like to improve upon the scalability of the current method for constructing penalty functions with large gaps. This would allow larger component subproblems and reduce the need for minor embedding between subproblems. Further, the methods we have described here for finding penalty functions assume an assignment of decision variables to qubits. Different assignment choices lead to different results and different hardware performance. We do not currently have an effective method for this assignment.

REFERENCES

1. Berkley AJ, Johnson MW, Bunyk P, Harris R, Johansson J, Lanting T, et al. A scalable readout system for a superconducting adiabatic quantum optimization system. *Superconduct Sci Technol.* (2010) **23**:105014. doi: 10.1088/0953-2048/23/10/105014
2. Dickson NG, Johnson MW, Amin MH, Harris R, Altomare F, Berkley AJ, et al. Thermally assisted quantum annealing of a 16-qubit problem. *Nat Commun.* (2013) **4**:1903. doi: 10.1038/ncomms2920
3. Harris R, Johansson J, Berkley AJ, Johnson MW, Lanting T, Han S, et al. Experimental demonstration of a robust and scalable flux qubit. *Phys Rev B* (2010) **81**:134510. doi: 10.1103/PhysRevB.81.134510
4. Johnson MW, Amin MHS, Gildert S, Lanting T, Hamze F, Dickson N, et al. Quantum annealing with manufactured spins. *Nature* (2011) **473**:194–8. doi: 10.1038/nature10012
5. Finilla AB, Gomez MA, Sebenik C, Doll DJ. Quantum annealing: a new method for minimizing multidimensional functions. *Chem Phys Lett.* (1994) **219**:343–348.
6. Kadowaki T, Nishimori H. Quantum annealing in the transverse Ising model. *Phys Rev E* (1998) **58**:5355. doi: 10.1103/PhysRevE.58.5355
7. Childs A, Farhi E, Preskill J. Robustness of adiabatic quantum computation. *Phys Rev A* (2001) **65**:012322. doi: 10.1103/PhysRevA.65.012322

8. Farhi E, Goldstone J, Gutmann S, Sipser M. *Quantum Computation by Adiabatic Evolution* (2000). Available online at: http://arxiv.org/abs/quant-ph/0001106
9. Bunyk PI, Hoskinson E, Johnson MW, Tolkacheva E, Altomare F, Berkley AJ, et al. *Architectural Considerations in the Design of a Superconducting Quantum Annealing Processor* (2014). Available online at: http://arxiv.org/abs/1401.5504
10. MacKay DJC. *Information Theory, Inference and Learning Algorithms.* New York, NY: Cambridge University Press (2002).
11. Nemhauser GL, Wolsey LA. *Integer and Combinatorial Optimization*. New York, NY: Wiley-Interscience (1988). doi: 10.1002/9781118627372
12. de Moura L, Bjørner N. Satisfiability modulo theories: introduction and applications. *Commun ACM* (2011) **54**:69–77. doi: 10.1145/1995376.1995394
13. Dutertre B, de Moura L. A fast linear-arithmetic solver for DPLL(T). In: Ball T, Jones RB, editors. *Computer Aided Verification. Vol. 4144 of Lecture Notes in Computer Science.* Berlin; Heidelberg: Springer (2006). p. 81–94. doi: 10.1007/11817963-11
14. Cimatti A, Griggio A, Schaafsma B, Sebastiani R. The MathSAT5 SMT solver. In: Piterman N, Smolka S, editors. *Proceedings of TACAS. Vol. 7795 of LNCS.* (Springer) 2013. Available online at: http://mathsat.fbk.eu/download.html
15. Dechter R. Bucket elimination: a unifying framework for reasoning. *Artif Intell.* (1999) **1–2**:41–85. doi: 10.1016/S0004-3702(99)00059-4
16. Gogate V, Dechter R. A complete anytime algorithm for Treewidth. In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence.* Arlington, VA: AUAI Press (2004). p. 201–8.
17. Gerez SH. *Algorithms for VLSI Design Automation, 1st Edn*. New York, NY: John Wiley and Sons, Inc. (1999).
18. Kahng AB, Lienig J, Markov IL, Hu J. *VLSI Physical Design - From Graph Partitioning to Timing Closure.* New York, NY: Springer (2011). doi: 10.1007/978-90-481-9591-6
19. Burkard R, Dell'Amico M, Martello S. *Assignment Problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics (2009). doi: 10.1137/1.9780898717754
20. Cook WJ, Cunningham WH, Pulleyblank WR, Schrijver A. *Combinatorial Optimization*. New York, NY: John Wiley and Sons, Inc. (1998).
21. Shragowitz E, Keel S. A global router based on a multicommodity flow model. *Integr VLSI J.* (1987) **5**:3–16. doi: 10.1016/S0167-9260(87)80003-2
22. Kou LT, Markowsky G, Berman L. A fast algorithm for steiner trees. *Acta Inf.* (1981) **15**:141–5. doi: 10.1007/BF00288961
23. Nair R. A simple yet effective technique for global wiring. *Trans Comp-Aided Des Integ Cir Sys.* (2006) **6**:165–72. doi: 10.1109/TCAD.1987.1270260
24. Ting BS, Bou NT. Routing techniques for gate array. *Computer-Aided Design Integr Circ Syst IEEE Trans.* (1983) **2**:301–12. doi: 10.1109/TCAD.1983.1270048
25. Choi V. Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. *Quantum Inf Process.* (2008) **7**:193–209. doi: 10.1007/s11128-008-0082-9
26. Adler I, Dorn F, Fomin FV, Sau I, Thilikos DM. Faster parameterized algorithms for minor containment. *Theor Comput Sci.* (2011) **412**:7018–28. doi: 10.1016/j.tcs.2011.09.015
27. Robertson N, Seymour PD. Graph minors XIII: the disjoint paths problem. *J Comb Theory Ser B* (1995) **63**:65–110. doi: 10.1006/jctb.1995.1006
28. Klymko C, Sullivan BD, Humble TS. Adiabatic quantum programming: minor embedding with hard faults. *Quantum Inf Process.* (2014) **13**:709–29. doi: 10.1007/s11128-013-0683-9
29. Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. *Bell Syst Tec J.* (1970) **49**:291–307. doi: 10.1002/j.1538-7305.1970.tb01770.x
30. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput.* (1998) **20**:359–92. doi: 10.1137/S1064827595287997
31. Kschischang FR, Frey BJ, Loeliger HA. Factor graphs and the sum-product algorithm. *IEEE Trans Inf Theor.* (2006) **47**:498–519. doi: 10.1109/18.910572
32. Mceliece RJ, Mackay DJC, Cheng J. Turbo decoding as an instance of Pearls belief propagation algorithm. *IEEE J Select Areas Commun.* (1998) **16**:140–52. doi: 10.1109/49.661103
33. Bertsekas DP. *Nonlinear Programming*. Nashua, NH: Athena Scientific (1995).
34. Wolsey LA. *Integer Programming*. A Wiley-Interscience publication (1998).
35. Aarts E, Lenstra JK, editors. *Local Search in Combinatorial Optimization*. New York, NY: John Wiley and Sons, Inc. (1997).
36. Ahuja RK, Ergun O, Orlin JB, Punnen AP. A survey of very large-scale neighborhood search techniques. *Dis Appl Math.* (2002) **123**:75–102. doi: 10.1016/S0166-218X(01)00338-9
37. Lanting T, Przybysz AJ, Smirnov AY, Spedalieri FM, Amin MH, Berkley AJ, et al. Entanglement in a quantum annealing processor. *Phys Rev X* (2014) **4**:021041. doi: 10.1103/PhysRevX.4.021041

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 26 June 2014; accepted: 01 September 2014; published online: 18 September 2014.

Citation: Bian Z, Chudak F, Israel R, Lackey B, Macready WG and Roy A (2014) Discrete optimization using quantum annealing on sparse Ising models. Front. Phys. ***2****:56. doi: 10.3389/fphy.2014.00056*

This article was submitted to Interdisciplinary Physics, a section of the journal Frontiers in Physics.