Solving SAT and MaxSAT with a Quantum Annealer: Foundations and a Preliminary Report

Zhengbing Bian[1], Fabian Chudak[1], William Macready[1], Aidan Roy[1](✉), Roberto Sebastiani[2], and Stefano Varotti[2]

[1] D-Wave Systems Inc., Burnaby, Canada
aroy@dwavesys.com
[2] DISI, University of Trento, Trento, Italy

Abstract. Quantum annealers (QA) are specialized quantum computers that minimize objective functions over discrete variables by physically exploiting quantum effects. Current QA platforms allow for the optimization of quadratic objectives defined over binary variables, that is, they solve quadratic unconstrained binary optimization (QUBO) problems. In the last decade, QA systems as implemented by D-Wave have scaled with Moore-like growth. Current architectures provide 2048 sparsely-connected qubits, and continued exponential growth is anticipated.

We explore the feasibility of such architectures for solving SAT and MaxSAT problems as QA systems scale. We develop techniques for effectively encoding SAT and MaxSAT into QUBO problems compatible with sparse QA architectures. We provide the theoretical foundations for this mapping, and present encoding techniques that combine offline Satisfiability and Optimization Modulo Theories with on-the-fly placement and routing. Preliminary empirical tests on a current generation 2048-qubit D-Wave system support the feasibility of the approach.

We provide details on our SMT model of the SAT-encoding problem in the hopes that further research may improve upon the scalability of this application of SMT technology. Further, these models generate hard SMT problems which may be useful as benchmarks for solvers.

1 Introduction

Quantum Annealing (QA) is a specialized form of computation that uses quantum mechanical effects to efficiently sample low-energy configurations of particular cost functions on binary variables. Currently, the largest QA system heuristically minimizes an Ising cost function given by

$$E(\underline{\mathbf{z}}) \stackrel{\text{def}}{=} \sum_{i \in V} h_i z_i + \sum_{(i,j) \in E} J_{ij} z_i z_j \tag{1}$$

$$\underset{\underline{\mathbf{z}} \in \{-1,1\}^{|V|}}{\operatorname{argmin}} \; E(\underline{\mathbf{z}}). \tag{2}$$

where $G = (V, E)$ is an undirected graph of allowed variable interactions. Ising models are equivalent to *Quadratic Unconstrained Binary Optimization* (QUBO)

C. Dixon and M. Finger (Eds.): FroCoS 2017, LNCS 10483, pp. 153–171, 2017.
DOI: 10.1007/978-3-319-66167-4_9

problems, which use $\{0, 1\}$-valued variables rather than ±1-valued variables. [1] The decision version of the Ising problem on most graphs G is NP-complete.

Theory suggests that quantum annealing may solve some optimization problems faster than state-of-the-art algorithms [18]. Quantum effects such as tunneling and superposition provide QA with novel mechanisms for escaping local minima, thereby potentially avoiding suboptimal solutions commonly found by classical algorithms based on bit-flip operations (such as WalkSAT). Practical QA systems are not guaranteed to return optimal solutions; however, the D-Wave processor has been shown to outperform a range of classical algorithms on certain problems designed to match its hardware structure [16,21]. These results also provide guidance about the kinds of energy landscapes on which QA is expected to perform well.

Our ultimate goal is to exploit QA as an engine for solving SAT and other NP-hard problem instances which are relatively small but hard enough to be out of the reach of state-of-the-art solvers (e.g., SAT problems coming from cryptanalysis). *SAT* is the problem of deciding the satisfiability of arbitrary formulas on atomic propositions, typically written in conjunctive normal form. *MaxSAT* is an optimization extension of SAT, in which each clause is given a positive penalty if the clause is not satisfied, and an assignment minimizing the sum of the penalties is sought.

In principle, converting SAT to optimization of an Ising cost function is straightforward. However, practical QA systems such as the D-Wave 2000Q offer sparse connectivity between variables. The connectivity graph G of current D-Wave processors is shown in Fig. 1, and is called the *Chimera* graph. Further, because the Ising model is solved on a physical, analog device, it is subject to engineering limitations. The D-Wave 2000Q system currently requires $h_i \in [-2, 2]$ and $J_{ij} \in [-1, 1]$ and there are limits on the precision to which these parameters may be specified. Parameter imprecisions act as small additive noise sources on parameter values, and arise from operating quantum mechanical systems in real-world environments. These real-world practicalities necessitate a carefully defined SAT-to-Ising encoding.

These practical constraints generate a challenging problem because the SAT encoding must be done both *effectively* (i.e., in a way that uses only the limited number of qubits and connections available within the QA architecture, while optimizing performance of the QA algorithm), and *efficiently* (i.e., using a limited computational budget for computing the encoding). In this paper, we formalize this problem and provide practical algorithms.

A direct formulation of the encoding problem results in a large system of linear inequalities over continuous- and Boolean-valued variables. This system can be effectively addressed with Satisfiability or Optimization Modulo Theory (SMT/OMT) [3,28] solvers. *Satisfiability Modulo the Theory of Linear Rational Arithmetic (SMT($\mathcal{LRA}$))* [3] is the problem of deciding the satisfiability of arbitrary formulas on atomic propositions and constraints in linear arithmetic over the rationals. *Optimization Modulo the Theory of Linear Rational Arithmetic*

[1] The transformation between $z_i \in \{-1, 1\}$ and $x_i \in \{0, 1\}$ is $z_i = 2x_i - 1$.

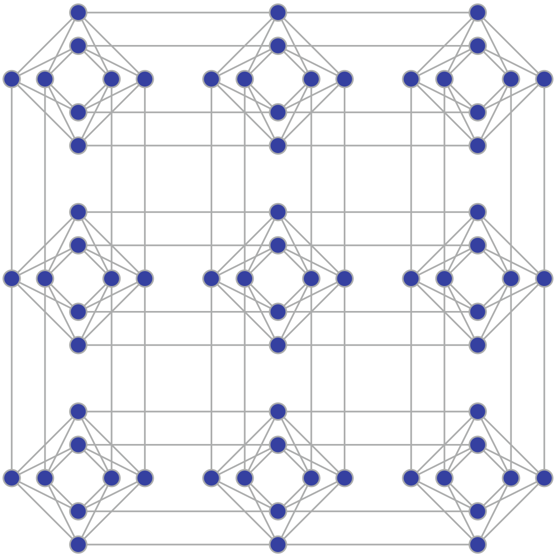


Fig. 1. Example of the Chimera topology: the hardware graph for system of 72 qubits in a 3-by-3 grid of tiles. (D-Wave 2000Q systems have 2048 qubits in a 16-by-16 grid.) This topology consists of a lattice of strongly-connected components of 8 qubits, called *tiles*. Each tile consists of a complete bipartite graph between two sets of four qubits. One set, the “vertical” set, is connected to the tiles above and below; the other set, the “horizontal” set, is connected to the tiles to the left and to the right. Notice that each qubit is connected with at most six other qubits. In other words, each variable z_i in the Ising model (1) has at most 6 non-zero J_{ij} interactions with other variables.

(OMT($\mathcal{LRA}$)) [28] extends SMT($\mathcal{LRA}$) by searching solutions which optimize some $\mathcal{LRA}$ objective(s). Efficient OMT($\mathcal{LRA}$) solvers like OptiMathSAT [29] allow for handling formulas with thousands of Boolean and rational variables [28].

This monolithic linear programming approach to encoding typically requires the introduction of additional ancillary Boolean variables, and the resultant SMT/OMT problem may be computationally harder than the original problem. In contrast, a large Boolean formula can be scalably converted into an Ising model by decomposing it into subformulae, converting each subformula into an Ising model (perhaps with introduction of additional fresh variables), and linking variables from different subformulae. Unfortunately, in practice this decomposition-based approach requires many auxiliary variables and connections, which are incompatible with the sparse connectivity restrictions imposed by QA architectures.

To cope with these difficulties, we propose a mixed approach, which combines (i) novel SMT/OMT-based techniques to produce off-line encodings of commonly-used Boolean subfunctions, with (ii) the usage of function instantiation and placement-and-routing techniques to combine and place on-the-fly the encoded functionalities within the QA architecture.

We have implemented prototype encoders on top of the SMT/OMT tool OptiMathSAT [29]. As a proof of concept, we present some preliminary empirical evaluation, in which we have executed encoded SAT and MaxSAT problems on a D-Wave 2000Q system. Although preliminary, the results confirm the feasibility of the approach. We stress the fact that this paper does not present a comparison with respect to state-of-the-art of classic computing. Rather, this is intended as a preliminary assessment of the challenges and potential of QA to impact SAT and MaxSAT solving.

The rest of the paper is organized as follows. Section 2 presents the theoretical foundations of this work; Sect. 3 describes our mixed approach to cope with this problem; Sect. 4 presents a preliminary empirical evaluation; Sect. 5 hints future developments. A longer and more detailed version of this paper, including a section that describes related work, is available online [7].

2 Foundations

Let $F(\underline{\mathbf{x}})$ be a Boolean function on a set of n *input Boolean variables* $\underline{\mathbf{x}} \overset{\text{def}}{=} \{x_1, ..., x_n\}$. We represent Boolean value $\bot$ with -1 and $\top$ with $+1$, so that we can assume that each $x_i \in \{-1, 1\}$. Suppose first that we have a QA system with n qubits defined on a hardware graph $G = (V, E)$, e.g., G can be any n-vertex subgraph of the Chimera graph of Fig. 1. Furthermore, assume that the state of each qubit z_i corresponds to the value of variable x_i, $i = 1, \ldots, n = |V|$. One way to determine whether $F(\underline{\mathbf{x}})$ is satisfiable using the QA system is to find an energy function as in (1) whose ground states $\underline{\mathbf{z}}$ correspond with the satisfiable assignments $\underline{\mathbf{x}}$ of $F(\underline{\mathbf{x}})$. For instance, if $F(\underline{\mathbf{x}}) \overset{\text{def}}{=} x_1 \oplus x_2$, since $F(\underline{\mathbf{x}}) = \top$ if and only if $x_1 + x_2 = 0$, the Ising model $(z_1 + z_2)^2$ in a graph containing 2 qubits joined by an edge has ground states $(+1, -1)$ and $(-1, +1)$, that is, the satisfiable assignments of F.

Because the energy $E(\underline{\mathbf{z}})$ in (1) is restricted to quadratic terms and graph G is typically sparse, the number of functions $F(\underline{\mathbf{x}})$ that can be solved with this approach is limited. To deal with this difficulty, we can use a larger QA system with a number of additional qubits, say h, representing *ancillary Boolean variables* (or *ancillas* for short) $\underline{\mathbf{a}} \overset{\text{def}}{=} \{a_1, ..., a_h\}$, so that $|V| = n + h$. A *variable placement* is a mapping of the $n + h$ input and ancillary variables into the qubits of V. Since G is not a complete graph, different variable placements will produce energy functions with different properties. We use *Ising encoding* to refer to the h_i and J_{ij} parameters in (1) that are provided to the QA hardware together with a variable placement of the variables. The *gap* of an Ising encoding is the energy difference between ground states (i.e., satisfiable assignments) and any other states (i.e., unsatisfiable assignments). An important observation from [5] is that the larger the gap the better the success rates of the QA process. The *encoding problem* for $F(\underline{\mathbf{x}})$ is to find an Ising encoding with maximum gap.

The encoding problem is typically over-constrained. In fact, the Ising model (1) has to discriminate between m satisfiable assignments and k unsatisfiable assignments, with $m + k = 2^n$, whereas the number of degrees of freedom is given by the number of the h_i and J_{ij} parameters, which in the Chimera architecture grows as $O(n + h)$.

In this section, we assume that a Boolean function $F(\underline{\mathbf{x}})$ is given and that h qubits are used for ancillary variables $\underline{\mathbf{a}}$.

2.1 Penalty Functions

Here we assume that a variable placement is given, placing $\underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ into the subgraph G. Thus, we can identify each variable z_j representing the binary value of the qubit associated with the jth vertex in V with either an x_k or a_ℓ variable, writing $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$. Then we define *penalty function* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ as the Ising model:

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \overset{\text{def}}{=} \theta_0 + \sum_{i \in V} \theta_i z_i + \sum_{(i,j) \in E} \theta_{ij} z_i z_j, \tag{3}$$

$$\text{with the property that} \quad \forall \underline{\mathbf{x}} \;\; min_{\{\underline{\mathbf{a}}\}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \begin{cases} = 0 & \text{if } F(\underline{\mathbf{x}}) = \top \\ \geq g_{min} & \text{if } F(\underline{\mathbf{x}}) = \bot \end{cases} \tag{4}$$

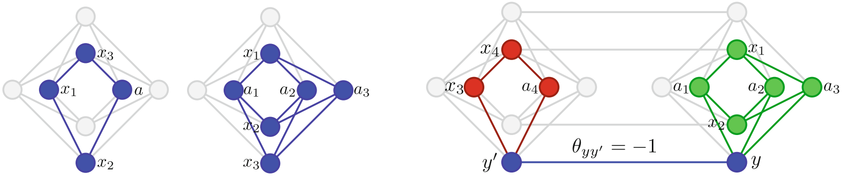
where $\theta_0 \in (-\infty, +\infty)$ ("*offset*"), $\theta_i \in [-2, 2]$ ("*biases*") and $\theta_{ij} \in [-1, 1]$ ("*couplings*"), s.t. $z_i, z_j \in \underline{\mathbf{z}}$, and $g_{min} > 0$ ("*gap*") are rational-valued parameters. Notice that a penalty function separates models from counter-models by an energy gap of at least g_{min}. We call $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ an *exact penalty function* iff it verifies a stronger version of (4) in which the condition "$\geq g_{min}$" is substituted with "$= g_{min}$". To simplify the notation we will assume that $\theta_{ij} = 0$ when $(i, j) \notin E$, and use $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ when $\underline{\mathbf{a}} = \emptyset$.

The QA hardware is used to minimize the Ising model defined by penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. By (4), a returned value of $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0$ implies that F is satisfiable. However, if $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}$, since QA does not guarantee optimality, there is still a chance that F is satisfiable. Nevertheless, the larger g_{min} is, the less likely this false negative case occurs.

The following examples show that ancillary variables are needed, even when G is a complete graph.

Example 1. The equivalence between two variables, $F(\underline{\mathbf{x}}) \overset{\text{def}}{=} (x_1 \leftrightarrow x_2)$, can be encoded without ancillas by means of a single coupling between two connected vertices, with zero biases: $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) \overset{\text{def}}{=} 1 - x_1 x_2$, so that $g_{min} = 2$. In fact, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 0$ if x_1, x_2 have the same value; $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 2$ otherwise. Notice that $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ is also an exact penalty function. Penalty $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ is called a *chain* of length 2.

Example 2. Consider the AND function $F(\underline{\mathbf{x}}) \overset{\text{def}}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$. If $x_1,\ x_2,\ x_3$ could be all connected in a 3-clique, then $F(\underline{\mathbf{x}})$ could be encoded without ancillas by setting $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = \frac{3}{2} - \frac{1}{2}x_1 - \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_1 x_2 - x_1 x_3 - x_2 x_3$, so that $g_{min} = 2$. Since the Chimera graph has no cliques, so that the above AND function needs



(a) $x_3 \leftrightarrow (x_1 \wedge x_2)$ with one ancilla.

(b) $x_3 \leftrightarrow (x_1 \oplus x_2)$ with three ancillas.

(c) $x_4 \leftrightarrow (x_3 \wedge (x_1 \oplus x_2))$ obtained by combining 2(b) and 2(a).

Fig. 2. Mappings within the Chimera graph, penalty functions use only colored edges. (c) combines (a) and (b) using chained proxy variables y, y'. The resulting penalty function is obtained by rewriting $x_4 \leftrightarrow (x_3 \wedge (x_1 \oplus x_2))$ into its equi-satisfiable formula $(x_4 \leftrightarrow (x_3 \wedge y')) \wedge (y' \leftrightarrow y) \wedge (y \leftrightarrow (x_1 \oplus x_2))$. (Color figure online)

(at least) one ancilla a to be encoded as: $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = \frac{5}{2} - \frac{1}{2}x_1 - \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_1x_2 - x_1x_3 - x_2a - x_3a$, which still has gap $g_{min} = 2$ and is embedded as in Fig. 2(a).

Example 3. Consider the XOR function $F(\underline{\mathbf{x}}) \overset{\text{def}}{=} x_3 \leftrightarrow (x_1 \oplus x_2)$. Even within a 3-clique, $F(\underline{\mathbf{x}})$ has no ancilla-free encoding. Within the Chimera graph, $F(\underline{\mathbf{x}})$ can be encoded with three ancillas a_1, a_2, a_3 as: $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 5 + x_3 + a_2 - a_3 + x_1a_1 - x_1a_2 - x_1a_3 - x_2a_1 - x_2a_2 - x_2a_3 + x_3a_2 - x_3a_3$, which has gap $g_{min} = 2$ and is embedded as in Fig. 2(b).

2.2 Properties of Penalty Functions and Problem Decomposition

After determining a variable placement, finding the values for the θs implicitly requires solving a set of equations whose size grows with the number of models of $F(\underline{\mathbf{x}})$ plus a number of inequalities whose size grows with the number of counter-models of $F(\underline{\mathbf{x}})$. Thus, the θs must satisfy a number of linear constraints that grows exponentially in n. Since the θs grow approximately as $4(n+h)$, the number of ancillary variables needed to satisfy (4) can also grow very rapidly. This seriously limits the scalability of a solution method based on (3)–(4). We address this issue by showing how to construct penalty functions by combining smaller penalty functions, albeit at the expense of a reduced gap.

The following two properties can be easily derived from the definition.

Property 1. Let $F^*(\underline{\mathbf{x}}) \overset{\text{def}}{=} F(x_1, ..., x_{r-1}, \neg x_r, x_{r+1}, ..., x_n)$ for some index r. Assume a variable placement of $\underline{\mathbf{x}}$ into V s.t. $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is a penalty function for $F(\underline{\mathbf{x}})$ of gap g_{min}. Then $P_{F^*}(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}^*)$, where $\underline{\boldsymbol{\theta}}^*$ is defined as follows for every $z_i, z_j \in \underline{\mathbf{x}}, \underline{\mathbf{a}}$:

$$\theta^*_i = \begin{cases} -\theta_i & \text{if } z_i = x_r \\ \theta_i & \text{otherwise;} \end{cases} \qquad \theta^*_{ij} = \begin{cases} -\theta_{ij} & \text{if } z_i = x_r \text{ or } z_j = x_r \\ \theta_{ij} & \text{otherwise.} \end{cases}$$

Notice that since the previously defined bounds over $\underline{\boldsymbol{\theta}}$ (namely $\theta_i \in [-2, 2]$ and $\theta_{ij} \in [-1, 1]$) are symmetric, if $\underline{\boldsymbol{\theta}}$ is in range then $\underline{\boldsymbol{\theta}}^*$ is as well.

Two Boolean functions that become equivalent by permuting or negating some of their variables are called *NPN-equivalent* [14]. Thus, given the penalty function for a Boolean formula, any other NPN equivalent formula can be encoded trivially by applying Property 1. Notice that checking NPN equivalence is a hard problem in theory, but it is fast in practice for small n (i.e., less than 16 [20]).

Property 2. Let $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ be Boolean formula such that $\underline{\mathbf{x}} = \cup_k \underline{\mathbf{x}}^k$, the $\underline{\mathbf{x}}^k$s may be non-disjoint, and each sub-formula F_k has a penalty function $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ with minimum gap g_{min}^k where $\underline{\mathbf{a}} = \cup_k \underline{\mathbf{a}}^k$ and the $\underline{\mathbf{a}}^k$s are all disjoint. Given a list w_k of positive rational values such that, for every $z_i, z_j \in \underline{\mathbf{x}} \cup \bigcup_{k=1}^{K} \underline{\mathbf{a}}^k$:

$$\theta_i \stackrel{\text{def}}{=} \sum_{k=1}^{K} w_k \theta_i^k \in [-2, 2], \quad \theta_{ij} \stackrel{\text{def}}{=} \sum_{k=1}^{K} w_k \theta_{ij}^k \in [-1, 1], \tag{5}$$

then a penalty function for $F(\underline{\mathbf{x}})$ can be obtained as:

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}^1 ... \underline{\mathbf{a}}^K|\underline{\boldsymbol{\theta}}) = \sum_{k=1}^{K} w_k P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k). \tag{6}$$

This new penalty function evaluates to zero if and only if all its summands do, and otherwise it is at least $g_{min} = \min_{k=1}^{K} w_k g_{min}^k$. Thus, in general, the (weighted) sum of the penalty functions of a set of formulas represents a penalty function for the conjunction of the formulas.

A formula $F(\underline{\mathbf{x}})$ can be decomposed (e.g., by a Tseitin transformation) into an equivalently-satisfiable one $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$:

$$F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}}) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m-1} (y_i \leftrightarrow F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i)) \wedge F_m(\underline{\mathbf{x}}^m, \underline{\mathbf{y}}^m), \tag{7}$$

where the F_is are Boolean functions which decompose the original formula $F(\underline{\mathbf{x}})$, and the y_is are fresh Boolean variables each labeling the corresponding F_i. By Property 2, this allows us to decompose $F(\underline{\mathbf{x}})$ into multiple $F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i)$ that can be encoded separately and recombined. The problem is to choose Boolean functions $F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i)$ whose penalty functions are easy to compute, have a large enough gap, and whose combination keeps the gap of the penalty function for the original function as large as possible.

Summing penalty functions with shared variables may cause problems with parameter ranges: penalty functions that share terms may sum up biases or couplings resulting in out-of-range values. Using weights, Property 2 can help to mitigate this, but also it is likely that the g_{min} of the final penalty function becomes small.

We can cope with this problem by mapping shared variables into distinct qubits and then linking them together. Consider again $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ as in Property 2. We rewrite it into its equi-satisfiable formula

$$F^*(\underline{\mathbf{x}}^*) \overset{\text{def}}{=} \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^{k^*}) \wedge \bigwedge_{\substack{x_i \in \underline{\mathbf{x}}^k \cap \underline{\mathbf{x}}^{k'} \\ k,k' \in [1..K],\ k<k'}} ({x_i}^{k^*} \leftrightarrow {x_i}^{k'^*}) \tag{8}$$

where $\underline{\mathbf{x}}^* = \cup_k \underline{\mathbf{x}}^{k^*}$ and the $\underline{\mathbf{x}}^{k^*}$ are all disjoint. Also, as in Property 2, assume we have $P_{F_k}(\underline{\mathbf{x}}^{k^*}, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$ for each k with disjoint $\underline{\mathbf{a}}^k$. If there is an edge between every two copies of the same variable x_i, we can write a penalty function in the following way (using the penalty of Example 1):

$$P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}}) = \sum_{k=1}^{K} P_{F_k}(\underline{\mathbf{x}}^{k^*}, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k) + \sum_{\substack{x_i \in \underline{\mathbf{x}}^k \cap \underline{\mathbf{x}}^{k'} \\ k,k' \in [1..K],\ k<k}} (1 - {x_i}^{k^*} {x_i}^{k'^*}), \tag{9}$$

and the θs stay within valid range because the $\underline{\mathbf{x}}^{k^*}$s are all disjoint. Thus, we can represent a single variable x_i with a series of qubits connected by strong couplings $(1 - z_i z_i')$. Figure 2(c) illustrates a simple example. Two observations are at hand. First, the gap g_{min} of $P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}})$ is at least $\min(\min_{k=1}^{K} w_k g_{min}^k, 2)$, since each $(1 - z_i z_i')$ penalty has a gap of 2. Second, not all copies of x_i need to be directly adjacent to obtain this bound: it suffices to use the edges of a tree connecting all copies. More generally, that tree may contain additional qubits to facilitate connectedness. A tree connecting all the copies of a variable x_i is called a *chain* and is the subject of the next section.

2.3 Embedding into Chimera Architecture

The process of representing a single variable x_i by a collection of qubits connected in chains of strong couplings is known as *embedding*, in reference to the minor embedding problem of graph theory [12,13]. More precisely, suppose we have a penalty function based on graph G (so x_i and x_j are adjacent iff $\theta_{ij} \neq 0$) and a QA hardware graph H. A *minor embedding* of G in H is a function $\Phi : V_G \to 2^{V_H}$ such that:

- for each G-vertex x_i, the subgraph induced by $\Phi(x_i)$ is connected;
- for all distinct G-vertices x_i and x_j, $\Phi(x_i)$ and $\Phi(x_j)$ are disjoint;
- for each edge (x_i, x_j) in G, there is at least one edge between $\Phi(x_i)$ and $\Phi(x_j)$.

The image $\Phi(x_i)$ of a G-vertex is a chain, and the set of qubits in a chain are constrained to be equal using $(1 - z_i z_i')$ couplings as in Fig. 2(c).

Embedding generic graphs is a computationally difficult problem [2], although certain structured problem graphs may be easily embedded in the Chimera topology [8,34] and heuristic algorithms may also be used [9]. A reasonable goal in embedding is to minimize the sizes of the chains, as quantum annealing becomes less effective as more qubits are included in chains [22].

A different approach to use QA for finding models for F, *global embedding*, is based on first finding a penalty function on a complete graph G on $n+h$ variables, and secondly, embedding G into a hardware graph H using chains (e.g., using [8]). Following [5], global embeddings usually need fewer qubits than the methods presented in this paper; however, the final gap of the penalty function obtained in this way is generally smaller and difficult to compute exactly.

3 Solving the Encoding Problem

3.1 Encoding Small Boolean Functions

Computing Penalty Functions via SMT/OMT($\mathcal{LRA}$). Given $\underline{\mathbf{x}} \overset{\text{def}}{=} \{x_1, ..., x_n\}$, $\underline{\mathbf{a}} \overset{\text{def}}{=} \{a_1, ..., a_h\}$, $F(\underline{\mathbf{x}})$ as in Sect. 2.1, a variable placement in a Chimera subgraph s.t. $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$, and some gap $g_{min} > 0$, the problem of finding a penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ as in (3) reduces to solving the following SMT($\mathcal{LRA}$) problem:

$$\Phi(\underline{\boldsymbol{\theta}}) \overset{\text{def}}{=} \bigwedge_{z_i \in \underline{\mathbf{x}}, \underline{\mathbf{a}}} (-2 \le \theta_i) \wedge (\theta_i \le 2) \wedge \bigwedge_{\substack{z_i z_j \in \underline{\mathbf{x}}, \underline{\mathbf{a}} \\ i<j}} (-1 \le \theta_{ij}) \wedge (\theta_{ij} \le 1) \tag{10}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0) \tag{11}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \ge 0) \tag{12}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \ge g_{min}). \tag{13}$$

Consequently, the problem of finding the penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ that maximizes the gap g_{min} reduces to solving the OMT($\mathcal{LRA}$) maximization problem $\langle \Phi(\underline{\boldsymbol{\theta}}), g_{min} \rangle$.

Intuitively: (10) states the ranges of the $\underline{\boldsymbol{\theta}}$; (11) and (12) state that, for every $\underline{\mathbf{x}}$ satisfying $F(\underline{\mathbf{x}})$, $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ must be zero for at least one "minimum" $\underline{\mathbf{a}}$ and nonnegative for all the others; (13) states that for every $\underline{\mathbf{x}}$ not satisfying $F(\underline{\mathbf{x}})$, $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ must greater or equal than the gap. Consequently, if the values of the $\underline{\boldsymbol{\theta}}$ in $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ satisfy $\Phi(\underline{\boldsymbol{\theta}})$, then $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ complies with (4).

Notice that $\Phi(\underline{\boldsymbol{\theta}})$ grows exponentially with $|\underline{\mathbf{x}}| + |\underline{\mathbf{a}}|$, and no longer contains Boolean atoms. Notice also that, if $\underline{\mathbf{a}} = \emptyset$, the OMT($\mathcal{LRA}$) maximization problem $\langle \Phi(\underline{\boldsymbol{\theta}}), g_{min} \rangle$ reduces to a linear program because the disjunctions in (11) disappear.

To force $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to be an *exact* penalty function, we conjoin to $\Phi(\underline{\boldsymbol{\theta}})$ the following:

$$... \wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = g_{min}). \tag{14}$$

Here, (14) forces $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to be exactly equal to the gap for at least one "minimum" $\underline{\mathbf{a}}$. Exact penalty functions can be used to encode (weighted) MaxSAT instances. Suppose we partition a formula into the conjunction of its soft constraints C_i each of weight $w_i \geq 0$. Then for each C_i we find an *exact* penalty function of (10)–(14) for C_i imposing a gap g_i proportional to w_i, and we combine the result as in Property 2.

Improving Efficiency and Scalability Using Variable Elimination. As before, assume that the variable placement is fixed and consider the SMT/OMT($\mathcal{LRA}$) formulation (10)–(13). Notice the exponential dependency on the number of hidden variables h. For practical purposes, this typically implies a limit on h of about 10. Here, we describe an alternative formulation whose size dependence on h is $O(h2^{\mathbf{tw}})$, where $\mathbf{tw}$ is the treewidth of the subgraph of G spanned by the qubits corresponding to the ancillary variables, G_a. For the Chimera graph, even when h is as large as 32, $\mathbf{tw}$ is at most 8 and therefore still of tractable size.

The crux of the reformulation is based on the use of the variable elimination technique [15] to solve an Ising problem on G_a. This method is a form of dynamic programming, storing tables in memory describing all possible outcomes to the problem. When the treewidth is $\mathbf{tw}$, there is a variable elimination order guaranteeing that each table contains at most $O(2^{\mathbf{tw}})$ entries. Rather than using numerical tables, our formulation replaces each of its entries with a continuous variable constrained by linear inequalities. In principle, we need to parametrically solve an Ising problem for each $\underline{\mathbf{x}} \in \{-1, 1\}^n$, generating $O(2^n h 2^{\mathbf{tw}})$ continuous variables. However, by the sequential nature of the variable elimination process, many of these continuous variables are equal, leading to a reduced (as much as an order of magnitude smaller) and strengthened SMT formulation. See [5] for more details.

Placing Variables & Computing Penalty Functions via SMT/OMT($\mathcal{LRIA} \cup \mathcal{UF}$). The formula $\Phi(\underline{\boldsymbol{\theta}})$ in (10)–(14) can be built only after a variable placement, so that each variable $z_j \in \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ has been previously placed in some vertex $v_j \in V$. There are many such placements. For example, if $n + h = 8$ and we want to encode the penalty function into a 8-qubit Chimera tile, then we have $8! = 40320$ candidate placements. Exploiting symmetry and the automorphism group of G, one can show that most of these placements are equivalent.

Alternatively, we can combine the generation of the penalty function with an automatic variable placement by means of SMT/OMT($\mathcal{LRIA} \cup \mathcal{UF}$), $\mathcal{LRIA} \cup \mathcal{UF}$ being the combined theories of linear arithmetic over rationals and integers plus uninterpreted function symbols. This works as follows.

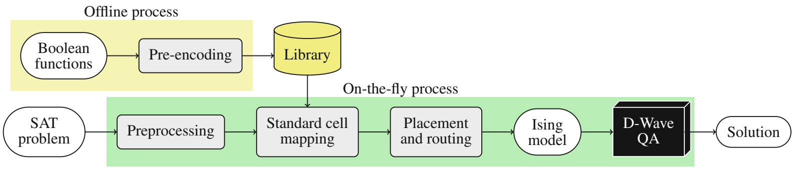


Fig. 3. Graph of the encoding process.

Suppose we want to produce the penalty function of some relatively small function (e.g., so $n + h \leq 8$, which fits into a single Chimera tile). We index the $n + h$ vertices in the set V into which we want to place the variables as $V \overset{\text{def}}{=} \{1, ..., n + h\}$, and we introduce a set of $n + h$ *integer* variables $\underline{\mathbf{v}} \overset{\text{def}}{=} \{v_1, ..., v_{n+h}\}$ s.t. each $v_j \in V$ represents (the index of) the vertex into which z_j is placed. (For example, "$v_3 = 5$" means that variable z_3 is placed in vertex #5.) Then we add the standard SMT constraint $\mathsf{Distinct}(v_1, ..., v_{n+h})$ to the formula to guarantee the injectivity of the map. Then, instead of using variables θ_i and θ_{ij} for biases and couplings, we introduce the *uninterpreted function symbols* $\mathsf{b} : V \longmapsto \mathbb{Q}$ ("bias") and $\mathsf{c} : V \times V \longmapsto \mathbb{Q}$ ("coupling"), so that we can rewrite each bias θ_j as $\mathsf{b}(v_j)$ and each coupling θ_{ij} as $\mathsf{c}(v_i, v_j)$ s.t $v_i, v_j \in [1, .., n + h]$ and $\mathsf{Distinct}(v_1, ..., v_{n+h})$.

This rewrites the SMT($\mathcal{LRA}$) problem (10)–(13) into the *SMT/OMT* ($\mathcal{LRIA} \cup \mathcal{UF}$) problem (15)–(26). Equation (19) must be used iff we need an exact penalty function. (Notice that (22) is necessary because we could have $\mathsf{c}(v_i, v_j)$ s.t. $v_i > v_j$.) By solving $\langle \Phi(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}), g_{min} \rangle$ we not only find the best values of the biases b and couplers c, but also the best placement $\underline{\mathbf{v}}$ of the variables into (the indexes of) the qubits.

3.2 Encoding Larger Boolean Functions

As pointed out in Sect. 2.2, encoding large Boolean functions using the SMT formulations of the previous section is computationally intractable, so other methods must be used. One sensible approach is to pre-compute a library of encoded Boolean functions and decompose a larger Boolean function $F(\underline{\mathbf{x}})$ into a set of pre-encoded ones $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$. The penalty models $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$ for these pre-encoded functions may then be combined using chains as described in Sect. 2.3. This schema is shown in Fig. 3. This is not the only possible method, but it is a natural choice for SAT and constraint satisfaction problems, and in terms of QA performance it has been shown experimentally to outperform other encoding methods for certain problem classes [6]. In this section, we describe each of the stages in turn.

$$\Phi(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \mathsf{Range}(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \wedge \mathsf{Distinct}(\underline{\mathbf{v}}) \wedge \mathsf{Graph}() \tag{15}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \quad \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \geq 0) \tag{16}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \quad \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) = 0) \tag{17}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \quad \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \geq g_{min}) \tag{18}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \quad \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) = g_{min}) \tag{19}$$

$$\mathsf{Range}(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \bigwedge_{1 \leq j \leq n+h} (1 \leq v_j) \wedge (v_j \leq n + h) \tag{20}$$

$$\wedge \bigwedge_{1 \leq j \leq n+h} (-2 \leq \mathsf{b}(j)) \wedge (\mathsf{b}(j) \leq 2) \tag{21}$$

$$\wedge \bigwedge_{1 \leq j \leq n+h} (\mathsf{c}(j, j) = 0) \wedge \bigwedge_{1 \leq i < j \leq n+h} (\mathsf{c}(i, j) = \mathsf{c}(j, i)) \tag{22}$$

$$\wedge \bigwedge_{1 \leq i < j \leq n+h} (-1 \leq \mathsf{c}(i, j)) \wedge (\mathsf{c}(i, j) \leq 1) \tag{23}$$

$$\mathsf{Distinct}(v_1, ..., v_{n+h}) \stackrel{\text{def}}{=} \bigwedge_{1 \leq i < j \leq n+h} \neg(v_i = v_j) \tag{24}$$

$$\mathsf{Graph}() \stackrel{\text{def}}{=} \wedge \bigwedge_{\substack{1 \leq i < j \leq n+h \\ \langle i, j \rangle \notin E}} (\mathsf{c}(i, j) = 0) \tag{25}$$

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \theta_0 + \sum_{1 \leq j \leq n+h} \mathsf{b}(v_j) \cdot z_j + \sum_{1 \leq i < j \leq n+h} \mathsf{c}(v_i, v_j) \cdot z_i \cdot z_j. \tag{26}$$

Pre-encoding. In this stage, we find effective encodings of common small Boolean functions, using the SMT methods in Sect. 3.1 or by other means, and store them in a library for later use. Finding these encodings may be computationally expensive, but this task may be performed offline ahead of time, as it is independent of the problem input, and it need only be performed once for each NPN-inequivalent Boolean function.

Preprocessing. Preprocessing, or Boolean formula minimization, consists of simplifying the input formula $F(\underline{\mathbf{x}})$ to reduce its size or complexity in terms of its graphical representation (typically and-inverter graphs). This is a well-studied problem with mature algorithms available [23,25].

Standard cell mapping. In the standard cell mapping phase, $F(\underline{\mathbf{x}})$ is decomposed into functions $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ that are available in the library. To minimize the size of the final Ising model, K should be as small as possible. For SAT or constraint satisfaction problems, this mapping may be performed naïvely: given a set of constraints $\{F_k(\underline{\mathbf{x}}^k)\}_{k=1}^{K}$ on the variables, each $F_k(\underline{\mathbf{x}}^k)$ is found in the library (possibly combining small constraints into larger ones [5]). However, more advanced techniques have been devised in the digital logic synthesis literature. For example, *technology mapping* is the process of mapping a technology-independent circuit representation to the physical gates used in a digital circuit [17,24]. Usually technology mapping is used to reduce circuit delay and load, and performs minimization as an additional step. Delay and load do not play a role in the context of QA, but minimization is important to simplify the placement and routing phase that follows.

Placement and routing. Once $F(\underline{\mathbf{x}})$ is decomposed into functions $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ with penalty models $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$, it remains to embed the entire formula onto the QA hardware as in equation (9). This process has two parts: *placement*, in which each $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ is assigned to a disjoint subgraph of the QA hardware graph; and *routing*, in which chains of qubits are built to ensure that distinct qubits x_i and x'_i representing the same variable take consistent values (using penalty functions of the form $1 - x_i x'_i$). Both placement and routing are very well-studied in design of digital circuits [4]. Nevertheless, this stage is a computational bottleneck for encoding large Boolean functions.

During placement, chain lengths can be minimized by placing penalty functions that share common variables close together. Heuristic methods for doing this include simulated annealing [31], continuous optimization [10], and recursive min-cut partitioning [27]. These algorithms can be applied in the present context, but require some modification as current QA architectures do not distinguish between qubits used for penalty functions and qubits used for chains.

During routing, literals are chained together using as few qubits possible. Finding an optimal routing is NP-hard, but polynomial-time approximation algorithms exist [19]. In practice, heuristic routing algorithms scale to problem sizes much larger than current QA architectures [11,26,33].

4 Preliminary Experimental Evaluation

In this section, we offer preliminary empirical validation of the proposed methods for encoding [Max]SAT by evaluating the performance of D-Wave's 2000Q system in solving certain hard SAT and MaxSAT problems.

Remark 1. To make the results reproducible to those who have access to a D-Wave system, we have set a website [1] where the problem files, translation files and demonstration code can be accessed. We also provide contact information for D-Wave 2000Q system access.

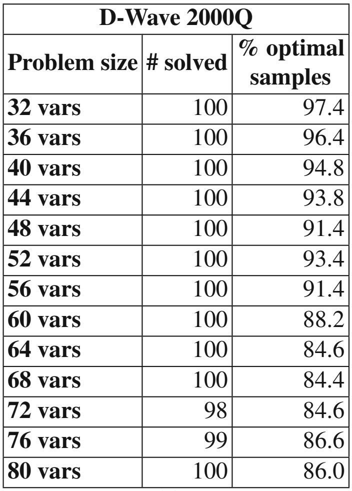
Due to the limitations in size and connectivity of current QA systems, we require [Max]SAT problems that become difficult with few variables. To this end we modified the tool sgen [30], which has been used to generate the smallest unsolvable problems in recent SAT competitions. In particular, we modified sgen to use 2-in-4-SAT constraints instead of at-most/at-least 1-in-5-SAT constraints, as 2-in-4-SAT is particularly suitable to encoding with Ising models (see [7] for details). We generated 100 problem instances for various problem sizes up to 80 variables, the largest embeddable with current hardware. At 260 variables, these problems become unsolvable within 1000 seconds with state-of-the-art SAT solvers on standard machines [7].

Another important consideration in solving [Max]SAT instances using QA is that the QA hardware cannot be made aware of the optimality of solution; for example, QA cannot terminate when all clauses in a SAT problem are satisfied. In this way, QA hardware behaves more like an SLS [Max]SAT solver than a CDCL-based SAT solver.

Propositional Satisfiability (SAT). To solve these SAT instances using QA, we encode and embed them as in Sect. 3 and then draw a fixed number of samples at an annealing rate of 10 µs per sample. Table 1(a) shows the results from the QA hardware. The QA hardware solves almost all problems within 50 µs of anneal time, and the rates of sampling optimal solutions remain relatively stable at this scale of problem.

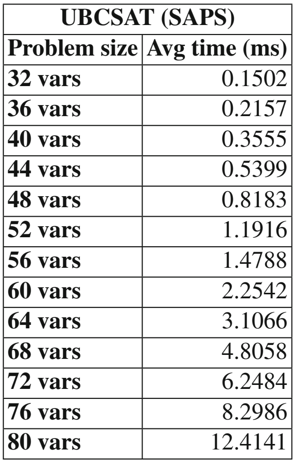
In order to evaluate the significance of the testbed, we solved the same problems with the UBCSAT SLS SAT solver using the best performing algorithm, namely SAPS [32]. Table 1(b) shows that the problems are nontrivial despite the small number of variables, and the run-times increase significantly with the size of the problem.

Table 1. (a) Number of problem instances (out of 100) solved by the QA hardware using 5 samples and average fraction of samples from the QA hardware that are optimal solutions. Annealing was executed at a rate of 10 µs per sample, for a total of 50 µs of anneal time per instance. Total time used by the D-Wave processor includes programming and readout; this amounts to about 150 µs per sample, plus a constant 10 ms of overhead. (b) Run-times in ms for SAT instances solved by UBCSAT using SAPS, averaged over 100 instances of each problem size. Computations were performed using an 8-core Intel® Xeon® E5-2407 CPU, at 2.20GHz.



D-Wave 2000Q		
Problem size	**# solved**	**% optimal samples**
32 vars	100	97.4
36 vars	100	96.4
40 vars	100	94.8
44 vars	100	93.8
48 vars	100	91.4
52 vars	100	93.4
56 vars	100	91.4
60 vars	100	88.2
64 vars	100	84.6
68 vars	100	84.4
72 vars	98	84.6
76 vars	99	86.6
80 vars	100	86.0

(a)



UBCSAT (SAPS)	
Problem size	**Avg time (ms)**
32 vars	0.1502
36 vars	0.2157
40 vars	0.3555
44 vars	0.5399
48 vars	0.8183
52 vars	1.1916
56 vars	1.4788
60 vars	2.2542
64 vars	3.1066
68 vars	4.8058
72 vars	6.2484
76 vars	8.2986
80 vars	12.4141

(b)

Table 2. (a) Number of problem instances (out of 100) solved by the QA hardware using 100 samples, and average fraction of samples from the QA hardware that are optimal solutions. Annealing was executed at a rate of 10 µs per sample, for a total of 1 ms of anneal time per instance. (b) Time in ms taken to find an optimal solution by various inexact weighted MaxSAT solvers, averaged over 100 MaxSAT instances of each problem size. Classical computations were performed on an Intel i7 2.90GHz × 4 processor. The solvers gw2sat, rots, and novelty are as implemented in UBCSAT [32]. All classical algorithms are performed with the optimal target weight specified; in the absence of a target weight they are much slower.

D-Wave 2000Q		
Problem size	**# solved**	**% optimal samples**
32 vars	100	78.7
36 vars	100	69
40 vars	100	60.2
44 vars	100	49.9
48 vars	100	40.4
52 vars	100	35.2
56 vars	100	24.3
60 vars	100	22.3
64 vars	99	17.6
68 vars	99	13
72 vars	98	9.6
76 vars	94	6.6
80 vars	93	4.3

(a)

MaxSAT solvers: avg time (ms)				
Problem size	**g2wsat**	**rots**	**maxwalksat**	**novelty**
32 vars	0.02	0.018	0.034	0.039
36 vars	0.025	0.022	0.043	0.06
40 vars	0.039	0.029	0.056	0.119
44 vars	0.049	0.043	0.07	0.187
48 vars	0.069	0.054	0.093	0.311
52 vars	0.122	0.075	0.115	0.687
56 vars	0.181	0.112	0.156	1.319
60 vars	0.261	0.13	0.167	1.884
64 vars	0.527	0.159	0.207	4.272
68 vars	0.652	0.21	0.27	8.739
72 vars	0.838	0.287	0.312	14.118
76 vars	1.223	0.382	0.396	18.916
80 vars	1.426	0.485	0.43	95.057

(b)

Remark 2. The results shown are not intended as a performance comparison between D-Wave's 2000Q system and UBCSAT. It is difficult to make a reasonable comparison for many reasons, including issues of specialized vs. off-the-shelf hardware, different timing mechanisms and timing granularities, and costs of encoding. Instead we aim to provide an empirical assessment of QA's potential for [Max]SAT solving, based on currently available systems.

Weighted MaxSAT sampling. One of the strengths of D-Wave's processor is its ability to rapidly sample the near-optimal solutions: current systems typically anneal at a rate of 10 µs or 20 µs per sample and are designed to take thousands of samples during each programming cycle. As a result, the first practical benefits of QA will likely come from applications which require many solutions rather than a single optimum. To demonstrate the performance of QA in this regime, we generated MaxSAT instances that have many distinct optimal solutions. These problems were generated from the 2-in-4-SAT instances described above by removing a fraction of the constraints and then adding constraints on single variables with smaller weight(details in [7]).

Table 2 summarizes the performance of the D-Wave processor in generating a single optimal MaxSAT solution, as well as the run-times for various high-

Table 3. Number of distinct optimal solutions found in 1 second by various MaxSAT solvers, averaged across 100 instances of each problem size. (a) "anneal only" accounts for only the 10 ms per sample anneal time used by the D-Wave processor. "wall-clock" accounts for all time used by the D-Wave processor, including programming and readout. (b) Classical computations were performed as in Table 2(b).

D-Wave 2000Q		
Size	**anneal only**	**wall-clock**
32 vars	448.5	443.9
36 vars	607	579.9
40 vars	1007.9	922
44 vars	1322.6	1066.6
48 vars	1555.4	1111.8
52 vars	3229	1512.5
56 vars	2418.9	1147.4
60 vars	4015.3	1359.3
64 vars	6692.6	1339.1
68 vars	6504.2	1097.1
72 vars	3707.6	731.7
76 vars	2490.3	474.2
80 vars	1439.4	332.7

(a)

MaxSAT solvers				
Size	**g2wsat**	**rots**	**maxwalksat**	**novelty**
32 vars	448.5	448.5	448.5	448.5
36 vars	607	606.9	606.9	606.8
40 vars	1007.7	1006.3	1005.3	1005
44 vars	1313.8	1307.1	1311.7	1255.5
48 vars	1515.4	1510.7	1504.9	1320.5
52 vars	2707.5	2813	2854.6	1616.2
56 vars	2021.9	2106.2	2186.6	969.8
60 vars	2845.6	3061.7	3289	904.4
64 vars	3100	4171	4770	570.6
68 vars	2742.2	3823.3	4592.4	354.8
72 vars	1841.1	2400.2	2943.4	212.6
76 vars	1262.5	1716	2059.2	116.4
80 vars	772.2	1111.1	1363.9	66.7

(b)

performing SLS MaxSAT solvers. The QA hardware solves almost all problems within 1 ms of anneal time. (Remark 2 also applies here.)

Table 3 considers generating distinct optimal solutions. For each solver and problem size, the table indicates the number of distinct solutions found in 1 s, averaged across 100 problem instances of that size. For the smallest problems, 1 s is sufficient for all solvers to generate all solutions, while the diversity of solutions found varies widely as problem size increases. Although the D-Wave processor returns a smaller fraction of optimal solutions for MaxSAT instances than for the SAT instances, it is still effective in enumerating distinct optimal solutions because its rapid sampling rate.

5 Ongoing and Future Work

Future QA architectures will be larger and more connected, enabling more efficient encodings of larger and more difficult SAT problems. Faster and more scalable SMT-based encoding methods for small Boolean functions is currently an important direction of research. The ability to increase the number of ancillary variables can lead to larger gaps, which in turn can make QA more reliable. Among the encoding challenges presented in this paper, a few are of particular interest and relevance to SMT research:

- *Variable placement.* Methods for simultaneously placing variables and computing penalty functions are currently less scalable, and have been less studied, than those for fixed variable placements.

- *Augmenting penalty models.* For large Boolean functions, generating penalty models directly from SMT becomes difficult because the number of constraints grows much more quickly than the number of available parameters. Function decomposition and chains provide one way around this, but chains limit the resulting energy gaps. There may be other methods of recombining a decomposed function that are not so restrictive. Alternatively, it may be possible to augment an existing penalty model with additional qubits for the purposes of increasing its energy gap. SMT formulations of these problems have not yet been explored.
- *Better function decompositions.* While Boolean function decomposition and minimization are mature classical subjects, those algorithms can probably be improved by taking into consideration the specifics of the embedding (placement and routing onto a QA hardware graph) that follow them.

Furthermore, we believe the problems presented here are not only practical, but also complex enough to be used to challenge new SMT solvers. To encourage the use of these problems as SMT benchmarks, we have provided example .smt files on the website of supplementary material [1].

References

1. Experimental data, source code, and supplementary material. https://bitbucket.org/aqcsat/frocos2017
2. Adler, I., Dorn, F., Fomin, F.V., Sau, I., Thilikos, D.M.: Faster parameterized algorithms for minor containment. Theor. Comput. Sci. **412**, 7018–7028 (2011)
3. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Handbook of Satisfiability (2009)
4. Betz, V., Rose, J.: Vpr: a new packing, placement and routing tool for FPGA research. In: Field Programmable Logic Workshop (1997)
5. Bian, Z., Chudak, F., Israel, R., Lackey, B., Macready, W.G., Roy A.: Discrete optimization using quantum annealing on sparse Ising models. Frontiers in Physics (2014)
6. Bian, Z., Chudak, F., Israel, R.B., Lackey, B., Macready, W.G., Roy, A.: Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. In: Frontiers in ICT (2016)
7. Bian, Z., Chudak, F., Macready, W., Roy, A., Sebastiani, R., Varotti, S.: Solving SAT and MaxSAT with a Quantum Annealer: Foundations and a Preliminary Report (2017). Extended version. https://bitbucket.org/aqcsat/frocos2017/raw/HEAD/sat2ising_extended.pdf
8. Boothby, T., King, A.D., Roy, A.: Fast clique minor generation in chimera qubit connectivity graphs. Quant. Inf. Proc. **15**, 495–508 (2016)
9. Cai, J., Macready, W.G., Roy, A.: A practical heuristic for finding graph minors. arXiv preprint (2014)
10. Chan, T., Cong, J., Kong, T., Shinnerl, J.: Multilevel optimization for large-scale circuit placement. In: ICCAD (2000)
11. Chen, H.Y., Hsu, C.H., Chang Y.W.: High-performance global routing with fast overflow reduction. In: ASPDAC, January 2009

12. Choi, V.: Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. Quant. Inf. Proc. **7**, 193–209 (2008)
13. Choi, V.: Minor-embedding in adiabatic quantum computation: II. minor-universal graph design. Quant. Inf. Proc. **10**, 343–353 (2011)
14. Correia, V.P., Reis A.I.: Classifying n-input boolean functions. In: VII Iberchip (2001)
15. Dechter, R.: Bucket elimination: a unifying framework for reasoning. In: Proceedings of the UAI (1996)
16. Denchev, V.S., Boixo, S., Isakov, S.V., Ding, N., Babbush, R., Smelyanskiy, V., Martinis, J., Neven, H.: What is the Computational Value of Finite-Range Tunneling? Ph.Rev.X (2016)
17. Een, N., Mishchenko, A., Sörensson, N.: Applying logic synthesis for speeding up SAT. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 272–286. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72788-0_26
18. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution. arXiv preprint (2000)
19. Gester, M., Müller, D., Nieberg, T., Panten, C., Schulte, C., Vygen, J.: Bonnroute: algorithms and data structures for fast and good VLSI routing. TODAES **18**, 32 (2013)
20. Huang, Z., Wang, L., Nasikovskiy, Y., Mishchenko, A.: Fast boolean matching based on NPN classification. In: ICFPT (2013)
21. King, J., Yarkoni, S., Raymond, J., Ozfidan, I., King, A.D., Nevisi, M.M., Hilton, J.P., McGeoch, C.C.: Quantum Annealing amid Local Ruggedness and Global Frustration. arXiv preprint (2017)
22. Lanting, T., Harris, R., Johansson, J., Amin, M.H.S., Berkley, A.J., Gildert, S., Johnson, M.W., Bunyk, P., Tolkacheva, E., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Chapple, E.M., Enderud, C., Rich, C., Wilson, B., Thom, M.C., Uchaikin, S., Rose, G.: Cotunneling in pairs of coupled flux qubits. Phys. Rev. B **82**, 060512 (2010)
23. Mishchenko, A., Chatterjee, S., Brayton, R.: Dag-aware aig rewriting: a fresh look at combinational logic synthesis. In: DAC (2006)
24. Mishchenko, A., Chatterjee, S., Brayton, R., Wang, X., Kam T.: Technology mapping with boolean matching, supergates and choices (2005)
25. Mishchenko, A., Chatterjee, S., Jiang, R., Brayton, R.K.: Fraigs: a unifying representation for logic synthesis and verification. Technical report (2005)
26. Roy, J.A., Markov, I.L.: High-performance routing at the nanometer scale. TCAD **27**, 1066–1077 (2008)
27. Roy, J.A., Papa, D.A., Adya, S.N., Chan, H.H., Ng, A.N., Lu, J.F., Markov, I.L.: Capo: robust and scalable open-source min-cut floorplacer. In: ISPD (2005)
28. Sebastiani, R., Tomasi, S.: Optimization modulo theories with linear rational costs. TOCL **16**, 12 (2015)
29. Sebastiani, R., Trentin, P.: OptiMathSAT: a tool for optimization modulo theories. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 447–454. Springer, Cham (2015). doi:10.1007/978-3-319-21690-4_27
30. Spence, I.: Sgen1: a generator of small but difficult satisfiability benchmarks. JEA **15**, 1–2 (2010)
31. Sun, W.-J., Sechen, C.: Efficient and effective placement for very large circuits. TCAD **14**, 349–359 (1995)

32. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 306–320. Springer, Heidelberg (2005). doi:10.1007/11527695_24
33. Xu, Y., Zhang, Y., Chu, C.: Fastroute 4.0: global router with efficient via minimization. In: ASPDAC (2009)
34. Zaribafiyan, A., Marchand, D.J.J., Rezaei, S.S.C.: Systematic and deterministic graph-minor embedding for cartesian products of graphs. CoRR (2016)