Information and Computation ••• (••••) ••••••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results

Zhengbing Bian [a,1], Fabian Chudak [a], William Macready [a,1], Aidan Roy [a,1], Roberto Sebastiani [b,*], Stefano Varotti [b]

[a] D-Wave Systems Inc., 3033 Beta Ave., Burnaby V5G 4M9, Canada
[b] DISI, University of Trento, via Sommarive 9, I-38123 Povo (TN), Italy

ARTICLE INFO

Article history:
Received 7 November 2018
Received in revised form 20 June 2020
Accepted 19 July 2020
Available online xxxx

Keywords:
Quantum annealing (QA)
Quadratic unconstrained binary optimization (QUBO)
Ising model
Satisfiability modulo theories
Optimization modulo theories
SAT
MaxSAT
Chimera graph

ABSTRACT

Quantum annealers (QAs) 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 (qubits), also known as Ising 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, together with increased connectivity.

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, with some limitations, MaxSAT– into Ising 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 for certain SAT and MaxSAT problems.

1. Motivations and goals

Quantum Computing (QC) promises significant computational speedups by exploiting the quantum-mechanical phenomena of *superposition*, *entanglement* and *tunneling*. QC relies on *quantum bits (qubits)*. As opposed to bits, qubits can be in a superposition state of 0 and 1.[2] Theoretically, quantum algorithms can outperform their classical counterparts. Examples of this are Shor's algorithm [1] for prime-number factoring and Grover's algorithm [2] for unstructured search. Once the technology is fully developed, it is expected that quantum computing will replace classical computing for some complex computational tasks.

However, despite large investment, the development of practical gate-model quantum computers is still in its infancy and current prototypes are limited to less than 20 qubits. An alternative approach to standard gate-model QC is *Quantum Annealing*, a form of computation that efficiently samples the low-energy configurations of a quantum system [3–5]. In

* Corresponding author.
E-mail address: roberto.sebastiani@unitn.it (R. Sebastiani).
[1] Present affiliation: Sanctuary AI, Vancouver, Canada.

[2] Superposition is perhaps the best-known and most surprising aspect of quantum physics (e.g. the famous Schrödinger's cat which is both dead and alive prior to observation).

https://doi.org/10.1016/j.ic.2020.104609
0890-5401/

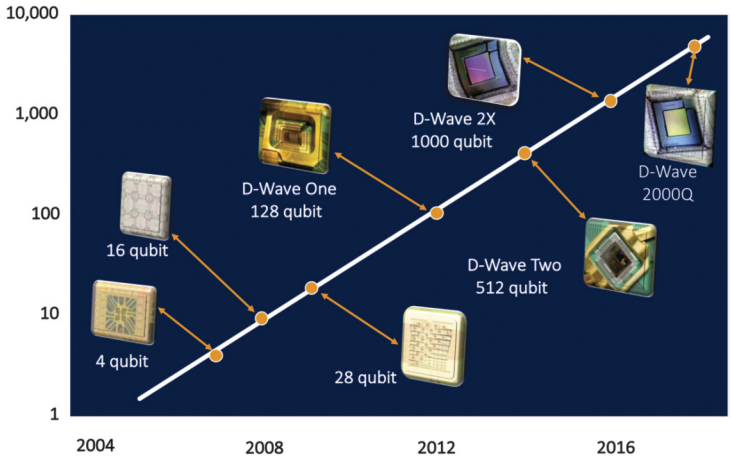


Fig. 1. Top: Moore-like progress diagram of the development of D-Wave's quantum annealers. X axis: year of release. Y axis: # of qubits. Notice the logarithmic scale of the Y axis. Bottom: The state-of-the-art D-Wave 2000Q quantum annealer. (Courtesy D-Wave Systems Inc.)

particular, D-Wave Systems Inc.[3] has developed special-purpose Quantum Annealers (QAs) which draw optima or near-optima from certain quadratic cost functions on binary variables. Since 2007, this approach has allowed D-Wave to improve QAs at a Moore-like rate, doubling the number of qubits roughly every 1.2 years, and reaching 2048 qubits in the state-of-the-art D-Wave 2000Q annealer in January 2017 (Fig. 1). These sophisticated devices are nearly-completely shielded from magnetic fields ($\leq 10^{-9}$ T) and are cooled to cryogenic temperatures (≤ 20 mK).

D-Wave's QAs can be used as specialized hardware for solving the *Ising problem*:

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

$$H(\underline{\mathbf{z}}) \stackrel{\text{def}}{=} \sum_{i\in V} \theta_i z_i + \sum_{(i,j)\in E} \theta_{ij} z_i z_j, \tag{2}$$

where each variable $z_i \in \{-1, 1\}$ is associated with a qubit; $G = \langle V, E\rangle$ is an undirected graph, *the hardware graph*, whose edges correspond to the physically allowed qubit interactions; and θ_i, θ_{ij} are programmable real-valued parameters. $H(\underline{\mathbf{z}})$ is known as the *Ising Hamiltonian* or *Ising model*. Ising problems are equivalent to *Quadratic Unconstrained Binary Optimization* (QUBO) problems, which use $\{0, 1\}$-valued variables rather than $\{-1, 1\}$-valued ones.[4] In current 2000Q systems, θ_i and θ_{ij} must be within the ranges $[-2, 2]$ and $[-1, 1]$ respectively, and G is a lattice of 16×16 8-qubit bipartite modules (*tiles*) known as the *Chimera* topology, shown in Figs. 2 and 3. The quadratic term in (2) is restricted to the edges of G, which is very sparse (vertices have degree at most 6). Despite this restriction, the Chimera Ising problem (1) is NP-hard [6].

Theory suggests that quantum annealing may solve certain optimization problems faster than state-of-the-art algorithms on classical computers [5]. Quantum effects such as tunneling and superposition provide QAs with novel mechanisms for escaping local minima, thereby potentially avoiding sub-optimal solutions commonly found by classical algorithms based on

[3] http://www.dwavesys.com.

[4] Ising variables z_i are related to QUBO variables x_i through $z_i = 2x_i - 1$.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

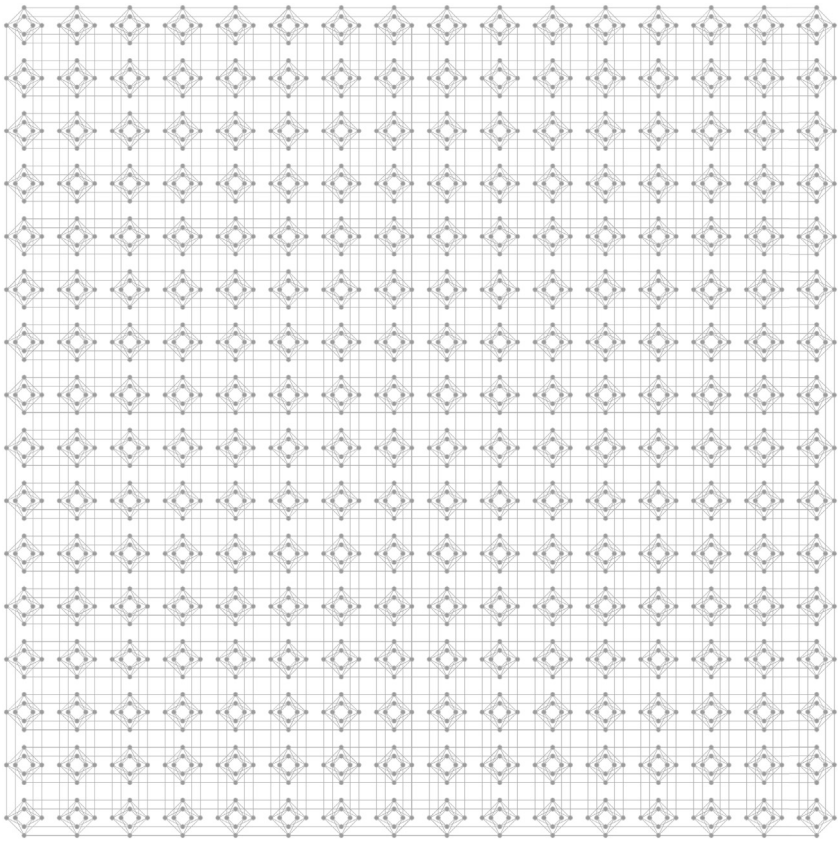


Fig. 2. The 2048-qubit connection graph of the D-Wave 2000Q quantum annealer architecture.

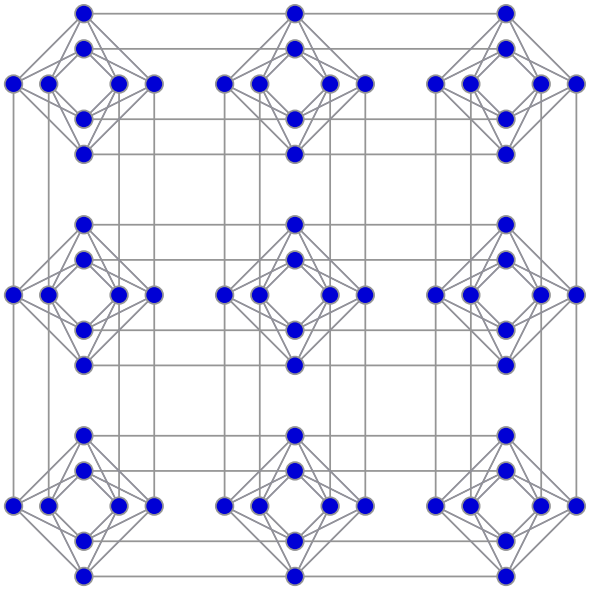


Fig. 3. Example of the Chimera topology: the hardware graph for system of 72 qubits in a 3-by-3 grid of 8-qubit tiles. (D-Wave 2000Q systems have 2048 qubits in a 16-by-16 grid.)

bit-flip operations (including WalkSAT, simulated annealing and others [7–9]). Although practical QA systems do not return optimal solutions with probability 1, the D-Wave processor has been shown to outperform a range of classical algorithms on certain problems designed to match its hardware structure [10,11]. This suggests the possible use of QAs to address hard

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

combinatorial decision/optimization problems, in particular NP-hard problems like SAT and MaxSAT [12,13], by encoding them into the Ising problem (1).

Our goal is to exploit quantum annealing as an engine for solving SAT, MaxSAT, and other NP-hard problems. Since current QAs have a limited number of qubits and connections, we target problem instances which are relatively small but computationally hard enough to be out of the reach of state-of-the-art classical solvers. Since QAs are not guaranteed to find an optimum and hence cannot certify the unsatisfiability of an encoded formula (§2.1), we target SAT problems such as cryptanalysis [14–16] or radio bandwidth repacking [17] which are surely or most-likely satisfiable, but whose solution is hard to find.

In this paper, we investigate the problem of encoding the satisfiability of an input Boolean formula $F(\underline{\mathbf{x}})$ into an Ising problem (1) from both theoretical and practical perspectives. In principle, converting SAT to Ising with an unbounded number of fully-connected qubits is straightforward. In practice, these encodings 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). We provide the necessary theoretical foundations, in which we analyze and formalize the problem and its properties. Based on this analysis, we then provide and implement practical encoding procedures. Finally, we empirically evaluate the effectiveness of these encodings on a D-Wave 2000Q quantum annealer.

We start from the observation that SATtoIsing can be formulated as a problem in Satisfiability or Optimization Modulo Theories (SMT/OMT) [18,19] on the theory of linear rational arithmetic, possibly enriched with uninterpreted function symbols. SATtoIsing is an intrinsically over-constrained problem, so a direct "monolithic" solution, encoding the whole input Boolean formula $F(\underline{\mathbf{x}})$ in one step, would typically require the introduction of many additional ancillary Boolean variables. These extra variables, in addition to wasting many qubits, would result in very large SMT/OMT formulas: solving the SATtoIsing via SMT would become computationally very hard, possibly even harder than the original SAT problem.

To cope with these issues, we adopt a scalable "divide-and-conquer" approach to SATtoIsing. First, we decompose the input Boolean formula into a conjunction of smaller subformulas. Then, we encode each subformula into an Ising model and place each subformula model into a disjoint subgraph of the hardware graph. Finally, we connect the qubits representing common variables from different subformulas using chains of qubits that are constrained to be logically identical.

To exploit the intrinsic modularity of the architecture graph (Figs. 2, and 3), we partition the input formula $F(\underline{\mathbf{x}})$ into subformulas which can be naturally encoded and placed into one or two adjacent 8-qubit tiles of the architecture, so that the encoding of each subformula is small enough to be handled efficiently by an SMT/OMT solver, and the encoded (sub)problems can be placed and interconnected within the modular structure of the graph. More concretely, we generate a library of encodings of commonly-used and relatively-small Boolean subfunctions. This library is only built once and consequently can use a large amount of computational resources. When presented with a SAT formula $F(\underline{\mathbf{x}})$, we decompose it, use the library to obtain encoded (sub)functions and use place-and-routing techniques to place and connect the encoded (sub)functions within the QA hardware graph.

We have implemented and made publicly available prototype encoders built on top of the SMT/OMT tool OptiMathSAT [20]. We present an empirical evaluation, in which we have run SATtoIsing-encoded problems and MaxSATtoIsing-encoded problems on a D-Wave 2000Q system. We have chosen input problems that are small enough to fit into the current architecture but are very hard with respect to their limited size, requiring some computational effort using a state-of-the-art solver.

We stress the fact that this evaluation is not meant to present a comparison with state-of-the-art of classic computing; rather, it is intended as a preliminary assessment of the challenges and potential of QAs to impact SAT and MaxSAT solving. This assessment is "preliminary" due to the limitations in number of qubits and qubit-connections of current QAs; however novel QAs currently under development at D-Wave have a more interconnected tile structure and higher per-qubit connectivity (degree 15 instead of 6, see also §8).[5]

Empirical evaluation shows that most encoded SAT problems are solved by the quantum annealer within negligible annealing time (≈ 10 μs). Although preliminary, the results confirm the feasibility of the approach. They also suggest that quantum annealers run on SATtoIsing-encoded problems (and to a lower extent, MaxSATtoIsing-encoded ones) might outperform standard algorithms on classical computers for certain difficult classes of relevant problems as soon as QA systems contain enough qubits and connections.

Content of the paper The rest of the paper is organized as follows: §2 presents necessary background on quantum annealing, SAT, MaxSAT, SMT and OMT; §3 presents the theoretical foundations of this work; §4 describes SMT/OMT-based encoding techniques for small Boolean formulas; §5 describes the process of encoding larger Boolean formulas by formula decomposition, encoding, placement and routing; §6 summarizes the related work; §7 presents preliminary empirical evaluation; §8 suggests future developments.

Disclaimer A preliminary and much shorter version of this paper was presented at the 11th International Symposium on Frontiers of Combining Systems, FroCoS'17 [21].

[5] See https://www.dwavesys.com/sites/default/files/mwj_dwave_qubits2018.pdf.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

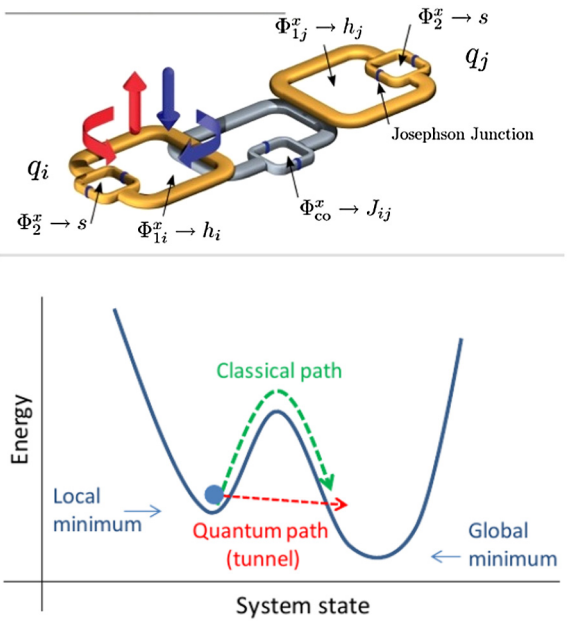


Fig. 4. Top: implementation of two coupled qubits. (Courtesy of D-Wave Systems Inc.) Bottom: graphical representation of the tunneling effect within an energy landscape.

2. Background

We provide the necessary background on quantum annealing (§2.1) SAT, MaxSAT, SMT and OMT (§2.2). For the readers' convenience, the list of the symbols we have used throughout the paper is summarized in Table 5 in the Appendix, with references to where each symbol is introduced.

2.1. Quantum annealing

As mentioned in §1, quantum annealers as currently implemented by D-Wave Systems are specialized chips that use quantum effects to sample or minimize energy configurations over binary variables (qubits) in the form of an Ising model (1) [6,22,23]. The qubits are interconnected in a grid of tightly connected groups of 8 qubits, called *tiles*, as displayed in Figs. 2 and 3. Each tile consists of a complete bipartite graph between two sets of four qubits: the "vertical" set, which is connected to the tiles above and below, and the "horizontal" set, which is connected to the tiles to the left and to the right. Each qubit is connected to at most six other qubits, so that each variable z_i occurs in at most 6 non-zero quadratic terms $\theta_{ij} z_i z_j$ (or $\theta_{ji} z_j z_i$). The graphs in Figs. 2 and 3 are known as *Chimera* graphs.

Single qubits z_i are implemented as inter-connected superconducting rings (Fig. 4, top), and a qubit's ± 1-value represents the direction of current in its ring. The user-programmable values $\theta_i \in [-2, 2]$ (*biases*) and $\theta_{ij} \in [-1, 1]$ (*couplings*) in (1) are real values within the specified interval, and are set by applying magnetic flux to the rings.[6] Overall, $H(\underline{\mathbf{z}})$ in (2) defines the energy landscape for a system of qubits whose global minima correspond to the solutions of problem (1).

During quantum annealing, the state of a qubit will be in a superposition of $+1$ and -1 simultaneously. The system of $|V|$ qubits is evolved from an initial Hamiltonian, whose lowest energy state is an equal superposition of all $2^{|V|}$ classical states, to a final, user-defined Hamiltonian as in (2). At the end of the annealing, the system is measured, and a single, classical state $\underline{\mathbf{z}} \in \{-1, 1\}^{|V|}$ is observed. In theory, if the evolution is sufficiently slow,[7] then the lowest energy state (the *ground state*) is maintained throughout. As a result, the final state $\underline{\mathbf{z}}$ is a solution to the Ising problem (1) (with some probability, see below). Unlike classical minimization techniques such as simulated annealing [8], the QA energy-minimization process can use *quantum tunneling* [24] to pass through tall, thin energy barriers, thereby avoiding trapping in certain classical local minima (Fig. 4, bottom).

QA theory shows that in the limit of arbitrarily low temperature, arbitrarily small noise, and arbitrarily slow annealing, the probability of obtaining a minimum energy solution converges to 1. In practice, these conditions cannot be achieved,

[6] We consider normalized bounds without units of measure and scale because the only relevant information for us is that both ranges are symmetric wrt. zero and that the bounds for the θ_is are twice as big as these for the θ_{ij}s in (2).

[7] Notice that here and elsewhere "slow" is intended in a quantum-physics sense, which is definitely not "slow" from a computer-science perspective: e.g., a complete annealing process on a D-Wave 2000Q annealer may typically take ≈ 10 µs.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

and minimum energy solutions are not guaranteed. Indeed, practical QA systems are physical, analog devices, subject to engineering limitations, and the optimal annealing rate is often determined empirically. Moreover, hardware performance is dramatically affected by the choice of Ising model. Among the most relevant factors are:

Thermal and electromagnetic noise. Despite cooling and shielding, thermal and electromagnetic noise still have noticeable effects. One (approximate) model of these effects is based on Boltzmann sampling, in which the probability of seeing a state $\underline{\mathbf{z}}$ with energy $H(\underline{\mathbf{z}})$ in (2) is proportional to $e^{-\beta H(\underline{\mathbf{z}})}$, with $\beta \in [3, 5]$ being observed for certain problem classes [25–27].

Intrinsic parameter errors. Due to engineering limitations and sources of environmental noise, the Ising model realized in QA hardware is not exactly the one programmed by the user. A simplified model of error is that each specified $\theta_i \in [-2, 2]$ and $\theta_{ij} \in [-1, 1]$ value is subject to additive Gaussian noise with standard deviation 0.03 and 0.02 respectively.

Freeze-out. Because of the limited connectivity, we often use chains of several interconnected qubits to represent a single Boolean variable (§3.4). However, the quantum tunneling effect on which quantum annealing is based is diminished for chains [24], thereby reducing the hardware's ability to find global minima. This effect can be mitigated by constructing Ising models with chains that are as small as possible.

Energy gaps. From the Boltzmann model, we see that a larger energy gap g_{min} between ground and excited states leads to a higher probability of an optimal solution, as a ground state is $e^{\beta g_{min}}$ times more likely than a first excited state. This suggests producing Ising models with large g_{min} in order to maximize the probability of obtaining an optimal solution.

The fact that QAs are not guaranteed to return a minimum-energy solution is partially addressed by taking a sequence of N samples from the same Ising model and selecting the result with smallest energy. Distinct samples are statistically independent, so the probability $P_{\mathsf{min}}[N]$ of obtaining at least one minimum solution over N samples converges exponentially to 1 with N:

$$P_{\mathsf{min}}[N] = 1 - (1 - P_{\mathsf{min}}[1])^N. \tag{3}$$

Typical annealing times and readout times are very short (≈ 10 µs and ≈ 120 µs respectively), and many samples can be drawn from the same Ising model within a single programming cycle, so is possible to obtain a large number of samples in reasonable time.

2.2. SAT, MaxSAT, SMT and OMT

We assume the reader is familiar with the basic syntax, semantics and properties of Boolean and first-order logic and theories. In the following we recall the main concepts of interest for our purposes, referring the reader to [12,28,13,18,19] for more details.

SAT & MaxSAT Given some finite set of Boolean variables $\underline{\mathbf{x}}$ (aka Boolean atoms) the language of Boolean logic ($\mathcal{B}$) is the set of formulas containing the atoms in $\underline{\mathbf{x}}$ and closed under the standard propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ (not, and, or, imply, iff, xor) with their usual meaning. A *literal* is an atom (positive literal) or its negation (negative literal). We implicitly remove double negations: e.g., if l is the negative literal $\neg x_i$, then by $\neg l$ we mean x_i rather than $\neg\neg x_i$. A *clause* is a disjunction of literals. A formula is in *conjunctive normal form (CNF)* iff it is written as a conjunction of clauses.

A truth value assignment $\underline{\mathbf{x}}$ *satisfies* $F(\underline{\mathbf{x}})$ iff it makes it evaluate to true. If so, $\underline{\mathbf{x}}$ is called a *model* for $F(\underline{\mathbf{x}})$. A formula $F(\underline{\mathbf{x}})$ is *satisfiable* iff at least one truth assignment satisfies it, *unsatisfiable* otherwise. $F(\underline{\mathbf{x}})$ is *valid* iff all truth assignments satisfy it. $F_1(\underline{\mathbf{x}}), F_2(\underline{\mathbf{x}})$ are *equivalent* iff they are satisfied by exactly the same truth assignments.

A formula $F(\underline{\mathbf{x}})$ which is not a conjunction can always be decomposed into a conjunction of smaller formulas $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ by means of Tseitin's transformation [29]:

$$F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}}) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m-1} (y_i \leftrightarrow F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i)) \wedge F_m(\underline{\mathbf{x}}^m, \underline{\mathbf{y}}^m), \tag{4}$$

where the F_is are formulas which decompose the original formula $F(\underline{\mathbf{x}})$, and the y_is are fresh Boolean variables each labeling the corresponding F_i. (If the input formula is itself a conjunction, then Tseitin's transformation can be applied recursively to each conjunct.) Tseitin's transformation (4) guarantees that $F(\underline{\mathbf{x}})$ is satisfiable if and only if $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ is satisfiable, and that if $\underline{\mathbf{x}}, \underline{\mathbf{y}}$ is a model for $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$, then $\underline{\mathbf{x}}$ is a model for $F(\underline{\mathbf{x}})$. To this extent, it is pervasively used also as a main recursive step for efficient CNF conversion of formulas [29].

A *quantified Boolean formula (QBF)* is defined inductively as follows: a Boolean formula is a QBF; if $F(\underline{\mathbf{x}})$ is a QBF, then $\forall x_i F(\underline{\mathbf{x}})$ and $\exists x_i F(\underline{\mathbf{x}})$ are QBFs. $\forall x_i F(\underline{\mathbf{x}})$ is equivalent to $(F(\underline{\mathbf{x}})_{x_i=\top} \wedge F(\underline{\mathbf{x}})_{x_i=\bot})$ and $\exists x_i F(\underline{\mathbf{x}})$ is equivalent to $(F(\underline{\mathbf{x}})_{x_i=\top} \vee F(\underline{\mathbf{x}})_{x_i=\bot})$ (aka *Shannon's expansion*).

Propositional Satisfiability (SAT) is the problem of establishing whether an input Boolean formula is satisfiable or not. SAT is NP-complete [30]. Efficient SAT solvers are publicly available, most notably those based on *Conflict-driven clause-learning*

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

(CDCL) [28] and on *stochastic local search* [31]. Most solvers require the input formula to be in CNF, implementing a CNF pre-conversion based on Tseitin's transformation (4) when this is not the case. See [12] for a survey of SAT-related problems and techniques.

Weighted *MaxSAT* $\{\langle F_k, c_k\rangle\}_k$ is an optimization extension of SAT, in which the input formula is a (typically unsatisfiable) conjunction of subformulas $F \stackrel{\text{def}}{=} \bigwedge_k F_k$ such that each conjunct F_k is given a positive penalty c_k if F_k is not satisfied, and an assignment minimizing the sum of the penalties is sought. (Often F is in CNF and the F_ks are single clauses or conjunctions of clauses.) *Partial Weighted MaxSAT* is an extension of Weighted MaxSAT in which some conjuncts, called *hard constraints*, have penalty $+\infty$. Efficient MaxSAT tools are publicly available (see, e.g., [13,9]).

SMT and OMT Satisfiability Modulo Theories (SMT) is the problem of checking the satisfiability of first order formulas in a background theory $\mathcal{T}$ (or combinations of theories thereof). We focus on the theories of interest for our purposes. Given $\underline{\mathbf{x}}$ as above and some finite set of rational-valued variables $\underline{\mathbf{v}}$, the language of the theory of *Linear Rational Arithmetic* ($\mathcal{LRA}$) extends that of Boolean logics with $\mathcal{LRA}$-atoms in the form $(\sum_i c_i v_i \bowtie c)$, c_i being rational values, $v_i \in \underline{\mathbf{v}}$ and $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$, with their usual meaning. In the theory of *linear rational-integer arithmetic with uninterpreted functions symbols* ($\mathcal{LRIA}\cup\mathcal{UF}$) the $\mathcal{LRA}$ language is extended by adding integer-valued variables to $\underline{\mathbf{v}}$ ($\mathcal{LRIA}$) and uninterpreted function symbols.[8] (E.g., $(x_i \rightarrow (3v_1 + f(2v_2) \leq f(v_3)))$ is a $\mathcal{LRIA}\cup\mathcal{UF}$ formula.) Notice that $\mathcal{B}$ is a sub-theory of $\mathcal{LRA}$ and $\mathcal{LRA}$ is a sub-theory of $\mathcal{LRIA}\cup\mathcal{UF}$. The notions of literal, assignment, clause and CNF, satisfiability, equivalence and validity, Tseitin's transformation and quantified formulas extend straightforwardly to $\mathcal{LRA}$ and $\mathcal{LRIA}\cup\mathcal{UF}$.

Satisfiability Modulo $\mathcal{LRIA}\cup\mathcal{UF}$ (SMT($\mathcal{LRIA}\cup\mathcal{UF}$)) [18] is the problem of deciding the satisfiability of arbitrary formulas on $\mathcal{LRIA}\cup\mathcal{UF}$ and its sub-theories. Efficient SMT($\mathcal{LRIA}\cup\mathcal{UF}$) solvers are available, including MathSAT5 [32].

Optimization Modulo $\mathcal{LRIA}\cup\mathcal{UF}$ (OMT ($\mathcal{LRIA}\cup\mathcal{UF}$)) [19] extends SMT($\mathcal{LRIA}\cup\mathcal{UF}$) searching solutions which optimize some $\mathcal{LRIA}$ objective(s). Efficient OMT($\mathcal{LRA}$) solvers like OptiMathSAT [33] are available.

3. Theoretical foundations

Let $F(\underline{\mathbf{x}})$ be a Boolean function on a set of n *Boolean variables* $\underline{\mathbf{x}} \stackrel{\text{def}}{=} \{x_1, ..., x_n\}$. 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 $G = (V, E)$, for instance, any n-vertex subgraph of the Chimera graph of Figs. 2 and 3. Furthermore, we 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 (2) whose ground states $\underline{\mathbf{z}}$ correspond to the satisfying assignments $\underline{\mathbf{x}}$ of $F(\underline{\mathbf{x}})$.

Example 1. Suppose $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_1 \oplus x_2$. Since $F(\underline{\mathbf{x}}) = \top$ if and only if $x_1 + x_2 = 0$, the Ising model $H(z_1, z_2) = z_1 \cdot z_2$ in a graph containing 2 qubits z_1, z_2 joined by an edge $(1, 2) \in E$ s.t. $\theta_{12} = 1$ has two ground states $(+1, -1)$ and $(-1, +1)$, which correspond to the satisfying assignments of F, and two excited states $(+1, +1)$ and $(-1, -1)$, corresponding to the non-satisfying ones.

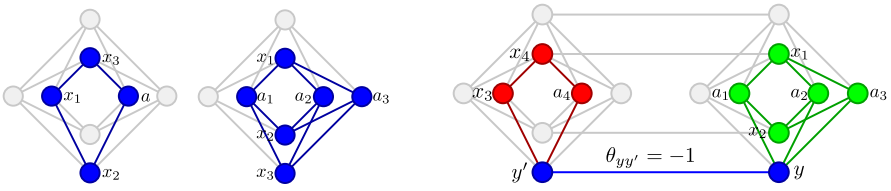
Because the energy $H(\underline{\mathbf{z}})$ in (2) is restricted to quadratic terms and graph G is typically sparse, the number of functions $F(\underline{\mathbf{x}})$ that can be solved with this approach is very limited. To deal in part with this difficulty, we can use a larger QA system with a number of additional qubits, say h, representing *ancillary Boolean variables* (or *ancillas* for short) $\underline{\mathbf{a}} \stackrel{\text{def}}{=} \{a_1, ..., a_h\}$, 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 θ_i and θ_{ij} parameters in (2) that are provided to the QA hardware together with a variable placement. The *gap* of an Ising encoding is the minimum energy difference between ground states (i.e., satisfying assignments) and the other states (i.e., non-satisfying assignments). In general, larger gaps lead to higher success rates in the QA process [34]. Thus, we define the *encoding problem* for $F(\underline{\mathbf{x}})$ as the problem of finding an Ising encoding with maximum gap.

Note that the encoding problem is typically over-constrained. The Ising model (2) has to discriminate between m satisfying assignments and k non-satisfying assignments, with $m + k = 2^n$, whereas the number of degrees of freedom is given by the number of the θ_i and θ_{ij} parameters, which grows as $O(n + h)$ in the Chimera architecture. Thus, in order to have a solution, the number of ancilla variables needed (h) may grow exponentially with the number of $\underline{\mathbf{x}}$ variables (n).

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

[8] An n-ary function symbol $f()$ is said to be *uninterpreted* if its interpretations have no constraint, except that of being a function (congruence): if $t_1 = s_1, ..., t_n = s_n$ then $f(t_1, ..., t_n) = f(s_1, ..., s_n)$.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609



(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 5(a) and 5(b).

Fig. 5. Mappings within the Chimera graph, penalty functions use only colored edges. 5(c) combines 5(a) and 5(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 $(x_4 \leftrightarrow (x_3 \wedge y')) \wedge (y \leftrightarrow (x_1 \oplus x_2)) \wedge (y' \leftrightarrow y)$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

3.1. 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 original variable $x_k \in \underline{\mathbf{x}}$ or as an ancilla variable $a_\ell \in \underline{\mathbf{a}}$, writing $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$.

Definition 1. *A* **penalty function** $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ *is an Ising model*

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \stackrel{def}{=} \theta_0 + \sum_{i \in V} \theta_i z_i + \sum_{(i,j) \in E} \theta_{ij} z_i z_j \tag{5}$$

with the property that for some $g_{min} > 0$,

$$\forall \underline{\mathbf{x}} \ min_{\{\underline{\mathbf{a}}\}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \begin{cases} = 0 & \text{if } F(\underline{\mathbf{x}}) = \top \\ \geq g_{min} & \text{if } F(\underline{\mathbf{x}}) = \bot \end{cases} \tag{6}$$

where $\theta_0 \in (-\infty, +\infty)$ *("*offset*"),* $\theta_i \in [-2, 2]$ *("*biases*") and* $\theta_{ij} \in [-1, 1]$ *("*couplers*") such that* $z_i, z_j \in \underline{\mathbf{z}}$*, and* g_{min} *are rational-valued parameters. The largest* g_{min} *such that* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ *satisfies* (6) *is called the* **gap** *of* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$.

Notice that a penalty function separates satisfying assignments from non-satisfying ones by a gap of at least g_{min}. The offset value θ_0 is added to set the value of $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to zero when $F(\underline{\mathbf{x}}) = \top$, so that $-\theta_0$ corresponds to the energy of the ground states of (2).

To simplify the notation we assume that $\theta_{ij} = 0$ when $(i, j) \notin E$, and use $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ when $\underline{\mathbf{a}} = \emptyset$.

Example 2. The equivalence between two variables, $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} (x_1 \leftrightarrow x_2)$, can be encoded without ancillas by means of a single coupling between two connected vertices, with zero biases: $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} 1 - x_1 x_2$, 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.

Penalty $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ in Example 2 is also called a (equivalence) *chain* connecting x_1, x_2, because it forces x_1, x_2 to have the same value.

The following examples show that ancillary variables are needed, even for small Boolean functions $F(\underline{\mathbf{x}})$ and even when G is a complete graph.

Example 3. Consider the AND function $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$. 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 $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = \frac{3}{2} - \frac{1}{2}x_1 - \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_1x_2 - x_1x_3 - x_2x_3$, so that $g_{min} = 2$. In fact, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 0$ if x_1, x_2, x_3 verify $F(\underline{\mathbf{x}})$, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 6$ if $x_1 = x_2 = -1$ and $x_3 = 1$, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 2$ otherwise. Since the Chimera graph has no cliques, the above AND function needs (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 can be embedded, e.g., as in Fig. 5(a).

Example 4. Consider the XOR function $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \oplus x_2)$. 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, e.g., as in Fig. 5(b).

The following fact is a straightforward consequence of Definition 1.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Proposition 1. *Let $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ be a penalty function of $F(\underline{\mathbf{x}})$ as in Definition 1. Then:*

- *If $\underline{\mathbf{x}}$,$\underline{\mathbf{a}}$ is such that $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0$, then $F(\underline{\mathbf{x}})$ is satisfiable and $\underline{\mathbf{x}}$ satisfies it.*
- *If $\underline{\mathbf{x}}$,$\underline{\mathbf{a}}$ minimizes $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ and $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}$, then $F(\underline{\mathbf{x}})$ is unsatisfiable.*

Proposition 1 shows that the QA hardware can be used as a satisfiability checker for $F(\underline{\mathbf{x}})$ by minimizing the Ising model defined by penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. A returned value of $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0$ implies that $F(\underline{\mathbf{x}})$ is satisfiable. If the QA hardware guaranteed minimality, then a returned value of $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}$ would imply that $F(\underline{\mathbf{x}})$ is unsatisfiable. However, since QAs do not guarantee minimality (§2.1), if $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}$ then there is still a chance that $F(\underline{\mathbf{x}})$ is satisfiable. Nevertheless, the larger g_{min} is, the less likely this false negative case occurs [34].

A penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is *normal* if $|\theta_i| = 2$ for at least one θ_i or $|\theta_{ij}| = 1$ for at least one θ_{ij}. In order to maximize g_{min}, it is important to use normal penalty functions to exploit the full range of the $\underline{\boldsymbol{\theta}}$ parameters. Any penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ can be normalized by multiplying all its coefficients by a *normalization factor*:

$$c \stackrel{\text{def}}{=} \min\left\{\min_i\left(\frac{2}{|\theta_i|}\right), \min_{\langle ij\rangle}\left(\frac{1}{|\theta_{ij}|}\right)\right\}. \tag{7}$$

Note that if $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is non-normal, then $c > 1$, so that the resulting gap $c \cdot g_{min} > g_{min}$. Normalization also works in the opposite direction to scale down some $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ whose θ's do not fit into the allowable ranges (in which case $c < 1$).

Hereafter we assume w.l.o.g. that all penalty functions are normal.

3.2. Properties of penalty functions and problem decomposition

As it will be made clear in §4.1, after a variable placement is set, finding the values for the θ's 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 θ's must satisfy a number of linear constraints that grows exponentially in n. Since the θ's grow approximately as $4(n+h)$, the number of ancillary variables needed to satisfy (6) can also grow very rapidly. This seriously limits the scalability of a solution method based on (5)-(6). We address this issue by showing how to construct penalty functions by combining smaller penalty functions, albeit at the expense of introducing extra variables.

The following properties are straightforward consequences of Definition 1.

Property 1. *Let $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ be a penalty function for $F(\underline{\mathbf{x}})$ and let $F^*(\underline{\mathbf{x}})$ be logically equivalent to $F(\underline{\mathbf{x}})$. Then $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is a penalty function also for $F^*(\underline{\mathbf{x}})$ with the same gap g_{min}.*

Property 1 states that a penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ does not depend on the syntactic structure of $F(\underline{\mathbf{x}})$ but only on its semantics.

Property 2. *Let $F^*(\underline{\mathbf{x}}) \stackrel{def}{=} F(x_1, \ldots, x_{r-1}, \neg x_r, x_{r+1}, \ldots, x_n)$ 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* [35]. Thus, given the penalty function for a Boolean formula, any other NPN equivalent formula can be encoded trivially by repeatedly applying Property 2. Notice that checking NPN equivalence is a hard problem in theory, but it is fast in practice for small n (i.e., $n \leq 16$) [36]. The process of negating a single variable in an Ising model as in Property 2 is known as a *spin-reversal transform*.

Example 5. Consider the OR function $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \vee x_2)$. We notice that this can be rewritten as $F(\underline{\mathbf{x}}) = \neg x_3 \leftrightarrow (\neg x_1 \wedge \neg x_2)$, that is, it is NPN-equivalent to that of Example 3. Thus, by Property 2 a penalty function for $F(\underline{\mathbf{x}})$ can be placed as in Fig. 5(a) and defined by taking that in Example 3 and toggling the signs of the coefficients of the x_i's: $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = \frac{5}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - x_3 + \frac{1}{2}x_1x_2 - x_1x_3 + x_2a + x_3a$, which still has gap $g_{min} = 2$.

Property 3. *Let $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ be a 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^k_{min} where the $\underline{\mathbf{a}}^k$s are all disjoint. Given a list w_k of positive rational values such that, for every $z_i, z_j \in \underline{\mathbf{x}} \cup \bigcup_{k=1}^{K} \underline{\mathbf{a}}^k$:*

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

$$\theta_i \stackrel{def}{=} \sum_{k=1}^{K} w_k \theta_i^k \in [-2, 2], \quad \theta_{ij} \stackrel{def}{=} \sum_{k=1}^{K} w_k \theta_{ij}^k \in [-1, 1], \tag{8}$$

then a penalty function for $F(\underline{\mathbf{x}})$ *is:*

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}^1 \ldots \underline{\mathbf{a}}^K | \underline{\boldsymbol{\theta}}) = \sum_{k=1}^{K} w_k P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k). \tag{9}$$

The gap for P_F *is* $g_{min} \geq \min_{k=1}^{K} w_k g_{min}^k$.

The choice of the set of weights w_k in Property 3 is not unique in general. Also note that g_{min} may be greater than $\min_{k=1}^{K} w_k g_{min}^k$, because, for example, it might be the case that $g_{min} = w_k g_{min}^k$ for some unique k and no truth assignment violating F_k with cost $w_k g_{min}^k$ satisfies all other F_i's.

Property 3 states that a penalty function for the conjunction of sub-formulas can be obtained as a (weighted) sum of the penalty functions of the sub-formulas. The weights w_k are needed because penalty functions of formulas that share variables sum up biases or couplings, possibly resulting into out-of-range values (8). If the w_k's are smaller than 1, then the gap g_{min} of the final penalty function may become smaller. Also, Property 3 requires placing variables into qubits that are shared among conjunct subformulas. This may restrict the chances of finding suitable placements for the variables in the graph.

An alternative way of coping with this problem is to map shared variables into distinct qubits which are connected by chains of equivalences. Consider $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ as in Property 3. For every variable x_i and for every F_k where x_i occurs, we can replace the occurrences of x_i in F_k with a fresh variable ${x_i}^{k*}$, obtaining a formula $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^{k*})$ such that the sets $\underline{\mathbf{x}}^{k*}$ are all disjoint. Let

$$F^*(\underline{\mathbf{x}}^*) \stackrel{\text{def}}{=} \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^{k*}) \wedge \bigwedge_{\langle {x_i}^{k*}, {x_i}^{k'*} \rangle \in Eq(x_i)} ({x_i}^{k*} \leftrightarrow {x_i}^{k'*}) \tag{10}$$

where $\underline{\mathbf{x}}^* = \cup_k \underline{\mathbf{x}}^{k*}$, and $Eq(x_i)$ is any set of pairs $\langle {x_i}^{k*}, {x_i}^{k'*} \rangle$ of the variables replacing x_i such that the conjunction of equivalences in (10) states that of all of them are equivalent. By construction, $F(\underline{\mathbf{x}})$ is satisfiable if and only if $F^*(\underline{\mathbf{x}}^*)$ is satisfiable, and from every model $\underline{\mathbf{x}}^*$ for $F^*(\underline{\mathbf{x}}^*)$ we have a model $\underline{\mathbf{x}}$ for $F(\underline{\mathbf{x}})$ by simply assigning to each x_i the value of the corresponding ${x_i}^{k*}$s.

Now assume we have a penalty function $P_{F_k}(\underline{\mathbf{x}}^{k*}, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$ for each k with disjoint $\underline{\mathbf{a}}^k$. We recall from Example 2 that $(1 - {x_i}^{k*} {x_i}^{k'*})$ are penalty functions of gap 2 for the $({x_i}^{k*} \leftrightarrow {x_i}^{k'*})$ subformulas in (10). Thus we can apply Property 3 with all weights $w_k = 1$ and write a penalty function for $F^*(\underline{\mathbf{x}}^*)$ in the following way:

$$P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}}) = \sum_{k=1}^{K} P_{F_k}(\underline{\mathbf{x}}^{k*}, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k) + \sum_{\langle {x_i}^{k*}, {x_i}^{k'*} \rangle \in Eq(x_i)} (1 - {x_i}^{k*} {x_i}^{k'*}). \tag{11}$$

Note that the θ's stay within valid range because the $\underline{\mathbf{x}}^{k*}$s and $\underline{\mathbf{a}}^k$s are all disjoint and the biases of the $(1 - {x_i}^{k*} {x_i}^{k'*})$ terms are zero, so distinct sub-penalty functions in (11) involve disjoint groups of biases and couplings. Thus we have the following.

Property 4. $P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}})$ *in* (11) *is a penalty function for* $F^*(\underline{\mathbf{x}}^*)$ *in* (10). *The gap of* $P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}})$ *is* $g_{min} \geq \min(\min_{k=1}^{K} g_{min}^k, 2)$.

Thus, we can represent a single variable x_i with a series of qubits connected by strong couplings $(1 - x_i x_i')$. (For $x_i \leftrightarrow \neg x_i'$, we use $(1 + x_i x_i')$.) Notice that it is not necessary that every copy of variable x_i be connected to every other one; rather, to enforce the condition that all copies of x_i are logically equivalent, it suffices that the copies of x_i induce a connected graph. Moreover, additional copies of x_i may be introduced on unused vertices of the hardware graph G to facilitate connectedness. A set of qubits all representing the same variable in this way is called a *chain* and is the subject of §3.4. Thus, it is possible to implement $P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}})$ in (11) by placing the distinct penalty functions $P_{F_k}(\underline{\mathbf{x}}^{k*}, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$ into sub-graphs and connect them with chains.

Recall from §2.2 that a formula $F(\underline{\mathbf{x}})$ which is not a conjunction can always be decomposed into a conjunction of smaller formulas $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ by means of Tseitin's transformation (4). By Properties 3 and 4, this allows us to AND-decompose $F(\underline{\mathbf{x}})$ into multiple and smaller conjuncts that can be encoded separately and recombined. The problem thus reduces to choosing Boolean functions $(y_i \leftrightarrow F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i))$ and $F_m(\underline{\mathbf{x}}^m, \underline{\mathbf{y}}^m)$ whose penalty functions are easy to compute, have large gap, and whose combination keeps the gap of the penalty function for the original function as large as possible.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Example 6. Let $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_4 \leftrightarrow (x_3 \wedge (x_1 \oplus x_2))$. Applying (4) and (10) this can be rewritten as $\mathbf{F}^*(\underline{\mathbf{x}}, y, y') = (x_4 \leftrightarrow (x_3 \wedge y')) \wedge (y \leftrightarrow (x_1 \oplus x_2)) \wedge (y' \leftrightarrow y)$. The penalty functions of the three conjuncts can be produced as in Examples 3, 4 and 2 respectively, and summed as in Property 4:

$$\begin{aligned}
&P_{F^*}(\underline{\mathbf{x}}, y, y', \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \\
&= \frac{5}{2} - \frac{1}{2}x_3 - \frac{1}{2}y' + x_4 + \frac{1}{2}x_3y' - x_3x_4 - y'a_4 - x_4a_4 \\
&+ 5 + y + a_2 - a_3 + x_1a_1 - x_1a_2 - x_1a_3 - x_2a_1 - x_2a_2 - x_2a_3 + ya_2 - ya_3 \\
&+ 1 - yy' \\
&= \frac{17}{2} - \frac{1}{2}x_3 + x_4 + y - \frac{1}{2}y' + a_2 - a_3 + x_1a_1 - x_1a_2 - x_1a_3 - x_2a_1 - x_2a_2 \\
&\quad -x_2a_3 - x_3x_4 + \frac{1}{2}x_3y' - x_4a_4 + ya_2 - ya_3 - yy' - y'a_4
\end{aligned}$$

Notice that there is no interaction between the biases and couplings of the three components, only the offsets are summed up. The resulting gap is $min\{2, 2, 2\} = 2$. Then they can be placed, e.g., as in Fig. 5(c).

Overall, these facts suggest a "divide-and-conquer" approach for addressing the SATtoIsing problem:

(i) AND-decompose the input formula, by rewriting every conjunct $F(\underline{\mathbf{x}})$ which is not small enough into an equivalently-satisfiable one $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ as in (4) such that penalty functions for all its conjuncts can be easily computed;
(ii) rename shared variables and compute the global penalty functions as in Property 4;
(iii) place the sub-penalty functions into subgraphs and connect by chains equivalent qubits representing shared variables.

3.3. Exact penalty functions and MaxSAT

In order to encode MaxSAT, we require a stronger version of the penalty function in Definition 1.

Definition 2. *A penalty function* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ *is* **exact** *if for all* $\underline{\mathbf{x}}$ *such that* $F(\underline{\mathbf{x}}) = \bot$,

$$\min_{\{\underline{\mathbf{a}}\}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = g_{min}.$$

That is, an exact penalty function separates satisfying assignments from all non-satisfying ones by exactly the same gap g_{min}.

Example 7. The penalty function of $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} (x_1 \leftrightarrow x_2)$ in Example 2 is exact, whereas those of $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$ and $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \oplus x_2)$ in Examples 3 and 4 are not exact.

Exact penalty functions allow for the encoding of weighted MaxSAT problems, with some restrictions. The following fact is a straightforward consequence of Property 3 and Definition 2.

Proposition 2. *Let* $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ *be a Boolean formula s.t.* $\underline{\mathbf{x}} = \cup_k \underline{\mathbf{x}}^k$, *and* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \stackrel{def}{=} \sum_{k=1}^{K} P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$, *where* $\underline{\mathbf{a}} \stackrel{def}{=} \cup_k \underline{\mathbf{a}}^k$ *s.t. the* $\underline{\mathbf{a}}^k$ *are all disjoint, each* $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ *is an exact penalty function for* F_k *of gap* g_k. *Let* $\underline{\mathbf{x}}$,$\underline{\mathbf{a}}$ *be a truth assignment which minimizes* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. *Then* $\underline{\mathbf{x}}$ *is a solution for the weighted MaxSAT problem* $\{\langle F_k, g_k\rangle\}_k$.

Proposition 2 allows for encoding a generic weighted MaxSAT problem $\{\langle F_k, c_k\rangle\}_k$ by setting $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \sum_{k=1}^{K} w_k P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ where $w_k \stackrel{\text{def}}{=} \frac{c_k}{g_k} \cdot c$ and c is a normalization factor (7). Notice that in Proposition 2 the penalty functions $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ must be exact; otherwise, a solution $\underline{\mathbf{x}}$,$\underline{\mathbf{a}}$ that is optimal for MaxSAT but violates some F_k might not minimize $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ if $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k) > g_k$.

In §3.2 we outlined a "divide-and-conquer" approach for SATtoIsing based on the idea of mapping shared variables into distinct qubits which are then connected by chains of equivalences. Applying the same approach to MaxSAT is not as straightforward, because Property 4 cannot always be combined with Proposition 2 in a useful way. Consider the scenario in Property 4, and suppose we want to use (11) to solve the MaxSAT problem $\{\langle F_k, g_k\rangle\}_k$ as with Proposition 2. As the following example shows, there may be minimum-energy solutions of (11) which violate some equivalence $({x_i}^{k*} \leftrightarrow {x_i}^{k'*})$ in (10) if this avoids violating one or more of the F_k's whose sum of gaps is greater than 2. Such a solution is *not* a solution of the MaxSAT problem, because it corresponds to assigning different truth values to distinct instances of the same variable in the original problem.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Example 8. Consider the trivial MaxSAT problem $\{\langle F_i(x), c\rangle\}_{i=1}^4$ for some penalty value $c > 0$ where $F_1(x) = F_2(x) \stackrel{\text{def}}{=} x$, and $F_3(x) = F_4(x) \stackrel{\text{def}}{=} \neg x$. The two possible solutions $x = \top$ and $x = \bot$ are both optimum with penalty $2c$ and falsify F_3, F_4 and F_1, F_2 respectively. We have the following normal and exact penalty functions: $P_{F_1}(x) = P_{F_2}(x) = 2 - 2x$ and $P_{F_3}(x) = P_{F_4}(x) = 2 + 2x$, each of gap $g_i = 4$. Suppose we want to encode the problem in such a way to fit into a linear chain of 4 qubits adopting the encoding in Property 4. We introduce four copies of x, namely x^1, x^2, x^3, x^4, and obtain:

$$\begin{aligned} F^*(x^1, x^2, x^3, x^4) &= x^1 \wedge x^2 \wedge \neg x^3 \wedge \neg x^4 \wedge (x^1 \leftrightarrow x^2) \wedge (x^2 \leftrightarrow x^3) \wedge (x^3 \leftrightarrow x^4) \\ P_{F^*}(x^1, x^2, x^3, x^4) &= (2 - 2x^1) + (2 - 2x^2) + (2 + 2x^3) + (2 + 2x^4) + \\ &\quad (1 - x^1x^2) + (1 - x^2x^3) + (1 - x^3x^4) \\ &= 11 - 2x^1 - 2x^2 + 2x^3 + 2x^4 - x^1x^2 - x^2x^3 - x^3x^4. \end{aligned}$$

The minimum-energy solution to P_{F^*} is $x^1 = x^2 = 1$ and $x^3 = x^4 = -1$ with $P_{F^*}(\ldots) = 2$, which violates the equivalence $(x^2 \leftrightarrow x^3)$. The correct MaxSAT solutions $x^1 = x^2 = x^3 = x^4 = 1$ and $x^1 = x^2 = x^3 = x^4 = -1$ both have $P_{F^*}(\ldots) = 8$.

In general, the problem arises when it is energetically cheaper to violate some equivalence $(x_i^{k^*} \leftrightarrow x_i^{k'^*})$ in a chain in (10) than to violate all the penalty functions $\{F_k(\underline{\mathbf{x}^k}) : x_i \in \underline{\mathbf{x}^k}\}$ on one side of the equivalence. One solution to this problem is to multiply the P_{F_k}'s by sufficiently small weights $w_k < 1$, at the cost reducing their gaps g_k. In the following we discuss the bounds that can be placed on w_k.

Let $\mathcal{I}$ denote the indices of the functions $F_k(\underline{\mathbf{x}^k})$ that use the variable x_i; that is, $\mathcal{I} = \{k : x_i \in \underline{\mathbf{x}^k}\}$. An equivalence $(x_i^{k^*} \leftrightarrow x_i^{k'^*})$ in the chain of x_i splits the chain into two subchains, and splits $\mathcal{I}$ into two subsets $\mathcal{I}_k$ and $\mathcal{I}_{k'}$ such that $(x_i^{k^*} \leftrightarrow x_i^{k'^*})$ connects the functions of $\mathcal{I}_k$ to the functions of $\mathcal{I}_{k'}$. Assume we have a desired gap $g_{desired} > 0$ separating solutions with broken chains from true solutions. Then a sufficiently large gap for the equivalence $(x_i^{k^*} \leftrightarrow x_i^{k'^*})$ is

$$g_{(k,k')} = \min\left(\sum_{j\in\mathcal{I}_k} g_j, \sum_{j\in\mathcal{I}_{k'}} g_j\right) + g_{desired},$$

as this gap ensures that it is $g_{desired}$ cheaper to violate all the constraints in $\mathcal{I}_k$ or $\mathcal{I}_{k'}$ then to violate $(x_i^{k^*} \leftrightarrow x_i^{k'^*})$. Recall from (10) that $Eq(x_i)$ is the set of variable pairs $(x_i^{k^*}, x_i^{k'^*})$ that form equivalences $(x_i^{k^*} \leftrightarrow x_i^{k'^*})$ in the chain of x_i. To ensure that all equivalence constraints are not violated, a sufficient gap for the entire chain is

$$g_{chain} = \max_{(x_i^{k^*}, x_i^{k'^*})\in Eq(x_i)} g_{(k,k')}. \tag{12}$$

Finally, recalling that each equivalence has gap 2, we update the weight definition in Proposition 2 for each $k \in \mathcal{I}$[9]:

$$w_k = \frac{2 \cdot c_k}{g_k \cdot g_{chain}} \tag{13}$$

An alternative bound on g_{chain} is given in [37]. In the paper, the author bounds the chain strength required to ensure that all minima of an embedded QUBO problem can be mapped to a minimum of the original QUBO problem (see §3.4 below). Let $\theta_i^* = \sum_k w_k\theta_i$ be the bias value obtained by sharing the x_i variable as in Property 3.[10] If x_i is substituted by a chain with l_i endpoints, QUBO minima are preserved if the chain gap is the following:

$$g_{chain} = 2\frac{l_i - 1}{l_i}\left(\sum_{(i,j)\in E} |\theta_{ij}^*| - |\theta_i^*|\right) + g_{desired} \tag{14}$$

This alternative bound is sometimes lower than (12), especially when $|\theta_i^*|$ is high. Note that, as the original paper explains, if the bound value is negative then P_{F^*} is monotonic on x_i. If that is the case, then $x_i = -sgn(\theta_i^*)$ always minimizes P_{F^*}, so we can fix the value of x_i and there is no need for a chain.

In general, neither (12) nor (14) are typically very tight bounds on required chain gap, and finding the smallest viable chain gap analytically appears to be a difficult problem. In practice g_{chain} is often determined empirically; this is discussed further in §7.

Overall, the MaxSATtoIsing problem is subject to some intrinsic limitations. Firstly, it requires the usage of *exact* penalty functions for its sub-formulas, which are more difficult to obtain. Secondly, the need to re-weight penalty functions to

[9] Note that the normalization factor c here is 1 as chains are normal.

[10] For simplicity, we assume to share a single x_i, so each $\theta_{ij}^* = w_k\theta_{ij}^k$ for some unique k.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

ensure chain equivalences are not violated typically results in smaller gaps. Thirdly, it is difficult to directly encode hard constraints in a MaxSAT problem; this again requires re-weighting soft constraints by very small factors, reducing their gaps accordingly.

3.4. 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 [37,38]. More precisely, let $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ be a penalty function whose interactions define a graph G_F (so x_i and x_j are adjacent iff $\theta_{ij} \neq 0$) and let G_H be a QA hardware graph. A *minor embedding* of G_F in G_H is a function $\Phi : V_{G_F} \to 2^{V_{G_H}}$ such that:

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

The image $\Phi(x_i)$ of a G_F-vertex is a chain, and the set of qubits in a chain are constrained to be equivalent using $(1 - x_i^{k*} x_i^{k'*})$ couplings as in Equation (11).

Embedding generic graphs is a computationally difficult problem [39], although certain structured problem graphs may be easily embedded in the Chimera graph [40,41] and heuristic algorithms may also be used [42]. 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 [24].

A different approach to finding models for $F(\underline{\mathbf{x}})$, *global embedding*, is based on first finding a penalty function on a complete graph G_F on $n + h$ variables, and secondly, embedding G_F into a hardware graph G_H using chains (e.g., using [40]). Following [34], 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.

4. Encoding small Boolean sub-formulas

In this section we present general SMT/OMT-based techniques to address the encoding problem for small Boolean formulas $F(\underline{\mathbf{x}})$.

4.1. Computing penalty functions via SMT/OMT($\mathcal{LRA}$)

Given $\underline{\mathbf{x}} \stackrel{\text{def}}{=} \{x_1, ..., x_n\}$, $\underline{\mathbf{a}} \stackrel{\text{def}}{=} \{a_1, ..., a_h\}$, $F(\underline{\mathbf{x}})$ as in Section 3, 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 (5) corresponds to solving the following problem[11]:

$$\begin{array}{l}\text{For every } i\ j, \ \text{ find } \theta_i \in [-2, 2],\ \theta_{ij} \in [-1, 1] \ \text{ such that}\\ \forall \underline{\mathbf{x}}. \left[\begin{array}{l}(\ \ F(\underline{\mathbf{x}}) \to \exists \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0)) \wedge \\ (\ \ F(\underline{\mathbf{x}}) \to \forall \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq 0)) \wedge \\ (\neg F(\underline{\mathbf{x}}) \to \forall \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}))\end{array}\right].\end{array} \tag{15}$$

By applying Shannon's expansion (§2.2) to the quantifiers in (15), the problem reduces straightforwardly to solving the following SMT($\mathcal{LRA}$) problem:

$$\Phi(\underline{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \bigwedge_{z_i \in \underline{\mathbf{x}}, \underline{\mathbf{a}}} (-2 \leq \theta_i) \wedge (\theta_i \leq 2) \wedge \bigwedge_{\substack{z_i, z_j \in \underline{\mathbf{x}}, \underline{\mathbf{a}} \\ i<j}} (-1 \leq \theta_{ij}) \wedge (\theta_{ij} \leq 1) \tag{16}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0) \tag{17}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq 0) \tag{18}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}). \tag{19}$$

[11] As in (5), we implicitly assume $\theta_{ij} = 0$ when $(i, j) \notin E$.

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 $\langle \Phi(\underline{\boldsymbol{\theta}}), g_{min} \rangle$. Notice that, since g_{min} is maximum, $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is also normal.

Intuitively, (16) states the ranges of the $\underline{\boldsymbol{\theta}}$; (17) and (18) 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; (19) 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 be greater than or equal to 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 (6); if $\Phi(\underline{\boldsymbol{\theta}})$ is unsatisfiable, then there is no $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ complying with (6) for the given placement.

Note that, if $\underline{\mathbf{a}} = \emptyset$, then the OMT($\mathcal{LRA}$) maximization problem $\langle \Phi(\underline{\boldsymbol{\theta}}), g_{min} \rangle$ reduces to a linear program because the disjunctions in (17) disappear.

Example 9. Consider the AND function $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$ of Example 3 and its placement in Fig. 5(a), involving one ancilla a. Then (15) is:

$$
\begin{array}{l}
\bigwedge_{i=1}^{3}((-2 \le \theta_{x_i}) \wedge (\theta_{x_i} \le 2)) \wedge (-2 \le \theta_a) \wedge (\theta_a \le 2) \wedge \\
\bigwedge_{i=2}^{3}(-1 \le \theta_{x_1 x_i}) \wedge (\theta_{x_1 x_i} \le 1) \wedge \bigwedge_{i=2}^{3}(-1 \le \theta_{x_i a}) \wedge (\theta_{x_i a} \le 1) \wedge \\
\forall x_1 \\ \forall x_2 \\ \forall x_3
\left[\begin{array}{l}
(\quad (x_3 \leftrightarrow (x_1 \wedge x_2)) \rightarrow \\
\exists a.(\theta_0 + \sum_{i=1}^{3} \theta_{x_i} x_i + \theta_a a x_i + \sum_{i=2}^{3} \theta_{x_1 x_i} x_1 x_i x_i + \sum_{i=2}^{3} + \theta_{x_i a} x_i a = 0)) \wedge \\
(\quad (x_3 \leftrightarrow (x_1 \wedge x_2)) \rightarrow \\
\forall a.(\theta_0 + \sum_{i=1}^{3} \theta_{x_i} x_i + \theta_a a x_i + \sum_{i=2}^{3} \theta_{x_1 x_i} x_1 x_i x_i + \sum_{i=2}^{3} + \theta_{x_i a} x_i a \ge 0)) \wedge \\
(\neg(x_3 \leftrightarrow (x_1 \wedge x_2)) \rightarrow \\
\forall a.(\theta_0 + \sum_{i=1}^{3} \theta_{x_i} x_i + \theta_a a x_i + \sum_{i=2}^{3} \theta_{x_1 x_i} x_1 x_i x_i + \sum_{i=2}^{3} + \theta_{x_i a} x_i a \ge g_{min}))
\end{array}\right]
\end{array}
$$

Example 10. Consider again the AND function $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$ of Example 3 and its placement in Fig. 5(a), involving one ancilla a. Then $\Phi(\underline{\boldsymbol{\theta}})$ of (16)-(19) is:

$$
\begin{array}{ll}
\bigwedge_{i=1}^{3}((-2 \le \theta_{x_i}) \wedge (\theta_{x_i} \le 2)) \wedge (-2 \le \theta_a) \wedge (\theta_a \le 2) \wedge & (16) \\
\bigwedge_{i=2}^{3}(-1 \le \theta_{x_1 x_i}) \wedge (\theta_{x_1 x_i} \le 1) \wedge \bigwedge_{i=2}^{3}(-1 \le \theta_{x_i a}) \wedge (\theta_{x_i a} \le 1) \wedge & \\
\hline
\left(\begin{array}{l}(\theta_0 + \theta_{x_1} + \theta_{x_2} + \theta_{x_3} + \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} + \theta_{x_2 a} + \theta_{x_3 a} = 0) \vee \\ (\theta_0 + \theta_{x_1} + \theta_{x_2} + \theta_{x_3} - \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} - \theta_{x_2 a} - \theta_{x_3 a} = 0)\end{array}\right) \wedge & (17) \\
\ldots & \\
\left(\begin{array}{l}(\theta_0 - \theta_{x_1} - \theta_{x_2} - \theta_{x_3} + \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} - \theta_{x_2 a} - \theta_{x_3 a} = 0) \vee \\ (\theta_0 - \theta_{x_1} - \theta_{x_2} - \theta_{x_3} - \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} + \theta_{x_2 a} + \theta_{x_3 a} = 0)\end{array}\right) \wedge & \\
\hline
\left(\begin{array}{l}(\theta_0 + \theta_{x_1} + \theta_{x_2} + \theta_{x_3} + \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} + \theta_{x_2 a} + \theta_{x_3 a} \ge 0) \wedge \\ (\theta_0 + \theta_{x_1} + \theta_{x_2} + \theta_{x_3} - \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} - \theta_{x_2 a} - \theta_{x_3 a} \ge 0)\end{array}\right) \wedge & (18) \\
\ldots & \\
\left(\begin{array}{l}(\theta_0 - \theta_{x_1} - \theta_{x_2} - \theta_{x_3} + \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} - \theta_{x_2 a} - \theta_{x_3 a} \ge 0) \wedge \\ (\theta_0 - \theta_{x_1} - \theta_{x_2} - \theta_{x_3} - \theta_a + \theta_{x_1 x_2} + \theta_{x_1 x_3} + \theta_{x_2 a} + \theta_{x_3 a} \ge 0)\end{array}\right) \wedge & \\
\hline
\left(\begin{array}{l}(\theta_0 + \theta_{x_1} + \theta_{x_2} - \theta_{x_3} + \theta_a + \theta_{x_1 x_2} - \theta_{x_1 x_3} + \theta_{x_2 a} - \theta_{x_3 a} \ge g_{min}) \wedge \\ (\theta_0 + \theta_{x_1} + \theta_{x_2} - \theta_{x_3} - \theta_a + \theta_{x_1 x_2} - \theta_{x_1 x_3} - \theta_{x_2 a} + \theta_{x_3 a} \ge g_{min})\end{array}\right) \wedge & (19) \\
\ldots & \\
\left(\begin{array}{l}(\theta_0 - \theta_{x_1} - \theta_{x_2} + \theta_{x_3} + \theta_a + \theta_{x_1 x_2} - \theta_{x_1 x_3} - \theta_{x_2 a} + \theta_{x_3 a} \ge g_{min}) \wedge \\ (\theta_0 - \theta_{x_1} - \theta_{x_2} + \theta_{x_3} - \theta_a + \theta_{x_1 x_2} - \theta_{x_1 x_3} + \theta_{x_2 a} - \theta_{x_3 a} \ge g_{min})\end{array}\right). &
\end{array}
$$

To force $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to be an *exact* penalty function, we add the following conjunct inside the square brackets of (15):

$$(\neg F(\underline{\mathbf{x}}) \rightarrow \exists \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = g_{min})), \tag{20}$$

which forces $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to be exactly equal to the gap for at least one $\underline{\mathbf{a}}$. Thus we conjoin the Shannon's expansion of (20) to $\Phi(\underline{\boldsymbol{\theta}})$ in (16)-(19):

$$\ldots \wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = g_{min}). \tag{21}$$

4.2. Improving efficiency and scalability using variable elimination

In the SMT/OMT($\mathcal{LRA}$) formulation (16)-(19), $\Phi(\underline{\boldsymbol{\theta}})$ grows exponentially with 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 **tw** is the treewidth of the subgraph of G spanned by the qubits corresponding to the

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

ancillary variables, G_a. For the Chimera graph, even when h is as large as 32, **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 [43] 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 **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 local 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.

To describe the method, we first reformulate equations (18)-(19) by introducing *witness* binary variables $\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}}) \in \{-1,1\}^h$ to enforce the equality constraints (17), that is, $P_F(\underline{\mathbf{x}}, \underline{\boldsymbol{\beta}}(\underline{\mathbf{x}})|\underline{\boldsymbol{\theta}}) = 0$. Thus, we can rewrite $\Phi(\underline{\boldsymbol{\theta}})$ as the SMT problem $\Phi(\underline{\boldsymbol{\theta}}, \underline{\boldsymbol{\beta}})$ defined by[12]

$$\Phi(\underline{\boldsymbol{\theta}}, \underline{\boldsymbol{\beta}}) \stackrel{\text{def}}{=} (16) \wedge (18) \wedge (19) \\ \wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} \big((\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}}) \equiv \underline{\mathbf{a}}) \wedge (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0) \big).$$

Consider first the case when the graph G_a has no edges. If, for $i = 1, \ldots, h$, we define

$$f_i(a_i|\underline{\mathbf{x}}) = \theta_i a_i + a_i \sum_{j:ij \in E} \theta_{ij} x_j,$$

then we can write

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = c(\underline{\mathbf{x}}) + \sum_{i=1}^{h} f_i(a_i|\underline{\mathbf{x}}),$$

where $c(\underline{\mathbf{x}})$ does not depend on the ancillary variables. Thus,

$$\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = c(\underline{\mathbf{x}}) + \sum_{i=1}^{h} \min_{a_i \in \{-1,1\}} f_i(a_i|\underline{\mathbf{x}}). \tag{22}$$

If $\underline{\boldsymbol{\theta}}$ is fixed, solving (22) is straightforward. However, since $\underline{\boldsymbol{\theta}}$ is a variable, the contribution $\min_{a_i \in \{-1,1\}} f_i(a_i|\underline{\mathbf{x}})$ is a function of $\underline{\boldsymbol{\theta}}$, for each $i = 1, \ldots, h$. Each of these minimums will be associated with a continuous variable, denoted by $m_i(\emptyset|\underline{\mathbf{x}})$, and referred to as a *message* variable (the naming will be clearer in the general case). To relate $m_i(\emptyset|\underline{\mathbf{x}})$ with $\min_{a_i \in \{-1,1\}} f_i(a_i|\underline{\mathbf{x}})$, we impose the constraints

$$m_i(\emptyset|\underline{\mathbf{x}}) \leq f_i(-1|\underline{\mathbf{x}}) \quad \text{and} \quad m_i(\emptyset|\underline{\mathbf{x}}) \leq f_i(1|\underline{\mathbf{x}}).$$

Thus, if $F(\underline{\mathbf{x}}) = \bot$, since the message variables are lower bounds on the true minimums of (22), to enforce (19) we need simply add the constraints

$$c(\underline{\mathbf{x}}) + \sum_{i=1}^{h} m_i(\emptyset|\underline{\mathbf{x}}) \geq g_{min}.$$

When $F(\underline{\mathbf{x}}) = \top$, we need to ensure that the message variables take the minimums of (22). Note that variable $\beta_i(\underline{\mathbf{x}})$ identifies the value of the ancillary variable i that achieves the minimum in (22). To relate the values of $\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}})$ and the message variables $m(\emptyset|\underline{\mathbf{x}})$ we add the SMT constraints

$$\beta_i(\underline{\mathbf{x}}) \Rightarrow \big(m_i(\emptyset|\underline{\mathbf{x}}) = f_i(1|\underline{\mathbf{x}})\big),$$
$$\neg\beta_i(\underline{\mathbf{x}}) \Rightarrow \big(m_i(\emptyset|\underline{\mathbf{x}}) = f_i(-1|\underline{\mathbf{x}})\big).$$

Finally, to impose (17) and (18), we need that

$$c(\underline{\mathbf{x}}) + \sum_{i=1}^{h} m_i(\emptyset|\underline{\mathbf{x}}) = 0.$$

[12] For vectors $\underline{\mathbf{a}}, \underline{\mathbf{b}}$, we use $\underline{\mathbf{a}} \equiv \underline{\mathbf{b}}$ as a shorthand for $(a_1 = b_1) \wedge (a_2 = b_2) \wedge (a_3 = b_3) \wedge \ldots$.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Since G is usually sparse, it is likely that two binary states $\underline{\mathbf{x}}$ and $\underline{\mathbf{x}}'$ agree on the bits adjacent to a fixed ancillary variable i. In this case, it is clear that $m_i(\emptyset|\underline{\mathbf{x}}) = m_i(\emptyset|\underline{\mathbf{x}}')$, and we can use a single message variable for both states. This observation can be extended to the general case and will be valuable to reduce the size and strengthen the SMT problem formulation.

Next consider the general case when $|E(G_a)| > 0$. In what follows, $c(\underline{\mathbf{x}})$ and $f_i(a_i|\underline{\mathbf{x}})$ are defined as above. Assume first $\underline{\boldsymbol{\theta}}$ is fixed. Given $\underline{\mathbf{x}}$, we want to solve the Ising model $\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. Variable elimination proceeds in order, eliminating one ancillary variable at a time. Suppose that ancillary variables are eliminated in the order $h, h-1, \ldots, 1$. Each ancillary variable i is associated with a set $\mathcal{F}_i$ of *factors*, which are functions that depend on ancillary variable i and none or more ancillary variables with index less than i. The sets $\mathcal{F}_i$ are called *buckets*, and are updated throughout the computation. Initially, each $\mathcal{F}_i$ consists of ancilla-ancilla edges[13] $f_{i,k}(a_i, a_k) = \theta_{ik}\ a_i a_k$ for $ik \in E(G_a)$, $k < i$. Let $\mathcal{V}_i$ denote the set of ancillary variables involved in the factors of bucket $\mathcal{F}_i$ other than variable i itself (thus, all variable indices in $\mathcal{V}_i$ are less than i, or $\mathcal{V}_i = \emptyset$). For a fixed $\underline{\mathbf{a}}$ and a subset of ancillary variables $\mathcal{U}$, we use $\underline{\mathbf{a}}_{\mathcal{U}}$ to denote $\{a_i : i \in \mathcal{U}\}$. Variable h is eliminated first. Note that once variables in $\mathcal{V}_h$ are instantiated to $\underline{\mathbf{a}}_{\mathcal{V}_h}$, the optimal setting of variable h is readily available by solving

$$g_h(\underline{\mathbf{a}}_{\mathcal{V}_h}) = \min_{a_h} f_h(a_h|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_h} f(\underline{\mathbf{a}}_{\mathcal{V}_h}, a_h). \tag{23}$$

Here $f = f_{i,h} \in \mathcal{F}_h$ represents an edge ih between ancillary variables i and h, $i < h$ (abusing notation we write $f(a_i, a_h)$ as $f(\underline{\mathbf{a}}_{\mathcal{V}_h}, a_h)$), and $\mathcal{F}_h$ contains all edges adjacent to h. The $2^{|\mathcal{V}_h|}$ possible settings of $\underline{\mathbf{a}}_{\mathcal{V}_h}$ define $2^{|\mathcal{V}_h|}$ values (23). These values define new factor g_h, a function of variables $\underline{\mathbf{a}}_{\mathcal{V}_h}$, that is added to the bucket $\mathcal{F}_i$ of variable i with largest index in $\mathcal{V}_h$. For each instantiation of $\underline{\mathbf{a}}_{\mathcal{V}_h}$ we define the message $m_h(\underline{\mathbf{a}}_{\mathcal{V}_h}|\underline{\mathbf{x}})$ as $g_h(\underline{\mathbf{a}}_{\mathcal{V}_h})$. Iteratively, eliminating variable i is accomplished by solving, for each setting of $\underline{\mathbf{a}}_{\mathcal{V}_i}$,

$$g_i(\underline{\mathbf{a}}_{\mathcal{V}_i}) = \min_{a_i} f_i(a_i|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, a_i) \tag{24}$$

generating a new factor g_i, a function of $\underline{\mathbf{a}}_{\mathcal{V}_i}$. For each one of the $2^{|\mathcal{V}_i|}$ possible values of g_i we define message $m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}})$ to be $g_i(\underline{\mathbf{a}}_{\mathcal{V}_i})$. Factor g_i is then added to bucket $\mathcal{F}_k$ where k is the largest index in $\mathcal{V}_i$. When $V_i = \emptyset$, (24) takes the form

$$\min_{a_i} f_i(a_i|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_i} f(a_i) \tag{25}$$

that determines the optimal value of a_i; the message corresponding to the value of this minimum is $m_i(\emptyset|\underline{\mathbf{x}})$. All variables with $V_i = \emptyset$ can be eliminated at the same time, so that, at termination, the value of the Ising problem $\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is equal to

$$c(\underline{\mathbf{x}}) + \sum_{i:\mathcal{V}_i=\emptyset} m_i(\emptyset|\underline{\mathbf{x}})$$

which will be equal to $\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. Notice that the number of additional messages is $O(\sum_i 2^{|\mathcal{V}_i|})$, where each $\mathcal{V}_i$ corresponds to the time when variable i is eliminated. When G_a has treewidth t, there is an elimination order for which each $|\mathcal{V}_i| \le t$, which typically, by our low treewidth assumption, will be much smaller than 2^h.

When $\underline{\boldsymbol{\theta}}$ is not fixed, as in the case when there were no edges, the messages are variables. Since these message variables represent minimums, we upper bound the message variables adding the constraints

$$m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) \le f_i(-1|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, -1)$$

$$m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) \le f_i(1|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, 1).$$

As before, if $F(\underline{\mathbf{x}}) = \bot$, the constraint (19) can be replaced with

$$c(\underline{\mathbf{x}}) + \sum_{i:\mathcal{V}_i=\emptyset} m_i(\emptyset|\underline{\mathbf{x}}) \ge g_{min}, \tag{26}$$

since the message variables provide a lower bound on (24). When $F(\underline{\mathbf{x}}) = \top$, we must ensure that all the message variables are tight. For a subset of ancillary variables $\mathcal{U}$, let $\underline{\boldsymbol{\beta}}_{\mathcal{U}}(\underline{\mathbf{x}}) = \{\beta_i(\underline{\mathbf{x}}) : i \in \mathcal{U}\}$. Thus, we must have that for all $\underline{\mathbf{a}}_{\mathcal{V}_i}$

[13] G_a is an undirected graph. An edge is defined by a pair of vertices, say i and k; for convenience, in this section we associate this edge with the ordered pair ik with $k < i$.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

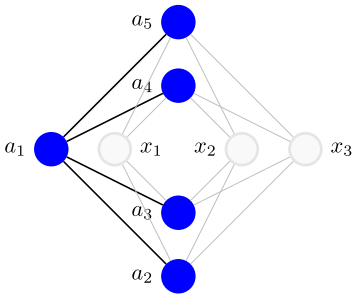


Fig. 6. Unit cell with 5 ancillas. The subgraph, G_a, induced by the ancilla nodes is highlighted in bold. Pairwise factors $f_{i,j}(a_i, a_j)$ are associated with each edge of G_a and unary factors $f_i(a_i|\underline{\mathbf{x}})$ with each node of G_a.

$$\left[\underline{\boldsymbol{\beta}}_{\mathcal{V}_i}(\underline{\mathbf{x}}) \equiv \underline{\mathbf{a}}_{\mathcal{V}_i} \wedge \beta_i(\underline{\mathbf{x}})\right] \Rightarrow \left[m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) = f_i(1|\underline{\mathbf{x}}) + \sum_{f\in\mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, 1)\right]$$

$$\left[\underline{\boldsymbol{\beta}}_{\mathcal{V}_i}(\underline{\mathbf{x}}) \equiv \underline{\mathbf{a}}_{\mathcal{V}_i} \wedge \neg\beta_i(\underline{\mathbf{x}})\right] \Rightarrow \left[m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) = f_i(-1|\underline{\mathbf{x}}) + \sum_{f\in\mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i} - 1)\right].$$

In this way, we can enforce that $\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\boldsymbol{\theta}) = 0$ (that is, constraints (17) and (18)), with the constraint

$$c(\underline{\mathbf{x}}) + \sum_{i:\mathcal{V}_i=\emptyset} m_i(\emptyset|\underline{\mathbf{x}}) = 0. \tag{27}$$

Example 11. Consider a penalty over 3 variables with 5 ancillas embedded in a unit cell (see Fig. 6). For each $\underline{\mathbf{x}}$, ancilla variables are eliminated in the order $a_5, a_4, \cdots, a_1$. The buckets are $\mathcal{F}_5 = \{f_{1,5}\}$, $\mathcal{F}_4 = \{f_{1,4}\}$, $\mathcal{F}_3 = \{f_{1,3}\}$, $\mathcal{F}_2 = \{f_{1,2}\}$, and $\mathcal{F}_1 = \{\}$. The dependent variable sets of each bucket are $\mathcal{V}_5 = \{a_1\}$, $\mathcal{V}_4 = \{a_1\}$, $\mathcal{V}_3 = \{a_1\}$, $\mathcal{V}_2 = \{a_1\}$, and $\mathcal{V}_1 = \{\}$. Elimination of a_5 is accomplished by posting the message constraints $m_5(a_1|\underline{\mathbf{x}}) \le f_5(1|\underline{\mathbf{x}}) + f_{1,5}(a_1, 1)$ and $m_5(a_1|\underline{\mathbf{x}}) \le f_5(-1|\underline{\mathbf{x}}) + f_{1,5}(a_1, -1)$. This elimination (for each choice of $a_{\mathcal{V}_5} = a_1$) generates a new factor $g_5(a_1)$ which is added to bucket $\mathcal{F}_1$. Next, we proceed to a_4. In this example, no new factors have been added to bucket $\mathcal{F}_4$ and elimination proceeds as above where factor $g_4(a_1)$ is added to $\mathcal{F}_1$ and message constraints $m_4(a_1|\underline{\mathbf{x}}) \le f_4(1|\underline{\mathbf{x}}) + f_{1,4}(a_1, 1)$ and $m_4(a_1|\underline{\mathbf{x}}) \le f_4(-1|\underline{\mathbf{x}}) + f_{1,4}(a_1, -1)$ are posted. This process repeats to the final ancilla a_1 whose bucket now includes $\mathcal{F}_1 = \{g_2, g_3, g_4, g_5\}$. Elimination of a_1 is accomplished by adding the message constraints $m_1(\emptyset|\underline{\mathbf{x}}) \le f_1(1|\underline{\mathbf{x}}) + \sum_{i=2}^{5} g_i(1)$ and $m_1(\emptyset|\underline{\mathbf{x}}) \le f_1(-1|\underline{\mathbf{x}}) + \sum_{i=2}^{5} g_i(-1)$. The minimal penalty value at the given $\underline{\mathbf{x}}$ is then lower bounded by $c(\underline{\mathbf{x}}) + m_1(\emptyset|\underline{\mathbf{x}})$. This bound is made tight for infeasible $\underline{\mathbf{x}}$ by requiring $c(\underline{\mathbf{x}}) + m_1(\emptyset|\underline{\mathbf{x}}) \ge g_{min}$ and made tight for feasible $\underline{\mathbf{x}}$ by imposing the constraints associated with the $\beta(\underline{\mathbf{x}})$ variables.

As noted in the case when G_a has no edges, some message variables will always have the same values. In fact, significant additional model reduction can be accomplished by identifying message variables that have to be the same across many states $\underline{\mathbf{x}}$. For instance, $m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) = m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}')$ if their corresponding upper bounds are the same (propagating from h down to i). Because G is sparse, the number of message variables can typically be reduced by an order of magnitude or more in this way.

In many cases, for counter-models $\underline{\mathbf{x}}$, $F(\underline{\mathbf{x}}) = \perp$, some constraints (19) may be dropped or relaxed without altering the optimal solution of the original SMT problem. For instance, we could include only constraints (19) for counter-models $\underline{\mathbf{x}}$ that are within Hamming distance at most d from all models of F. In our experiments, using $d \le 3$ sufficed in most cases.

Alternatively, also for counter-models, the variable elimination lower bounds (26) can be relaxed by weaker lower bounds such as a linear programming relaxation of the corresponding Ising problem, that requires $O(|V| + |E|)$ continuous variables and inequalities per $\underline{\mathbf{x}}$, $F(\underline{\mathbf{x}}) = \perp$. For instance, a linear programming lower bound on the QUBO formulation

$$\min_{y_i\in\{0,1\}} \sum_{i\in V} c_i y_i + \sum_{e=\{i,j\}\in E} q_e\, y_i y_j\,,$$

is the following:

$$\text{Minimize} \quad \sum_{i\in V} c_i x_i + \sum_{e\in E} q_e\, z_e \tag{28}$$

subject to

$$z_e - y_i - y_j \ge -1 \qquad \text{for each } e = ij \in E, i < j \quad (\lambda_e) \tag{29}$$

$$-z_e + y_i \ge 0 \qquad \text{for each } e = ij \in E, i < j \quad (\lambda^h_{e,i}) \tag{30}$$

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

$$-z_e + y_j \geq 0 \qquad \text{for each } e = ij \in E, i < j \quad (\lambda^t_{e,j}) \tag{31}$$

$$-y_i \geq -1 \qquad \text{for each } i \in V \quad (\alpha_i) \tag{32}$$

$$y_i, z_e \geq 0 \tag{33}$$

Its linear programming dual is given by

$$\text{Maximize} \quad -\sum_{e\in E}\lambda_e - \sum_{i\in V}\alpha_i \tag{34}$$

subject to

$$\lambda_e - \lambda^h_{e,i} - \lambda^t_{e,j} \leq q_e \qquad \text{for each } e = ij \in E, i < j \tag{35}$$

$$-\sum_{e:i\in e}\lambda_e + \sum_{e=ik\in E, i<k}\lambda^h_{e,i} + \sum_{e=ki\in E,k<i}\lambda^t_{e,i} - \alpha_i \leq c_i \qquad \text{for each } i \in V \tag{36}$$

$$\lambda_e, \lambda^h_{e,i}, \lambda^t_{e,i}, \alpha_i \geq 0 \tag{37}$$

Notice that if c and q are variables, the dual problem is still linear in the dual variables, c and q. Thus, we can guarantee (in one direction only) that the value of the QUBO is at least g with the set of linear inequalities

$$-\sum_{e\in E}\lambda_e - \sum_{i\in V}\alpha_i \geq g \tag{38}$$

$$(35), (36), (37) \tag{39}$$

Note that we can always take

$$(-\sum_{e:i\in e}\lambda_e + \sum_{e=ik\in E,i<k}\lambda^h_{e,i} + \sum_{e=ki\in E,k<i}\lambda^t_{e,i} - c_i)^+ = \alpha_i.$$

To make this work for an Ising problem, the c and q have to be written as linear functions of $\underline{\boldsymbol{\theta}}$, which is straightforward.

4.3. Inequivalent variable placements

Recall that a variable placement is a mapping from the input and ancilla variables $\underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ onto the vertices V; the formula $\Phi(\underline{\boldsymbol{\theta}})$ in (16)-(21) can be built only after each $z_i \in \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ has been placed. In general there will be many such placements, but by exploiting symmetry and the automorphism group of G, we can reduce the number of placements that need be considered.

Let $\underline{\mathbf{v}} \stackrel{\text{def}}{=} (v_1, ..., v_{n+h})$ denote a variable placement, so v_i is the vertex of V onto which z_i is placed. Two variable placements $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}' \stackrel{\text{def}}{=} (v'_1, ..., v'_{n+h})$ are *equivalent* if there is a graph isomorphism ϕ of G that point-wise maps the input variables (x_i) in $\underline{\mathbf{v}}$ to the input variables in $\underline{\mathbf{v}}'$; that is, $v_i = \phi(v'_i)$ for all $i \leq n$. If $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ are equivalent, then a penalty function for $\underline{\mathbf{v}}$ can be transformed into a penalty function for $\underline{\mathbf{v}}'$ by applying ϕ. Therefore, in order to find a penalty function of maximal gap among all variable placements, it suffices to consider only inequivalent ones.

Example 12. Suppose we want to encode a penalty function with $n + h = 8$ variables into an 8-qubit Chimera tile. There are $8! = 40320$ candidate variable placements. However, the tile structure is highly symmetric: any permutation of $\underline{\mathbf{v}}$ that either

(i) swaps horizontal qubits with vertical qubits, or
(ii) maps horizontal qubits to horizontal qubits and vertical qubits to vertical qubits

is an automorphism. This fact can be exploited to reduce the number of candidate placements to only $\binom{7}{3} = 35$ as follows. Let $1, ..., 4$ and $5, ..., 8$ be the indexes of the horizontal and vertical qubits respectively. By (i), we assume w.l.o.g. that z_1 is mapped into an horizontal qubit, and by (ii) we assume w.l.o.g. that $v_1 = 1$. Next, consider some size-3 subset S of $\{v_2, ..., v_8\}$. By (ii), all placements that map S into the remaining 3 horizontal qubits and map $\{v_2, ..., v_8\} \backslash S$ into the vertical qubits are equivalent. Since there are $\binom{7}{3} = 35$ such subsets S, there are at most 35 inequivalent placements to consider.

This notion of equivalence of variable placements can be coarsened slightly by taking advantage of NPN-equivalence. We define variables x_1 and x_2 in a Boolean function F to be *NPN-symmetric* if swapping the variables, and negating some subset of variables, produces an equivalent formula. For example, consider $F(x_1, x_2, x_3) \stackrel{\text{def}}{=} x_3 \leftrightarrow (x_1 \wedge \neg x_2)$. Variables x_1 and x_2 are NPN-symmetric because $F(x_1, x_2, x_3) \leftrightarrow F(\neg x_2, \neg x_1, x_3)$. This symmetry defines an equivalence relation on the variables: for x_i and x_j the same equivalence class, there is a permutation and negation of the variables that does not change F but maps x_i to x_j while not permuting variables outside the equivalence class.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

We say that two variable placements $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ are *equivalent up to NPN-symmetry* if there is a graph isomorphism ϕ of G that maps the input variables in $\underline{\mathbf{v}}$ to the input variables in $\underline{\mathbf{v}}'$ up to NPN-symmetry classes. That is, for all $i \leq n$, there exists a $j \leq n$ such that x_i and x_j are NPN-symmetric and $v_i = \phi(v'_j)$. Again, penalty functions for $\underline{\mathbf{v}}$ and can be transformed into penalty functions for $\underline{\mathbf{v}}'$ and vice versa.

Example 13. Consider placing the function $\text{AND}(x_1, \ldots, x_4) = x_1 \wedge x_2 \wedge x_3 \wedge x_4$ with $h = 4$ auxiliary variables on the 8-qubit Chimera tile. From Example 12, it suffices to consider 35 variable placements. However, the variables $x_1, \ldots, x_4$ in AND are all NPN-symmetric. Therefore any two variable placements $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ that map the same number x_i's to horizontal qubits are equivalent, since there is an automorphism that will map the horizontal x_i's in $\underline{\mathbf{v}}$ to the horizontal x_i's in $\underline{\mathbf{v}}'$. Moreover, a placement mapping $k \leq 4$ of the x_i's to horizontal qubits is equivalent to one mapping $4-k$ of the x_i's to horizontal qubits, by swapping horizontal and vertical qubits. As a result, there are only 3 inequivalent variable placements to consider, in which 0, 1 or 2 of the x_i's are mapped to horizontal qubits.

One way to check for equivalent variable placements is to use vertex-colored graph isomorphisms. Two vertex-colored graphs (G, c) and (G', c') are *vertex-colored graph-isomorphic* if there is a permutation ϕ mapping $V(G)$ to $V(G')$ that preserves edges and maps every vertex of G to a vertex of the same color in G' (for all $v \in V$, $c'(\phi(v)) = c(v)$). Using a variable placement $\underline{\mathbf{v}}$ and NPN-symmetry, define a vertex-coloring c of G as follows:

$$c(g) = \begin{cases} s & \text{if } v_i = g \text{ and } x_i \text{ is in the } s\text{-th equivalence class of NPN-symmetry,} \\ 0 & \text{if } g \text{ is not in } \{v_1, \ldots, v_n\}. \end{cases}$$

Similarly define a vertex coloring c' for variable placement $\underline{\mathbf{v}}'$. From these definitions, $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ are equivalent up to NPN-symmetry if and only if the vertex colored graphs (G, c) and (G, c') are vertex-colored graph-isomorphic.

In practice, we can use the graph package NAUTY [44] to compute a canonical form for each vertex-colored graph and check if two are the same. NAUTY works with vertex-colored canonical forms natively as part of its graph isomorphism algorithm, and can compute canonical forms for graphs with thousands of vertices.

4.4. Placing variables & computing penalty functions via SMT/OMT($\mathcal{LRIA} \cup \mathcal{UF}$)

As an alternative to identifying equivalent variable placements, for small formulae $F(\underline{\mathbf{x}})$, 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 (§2.2). This works as follows.

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 $V \stackrel{\text{def}}{=} \{1, \ldots, n+h\}$, and we introduce a set of $n + h$ *integer* variables $\underline{\mathbf{v}} \stackrel{\text{def}}{=} \{v_1, \ldots, v_{n+h}\}$ such that $v_j \in V$ is (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 $\text{Distinct}(v_1, \ldots, v_{n+h})$ 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 $\text{Distinct}(v_1, \ldots, v_{n+h})$.

This rewrites the SMT($\mathcal{LRA}$) problem (16)-(19) into the $SMT(\mathcal{LRIA} \cup \mathcal{UF})$ problem (40)-(51) in Fig. 7. Equation (44) must be used if and only if we need an exact penalty function. (Notice that (47) 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 couplings c, but also the best placement $\underline{\mathbf{v}}$ of the variables into (the indexes of) the qubits.

Example 14. Consider $\underline{\mathbf{x}} \stackrel{\text{def}}{=} \{x_1, x_2, x_3\}$, $\underline{\mathbf{a}} \stackrel{\text{def}}{=} \{a_1\}$ and $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} (x_3 \leftrightarrow (x_1 \wedge x_2))$, and 4-qubit fraction of a tile with 2 horizontal and 2 vertical qubits. Let z_1, z_2, z_3 and z_4, denote x_1, x_2, x_3 and a_1 respectively, so that each v_j denotes the vertex into which z_j is placed. We consider the encoding (40)-(51), in particular we have that:

$$\begin{aligned} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} {} & \theta_0 + \mathsf{b}(v_1)x_1 + \mathsf{b}(v_2)x_2 + \mathsf{b}(v_3)x_3 + \mathsf{b}(v_4)a_1 + \\ & \mathsf{c}(v_1, v_2)x_1x_2 + \mathsf{c}(v_1, v_3)x_1x_3 + \mathsf{c}(v_1, v_4)x_1a_1 + \\ & \mathsf{c}(v_2, v_3)x_2x_3 + \mathsf{c}(v_2, v_4)x_2a_1 + \mathsf{c}(v_3, v_4)x_3a_1 \\ \text{Graph}() \stackrel{\text{def}}{=} {} & \mathsf{c}(1,2) = 0 \wedge \mathsf{c}(2,1) = 0 \wedge \mathsf{c}(3,4) = 0 \wedge \mathsf{c}(4,3) = 0 \end{aligned}$$

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

$$\Phi(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \mathsf{Range}(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \wedge \mathsf{Distinct}(\underline{\mathbf{v}}) \wedge \mathsf{Graph}() \tag{40}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n \mid F(\underline{\mathbf{x}}) = \top\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \geq 0) \tag{41}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n \mid F(\underline{\mathbf{x}}) = \top\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) = 0) \tag{42}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n \mid F(\underline{\mathbf{x}}) = \bot\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \geq g_{min}) \tag{43}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n \mid F(\underline{\mathbf{x}}) = \bot\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) = g_{min}) \tag{44}$$

where:

$$\mathsf{Range}(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \bigwedge_{1 \leq j \leq n+h} (1 \leq v_j) \wedge (v_j \leq n+h) \tag{45}$$

$$\wedge \bigwedge_{1 \leq j \leq n+h} (-2 \leq \mathsf{b}(j)) \wedge (\mathsf{b}(j) \leq 2) \tag{46}$$

$$\wedge \bigwedge_{1 \leq j \leq n+h} (\mathsf{c}(j,j) = 0) \wedge \bigwedge_{1 \leq i < j \leq n+h} (\mathsf{c}(i,j) = \mathsf{c}(j,i)) \tag{47}$$

$$\wedge \bigwedge_{1 \leq i < j \leq n+h} (-1 \leq \mathsf{c}(i,j)) \wedge (\mathsf{c}(i,j) \leq 1) \tag{48}$$

$$\mathsf{Distinct}(v_1, \ldots, v_{n+h}) \stackrel{\text{def}}{=} \bigwedge_{1 \leq i < j \leq n+h} \neg(v_i = v_j) \tag{49}$$

$$\mathsf{Graph}() \stackrel{\text{def}}{=} \wedge \bigwedge_{\substack{1 \leq i < j \leq n+h \\ \langle i,j \rangle \notin E}} (\mathsf{c}(i,j) = 0) \tag{50}$$

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \theta_0 + \sum_{1 \leq j \leq n+h} \mathsf{b}(v_j) \cdot z_j + \sum_{1 \leq i < j \leq n+h} \mathsf{c}(v_i, v_j) \cdot z_i \cdot z_j. \tag{51}$$

Fig. 7. SMT ($\mathcal{LRIA} \cup \mathcal{UF}$) encoding with automatic placement.

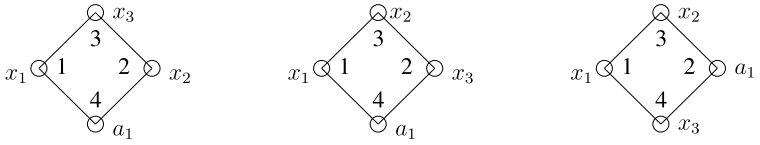


Fig. 8. 3 possible placements of $\underline{\mathbf{z}} \stackrel{\text{def}}{=} \{x_1, x_2, x_3\} \cup \{a_1\}$ into a 4-qubit tile fraction with 2 horizontal and 2 vertical qubits. All $4! = 24$ combinations are equivalent to one of them.

One possible solution is given in the following tables:

g	v_1	v_2	v_3	v_4
2	1	3	2	4

θ_0	$\mathsf{b}(v_1)$	$\mathsf{b}(v_2)$	$\mathsf{b}(v_3)$	$\mathsf{b}(v_4)$
	$\mathsf{b}(1)$	$\mathsf{b}(3)$	$\mathsf{b}(2)$	$\mathsf{b}(4)$
5/2	$-1/2$	$-1/2$	1	0

$\mathsf{c}(v_1, v_2)$	$\mathsf{c}(v_1, v_3)$	$\mathsf{c}(v_1, v_4)$	$\mathsf{c}(v_2, v_3)$	$\mathsf{c}(v_2, v_4)$	$\mathsf{c}(v_3, v_4)$
$\mathsf{c}(1,3)$	$\mathsf{c}(1,2)$	$\mathsf{c}(1,4)$	$\mathsf{c}(3,2)$	$\mathsf{c}(3,4)$	$\mathsf{c}(2,4)$
1/2	0	-1	-1	0	-1

which corresponds to the placing in Fig. 8 (center).

4.4.1. Exploiting symmetries

When using an SMT/OMT solver to search for penalty functions across all variable placements as in (40)-(51), we may restrict the search space by considering only one variable placement from each equivalence class under the automorphisms of G.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

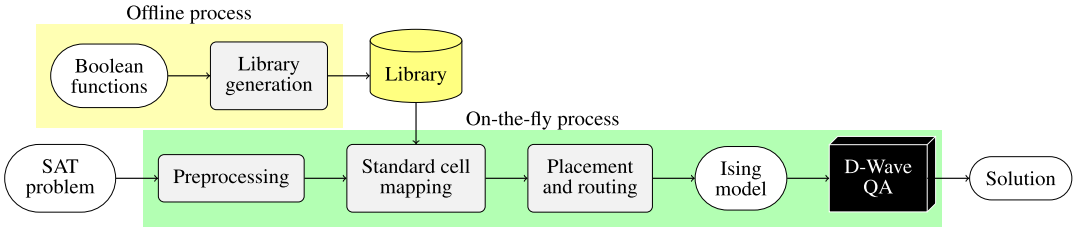


Fig. 9. Graph of the encoding process.

Example 15. In Example 12, when encoding a penalty function with $n+h=8$ variables into a Chimera tile, automorphisms reduced the number of variable placements under consideration from $8!=40320$ to $\binom{7}{3}=35$. We can force the SMT/OMT solver to restrict the search to only 35 maps by adding the following constraint to (40)-(51), consisting into the disjunction of 35 cubes, each representing one placement.

$$
\begin{aligned}
&(\overbrace{v_1=1}^{Fixed}\wedge\overbrace{v_2=2\wedge v_3=3\wedge v_4=4}^{\text{size-3 subset of } \{v_2,\dots,v_8\} \text{ mapped to horizontal qubits}}\wedge\overbrace{v_5=5\wedge v_6=6\wedge v_7=7\wedge v_8=8}^{\text{complement of the previous subset mapped to vertical qubits}})\vee\\
&(v_1=1\wedge v_2=2\wedge v_3=3\wedge v_5=4\wedge v_4=5\wedge v_6=6\wedge v_7=7\wedge v_8=8)\vee\\
&\dots\\
&(v_1=1\wedge v_6=2\wedge v_7=3\wedge v_8=4\wedge v_2=5\wedge v_3=6\wedge v_4=7\wedge v_5=8).
\end{aligned}
$$

If we add this constraint, the first conjunction in (45) can be dropped.

Example 16. In Example 14 we have $4!=24$ possible placements on to a tile of 2 horizontal and 2 vertical qubits. If we exploit symmetries as above, we have only $\binom{3}{1}=3$ inequivalent placements, which are described in Fig. 8. These can be obtained by adding the constraint:

$$
\begin{aligned}
&(v_1=1\wedge v_2=2\wedge v_3=3\wedge v_4=4)\vee\\
&(v_1=1\wedge v_3=2\wedge v_2=3\wedge v_4=4)\vee\\
&(v_1=1\wedge v_4=2\wedge v_2=3\wedge v_3=4).
\end{aligned}
$$

5. Encoding larger Boolean formulas

As pointed out in Section 3.2, encoding large Boolean functions using the SMT formulations of the previous section is computationally intractable, as the number of constraints in the model increases roughly exponentially with the number of variables in the Boolean function. In this section, we describe the natural approach of pre-computing a library of encoded Boolean functions and rewriting a larger Boolean function $F(\underline{\mathbf{x}})$ as a set of pre-encoded ones $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$. The penalty functions $P_{F_k}(\underline{\mathbf{x}}^k,\underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ for these pre-encoded functions may then be combined using chains as described in Section 3.4. This schema is shown in Fig. 9. In terms of QA performance, this method has been shown experimentally to outperform other encoding methods for certain problem classes [45]. We will describe each of the stages in turn (see also [34,45,46]).

5.1. Pre-encoding

In this stage, we find effective encodings of common small Boolean functions, using the SMT methods in Section 4 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 needs only be performed once for each NPN-inequivalent Boolean function.

Note that there exist many different penalty functions $P_F(\underline{\mathbf{x}},\underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ for any small Boolean function $F(\underline{\mathbf{x}})$. Penalty functions with more qubits may have larger gaps, but using those functions may result in longer chains, so it is not always the case that larger gaps lead to better QA hardware performance. Choosing the most appropriate function may be a nontrivial problem. A reasonable heuristic is to choose penalty functions with gaps of similar size to the gap associated with a chain, namely $g_{min}=2$.

5.2. Preprocessing

Preprocessing, or Boolean formula minimization, consists of simplifying the input formula $F(\underline{\mathbf{x}})$ to reduce its size or complexity. While not strictly necessary, it not only improves QA performance by reducing the size of $P_F(\underline{\mathbf{x}},\underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ but also

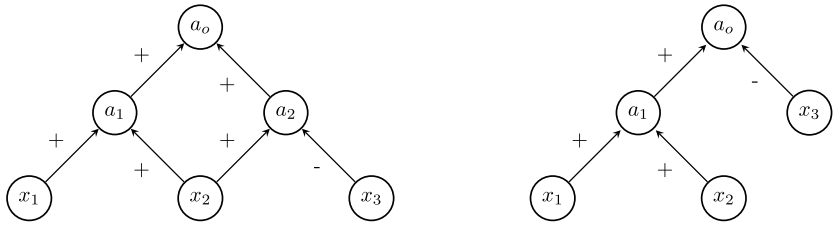


Fig. 10. Two And-Inverter Graphs representing the function $F(\underline{\mathbf{x}}) = x_1 \wedge x_2 \wedge \neg x_3$.

reduces the computational expense of the encoding process. Moreover, the graphical representation commonly used in preprocessing, the *AND-Inverter Graph* (AIG), is necessary for the subsequent phase of encoding.

An AIG encodes $F(\underline{\mathbf{x}})$ as a series of 2-input AND gates and negations. More precisely, a directed acyclic graph D on vertex set $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}} = (x_1, \ldots, x_n, a_1, \ldots, a_m)$ is an *AIG representing* $F(\underline{\mathbf{x}})$ if it has the following properties:

1. Each x_i has no incoming arcs and each a_k has 2 incoming arcs (the *inputs* to a_k), and there is a unique a_o with no outgoing arcs (the *primary output*).
2. Each arc $z \to a$ is labeled with a sign $+$ or $-$ indicating whether or not z should be negated as an input to a; define a literal $l_a(z) = z$ for an arc with sign $+$ and $l_a(z) = \neg z$ for an arc with sign $-$.
3. For each node a_k with arcs incoming from z_1 and z_2, there is an AND function $A_k(a_k, z_1, z_2) = a_k \leftrightarrow l_{a_k}(z_1) \wedge l_{a_k}(z_2)$, such that

$$F(\underline{\mathbf{x}}) \leftrightarrow \bigwedge_{k=1}^{m} A_k(\underline{\mathbf{z}}) \wedge (a_o = \top). \tag{52}$$

For example, the function $F(\underline{\mathbf{x}}) = x_1 \wedge x_2 \wedge \neg x_3$ is represented by both of the And-Inverter Graphs in Fig. 10.

There are many And-Inverter Graphs representing a given $F(\underline{\mathbf{x}})$. Is $F(\underline{\mathbf{x}})$ is in CNF form, we can construct an AIG by rewriting each OR clause as an AND function via De Morgan's Law, and then rewriting each AND function with more than 2 inputs as a sequence of 2-input AND functions.

Preprocessing is a well-studied problem with mature algorithms available [47,48]; here, we use *DAG-aware minimization* as implemented by the logic optimizer ABC.[14] DAG-aware minimization attempts to find an AIG with a minimal number of nodes by repeatedly identifying a small subgraph that can be replaced with another, smaller subgraph without changing the truth assignments of $F(\underline{\mathbf{x}})$.

More precisely, a *cut* C of node z in D is a subset of vertices such that every directed path from an input x_i to z must pass through C. The subgraph of D induced by all paths from C to z is a candidate to be replaced by a smaller subgraph, since the Boolean value of z is determined by C. We call this value of z as a function of C the *Boolean function represented by* C. Cut C is *k-feasible* if $|C| \leq k$ and *non-trivial* if $C \neq \{z\}$. For fixed k, there is an $O(n)$-time algorithm to identify all k-feasible cuts in an AIG: traverse the graph from the inputs $\underline{\mathbf{x}}$ to the primary output, identifying the k-feasible cuts of node a_i by combining k-feasible cuts of a_i's inputs. During traversal, DAG-aware minimization identifies a 4-feasible cut C and replaces the subgraph induced by C with the smallest subgraph representing the same Boolean function. (There are 222 NPN-inequivalent 4-input Boolean functions, and smallest subgraph representing each one is pre-computed.) See [49] for more details.

5.3. Standard cell mapping

In the standard cell mapping phase, $F(\underline{\mathbf{x}})$ is decomposed into component functions $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ that are available in the library of penalty functions. 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 [34]). However, more advanced techniques have been devised in the digital logic synthesis literature. *Technology mapping* is the process of mapping a technology-independent circuit representation to the physical gates used in a digital circuit [49,50]. 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 QAs, but minimization is important to simplify the placement and routing phase that follows.

In order to find an efficient decomposition, a technology mapping algorithm takes as input costs for small $F_k(\underline{\mathbf{x}}^k)$ and attempts to minimize the sum of the costs of the components in $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$. We define the cost of F_k to be the number of qubits used by the penalty model P_{F_k}, so that the cost of $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ is the total number of qubits used to represent $F(\underline{\mathbf{x}})$, prior to adding chains.

[14] See https://github.com/berkeley-abc/abc and https://people.eecs.berkeley.edu/~alanmi/abc/.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Here, we apply the technology mapping algorithm in [49]: the idea is to decompose the AIG representing $F(\underline{\mathbf{x}})$ into a collection of cuts such that each cut represents a small function $F_k(\underline{\mathbf{z}}^k)$ that can be found in the penalty library. A *mapping* M of an AIG D is a partial function that maps a node a_i of D to a non-trivial, k-feasible cut $M(a_i)$. We say a_i is *active* when $M(a_i)$ is defined and *inactive* otherwise. Mapping M is *proper* if:

1. the primary output a_o is active;
2. if a_i is active; then every $a_j \in M(a_i)$ is active; and
3. if $a_j \neq a_o$ is active; then $a_j \in M(a_i)$ for some active a_i.

For each active node a_k in a proper mapping M, there is a Boolean function $F_k(\underline{\mathbf{z}}^k)$ represented by the cut $M(a_k)$, and the original Boolean function $F(\underline{\mathbf{x}})$ decomposes as

$$F(\underline{\mathbf{x}}) \leftrightarrow \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{z}}^k) \wedge (a_o = \top).$$

Therefore, choosing k-feasible cuts with small k, proper mappings provide decompositions of $F(\underline{\mathbf{x}})$ into small Boolean functions that can be found in the penalty library. One example of a proper mapping is the trivial mapping, in which each a_i is mapped to the cut consisting of its two input nodes. Under the trivial mapping, $F(\underline{\mathbf{x}})$ is decomposed into a collection 2-input AND's.

The algorithm in [49] iteratively refines mapping M in order to improve the cost of the decomposition, in the following way. For each node a_i, maintain a list $L(a_i)$ of k-feasible cuts, ordered by their cost. (The cost of a cut is a function of the cost of the Boolean function it represents, taking into account the anticipated recursive effects of having a new set of active nodes: see [49] for details.) Traverse the graph from inputs $\underline{\mathbf{x}}$ to primary output a_o. At each a_i, first update the costs of the cuts in $L(a_i)$ based on the changes to the costs of earlier nodes in the traversal. Next, if a_i is active and the current cut $M(a_i)$ is not the cut in $L(a_i)$ of lowest cost, update $M(a_i)$. To do this, first inactivate a_i (which recursively inactivates nodes in $M(a_i)$ if they are no longer necessary) and then reactivate a_i (which reactivates nodes in $M(a_i)$, also recursively). This process of refining the mapping by traversing the graph is repeated several times.

Given the connectivity of the Chimera hardware graph, a natural choice is to decompose into Boolean functions that can be modeled with a single 8-qubit tile. In particular all 3-input, 1-output Boolean functions (all 3-feasible cuts) can be modeled in one tile.

Example 17. Consider the XOR function $F(\underline{\mathbf{x}}) \stackrel{\text{def}}{=} x_6 \leftrightarrow (x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5)$. This function cannot be encoded into a single 8-qubit Chimera tile. However, $F(\underline{\mathbf{x}})$ is equivalent to the decomposition $F_1(\underline{\mathbf{z}}) \wedge F_2(\underline{\mathbf{z}})$ with

$$F_1(\underline{\mathbf{z}}) \stackrel{\text{def}}{=} a_0 \leftrightarrow (x_1 \oplus x_2 \oplus x_3),$$

$$F_2(\underline{\mathbf{z}}) \stackrel{\text{def}}{=} x_6 \leftrightarrow (a_0 \oplus x_4 \oplus x_5),$$

where each of F_1 and F_2 maps to a unit tile individually.

5.4. Placement and routing

Once $F(\underline{\mathbf{x}})$ is decomposed into smaller functions $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$ with penalty functions $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$, it remains to embed the entire formula onto the QA hardware as in equation (11). 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 equivalence constraints with penalty functions of the form $1 - x_i x_i'$). Both placement and routing are very well-studied in design of digital circuits [51]. Nevertheless, this stage is a computational bottleneck for encoding large Boolean functions.

5.4.1. Placement

During placement, chain lengths can be minimized by placing penalty functions that share common variables close together. Current QA processors have a nearly 2-dimensional structure, which lets us measure distance between variables using planar coordinates. (For example, for the 2048-qubit Chimera graph in Fig. 2, define the planar coordinates of a unit cell to be its row and column index in the 16×16 grid.) One common objective function from digital circuit design is "half-perimeter wire length" [52]. Define the *location* of a Boolean function $F_k(\underline{\mathbf{x}}^k)$ to be the subgraph of G onto which $F_k(\underline{\mathbf{x}}^k)$ is placed, and define a placement function $p : \{1, \ldots, K\} \to \mathbb{R}^2$ which maps each k to the planar coordinates $p(k) = (a_k, b_k)$ of the location of $F_k(\underline{\mathbf{x}}^k)$. The *half-perimeter wire length (HPWL)* of a variable x_i is the total length and width of the smallest box that can be drawn around the locations of functions containing x. That is, for $S_i = \{k : x_i \in \underline{\mathbf{x}}^k\}$,

$$HPWL(x_i) \stackrel{\text{def}}{=} (\max_{k \in S_i} a_k - \min_{k \in S_i} a_k) + (\max_{k \in S_i} b_k - \min_{k \in S_i} b_k).$$

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

A placement algorithm attempts to find a placement that minimizes $\sum_{i=1}^{n} HPWL(x_i)$.

Heuristic methods for placement include simulated annealing [53], continuous optimization [54], and recursive min-cut partitioning [55]. 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. For example, in some algorithms, a placement is optimized on the assumption that the resulting routing problem is feasible (possibly by expanding the planar area made available for routing). This assumption may not necessarily hold using a fixed QA hardware graph of limited size and connectivity. If unit cells are packed tightly with Boolean functions, then there will be few remaining qubits available for routing. On the other hand, reserving too many qubits for routing will have a negative impact on hardware performance in the form of longer chains.

In the experiments in §7 we made use of mPL,[15] a publicly available academic placement tool [54]. mPL is multilevel method in which the placement problem is repeatedly coarsened (so that several P_{F_k} are clustered and treated as one), placed, and uncoarsened with local improvements. At the coarsest level, placement is performed using a customized non-linear programming algorithm which maps penalty functions to real coordinates minimizing a quadratic distance function between shared variables.

5.4.2. Routing

During routing, literals are chained together using as few qubits possible; this problem may be formalized as follows. Assume a single variable x_i has been assigned to a set of vertices $T_i \subseteq V$, its *terminals*, during the placement of small Boolean functions. To create a valid embedding, the chain of vertices representing x_i, call it C_i, must contain T_i and induce a connected subgraph in G. Finding C_i with a minimum number of vertices is an instance of the *Steiner tree problem* [56] and C_i is a Steiner tree. Given variables $(x_1, \ldots, x_n)$ assigned to terminals $(T_1, \ldots, T_n)$, the *routing* problem demands a set of chains $(C_1, \ldots, C_n)$ such that each C_i contains T_i, every chain is connected, and all chains are pairwise disjoint. Among routing solutions, we try to minimize the total number of vertices of G used or the size of the largest chain.

Routing to minimize the total number of vertices used is NP-hard, but polynomial-time approximation algorithms exist [57]. In practice, heuristic routing algorithms scale to problem sizes much larger than current QA architectures [58–62].

Routing in the current context differs from routing used in digital circuit design in the sense that vertices (qubits) are the sparse resource that variables compete for, rather than edges. As a result, we make use vertex-weighted Steiner tree algorithms rather than edge-weighted ones. This makes the problem harder, as the edge-weighted Steiner tree problem is (1.39)-approximable in polynomial time [63], while vertex-weighted Steiner-tree is only $(\log k)$-approximable for k terminals in polynomial time unless P=NP [64]. Nevertheless, in practice, simple 2-approximation algorithms for edge-weighted Steiner tree such as the MST algorithm [65] or Path Composition [66] also work very well for the vertex-weighted problem. In this section, we describe a modification of the routing algorithm BonnRoute [66] for vertex-weighted Steiner trees.

We first solve a continuous relaxation of the routing problem called *min-max resource allocation*. Given a set of vertices $C \subseteq V$, the *characteristic vector* of C is the vector $\chi(C) \in \{0,1\}^{|V|}$ such that $\chi(C)_v = 1$ if $v \in C$ and 0 otherwise. Let H_i be the convex hull of all characteristic vectors of Steiner trees of T_i in G. Then the *min-max resource allocation* problem for terminals $T_1, \ldots, T_n$ is to minimize, over all $z_i \in H_i$, $i \in \{1, \ldots, n\}$,

$$\lambda(z_1, \ldots, z_n) \stackrel{\text{def}}{=} \max_{v \in V} \sum_{i=1}^{n} (z_i)_v.$$

The vertices v are the *resources*, which are allocated to *customers* $(z_1, \ldots, z_n)$.[16] To recover the routing problem, note that if each z_i is a characteristic vector of a single Steiner tree, then $\sum_{i=1}^{n} (z_i)_v$ the number of times vertex v is used in a Steiner tree. In that case, $\lambda(x) \leq 1$ if and only if the Steiner trees are a solution to the routing problem.

To solve the min-max resource allocation, we iteratively use a weighted-Steiner tree approximation algorithm to generate a probability distribution over the Steiner trees for each x_i. After a Steiner tree is generated, the weights of the vertices in that Steiner tree are increased to discourage future Steiner trees from reusing them (see Algorithm 1 for details). This algorithm produces good approximate solutions in reasonable time. More precisely, given an oracle that computes vertex-weighted Steiner tree approximations within a factor σ of optimal, for any $\omega > 0$ Algorithm 1 computes a $\sigma(1+\omega)$-approximate solution to min-max resource allocation problem using $O((\log|V|)(n+|V|)(\omega^{-2} + \log\log|V|))$ calls to the oracle [67].

Once a solution to the min-max resource allocation has been found, we recover a solution to the original routing problem by formulating an integer linear program (IP), which may be solved via OMT($\mathcal{LRA}$).[17] For each Steiner tree S_i with non-zero probability in the distribution returned from min-max resource allocation, define a binary variable as follows:

[15] Available at http://cadlab.cs.ucla.edu/cpmo/.

[16] The original BonnRoute algorithm uses min-max resource allocation with edges rather than vertices as resources.

[17] The original BonnRoute algorithm uses randomized rounding to recover a routing solution from min-max resource allocation, but at current QA hardware scales this is not necessary.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Algorithm 1 BonnRoute Resource Sharing Algorithm [66].

Require: Graph G, Steiner tree terminals $\{T_1, \ldots, T_n\}$, number of iterations t, weight penalty $\alpha > 1$
Ensure: For each i, a probability distribution p_{i,S_i} over all Steiner trees S_i for terminals T_i

function BonnRoute(G,$\{T_1, \ldots, T_n\}$)
  for each $v \in V(G)$ **do**
    $w_v \leftarrow 1$
  for each Steiner tree S_i for terminals T_i, $i \in [n]$ **do**
    $z_{i,S_i} \leftarrow 0$
  for j from 1 to t **do**
    for each $i \in [n]$ **do**
      Find a Steiner tree S_i for terminals T_i with vertex-weights w_v
      $z_{i,S_i} \leftarrow z_{i,S_i} + 1$
      $w_v \leftarrow w_v * \alpha$ for all $v \in S_i$
  Return $p_{i,S_i} \leftarrow z_{i,S_i}/t$

$$x_{i,S_i} = \begin{cases} 1, & \text{if } S_i \text{ is the selected Steiner tree for variable } i; \\ 0, & \text{otherwise.} \end{cases}$$

Then minimize the number of qubits selected, subject to selecting one Steiner tree for each i and using each vertex at most once. That is,

$$\begin{aligned} \min \quad & \sum_i \sum_{S_i} |S_i| x_{i,S_i} \\ \text{s.t.} \quad & \sum_{S_i} x_{i,S_i} = 1 \text{ for all } i \\ & x_{i,S_i} + x_{j,S_j} \leq 1 \text{ for all } S_i, S_j \text{ s.t. } S_i \cap S_j \neq \emptyset. \end{aligned}$$

When applying routing to the Chimera graph, because of the symmetry within each unit tile, it is convenient to work with a *reduced* graph in which the horizontal qubits in each unit tile are identified as a single qubit, and similarly for the vertical qubits. As a result the scale of the routing problem is reduced by a factor of 4. This necessitates the use of vertex capacities within the routing algorithm (each reduced vertex has a capacity of 4), and variables are assigned to individual qubits within a tile during a secondary, detailed routing phase.

In the digital circuit literature, the placement and routing stages of embedding are typically performed separately. However, because of current limited number of qubits and the difficulty in allocating them to either placement or routing, a combined place-and-route algorithm can be more effective. This approach is discussed in detail in [45].

6. Related work

There have been several previous efforts to map specific small Boolean functions (usually in the guise of constraint satisfaction problems) to Ising models. Most of those mappings have been ad hoc, but some were more systematic (beyond [34] and [45] as previously discussed). Lucas [68] and Chancellor et al. [69] developed Ising models for several specific NP-hard problems, while Su et al. [46] and Pakin [70,71] decomposed Boolean functions into common primitives.

There have also been several attempts to map large Boolean functions or more generally large constrained Boolean optimization problems to D-Wave hardware. Most of these efforts (e.g. [72–80]) have used global embedding, in which an entire Ising model is minor-embedded heuristically [42] or a fixed embedding is used [40,41]. However Su et al. [46] used a general place-and-route approach, while Trummer et al. [81], Chancellor et al. [69], Zaribafiyan et al. [41], and Andriyash et al. [82] used a placement approach optimized for the specific constraints at hand.

The main new contributions of this paper to the mapping problem, beyond [34] and [45], are:

- a rigorous formalization of the mapping problem in terms of penalty functions and SMT/OMT formulations (§3, §4.1);
- improvements to the efficiency of SMT solutions using variable elimination and symmetries (§4.2, §4.3);
- using SMT/OMT($\mathcal{LRIA} \cup \mathcal{UF}$) to combine variable placement and penalty function generation (§4.4);
- the use of AND-Inverter Graph preprocessing and standard cell mapping to decompose large Boolean functions (§5.2, §5.3).

Looking at SAT instances in particular, there have been at least two previous attempts at benchmarking D-Wave hardware performance: McGeoch et al. [83] and Santra et al. [84] looked at (weighted) Max2SAT problems, and Douglass et al. [85] and Pudenz et al. [86] looked at SAT problems with the goal of sampling diverse solutions. Farhi et al. [87] and Hen and Young [88] studied the performance of quantum annealing on SAT problems more generally. The applicability of QAs for various SAT formulations has also been discussed in [89,90]. The distinguishing feature of the SAT problems considered in

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

this paper is the design of "small but hard" SAT problems, which accommodate the limited connectivity and size of current QA hardware while still challenging conventional SAT solvers.

7. Preliminary experimental evaluation

We have implemented and made publicly available prototype encoders built on top of the SMT/OMT tool OptiMathSAT [33]. In particular each SATtoIsing-specific step outlined in Fig. 9 has been implemented as a Python library. For preprocessing we rely on the ABC tool suite [91]. The same software is capable of performing technology mapping, though a Python version is available in the techmapping library.[18] Finally the placeandroute library[19] performs the combined placement and routing step. Regarding the off-line part of the process, the gatecollector library[20] extracts the most common gates in a dataset of functions and generates a function library in the ABC-compatible genlib format. The pfencoding library[21] is then used to call OptiMathSAT to encode them for later use. Currently the most expensive step in the on-the-fly process is the placement and routing step. In the current setup we use ≈20 minutes on a Intel i7-5600U CPU when we encode the problems used in the experimental evaluation. The software run-time is heavily tunable in order to trade off efficiency and effectiveness of the place-and-route process.

We offer preliminary empirical validation of the proposed methods for solving SAT via SATtoIsing encoding by evaluating the performance of D-Wave's 2000Q system in solving certain hard SAT problems (§7.1); we perform a similar evaluation also on MaxSAT problems (§7.2), despite the limitations highlighted in §3.3.

This task is subject to some limitations. First, we require instances that can be entirely encoded in a quantum annealer of 2000 qubits (although algorithms for solving much larger constraint satisfaction problems have been proposed; see [34,45]). Furthermore, SAT solvers are already quite effective on the average case, so we need concrete worst-case problems. Another important consideration in solving [Max]SAT instances is that the QA hardware cannot be made aware of the optimality of solution; for example, the algorithm cannot terminate when all clauses in a SAT problem are satisfied. In this way, QA hardware behaves more like an SLS solver than a CDCL-based one. To this extent, and in order to evaluate the significance of the testbed, we solved the same problems with the state-of-the-art UBCSAT SLS SAT solver using the best performing algorithm, namely SAPS [9]. UBCSAT was run on a computer using a 8-core Intel® Xeon® E5-2407 CPU, at 2.20 GHz.

Remark 1. The results reported in this section are not intended as a performance comparison between D-Wave's 2000Q system and UBCSAT, or any other classic computing tool. 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 parallel processing. In particular, we do not include the cost of encoding in our D-Wave timings, and this time is often greater than that required to solve the encoded problem itself (though for some applications such as encoding a Boolean circuit, encoding may be treated as a one-time preprocessing cost). Instead, we aim to provide an empirical assessment of QA's potential for [Max]SAT solving, based on currently available systems.

Reproducibility of results To make the results reproducible to those who have access to a D-Wave system, we have set a website where experimental data, problem files, translation files, demonstration code and supplementary material can be accessed.[22] Notice that public access to a D-Wave 2000Q machine is possible through D-Wave's Leap cloud service.[23]

7.1. SAT

Choosing the benchmark problems In order to provide a significant empirical evaluation, and due to the limitations in size and connectivity of current QA systems, we require SAT problems which have a low number of variables but are nevertheless hard for standard SAT solvers.

To this end we chose and modified the tool sgen [92], which has been used to generate the smallest unsolvable problems in recent SAT competitions. The problems share a structure that is suited for the problem embedding, as it contains multiple clones of slightly complex constraints, and even problems with few hundreds variables are considerably hard. The sgen family of random generators received many improvements over the years, but the method to generate satisfiable instances has remained the same [93,94]. sgen works by setting cardinality constraints over different partitions of the variable set. The generator operates as follows:

1. The user decides the number of Boolean variables in the problem.
2. The tool partitions the variable set into sets of 5 elements.

[18] Available at https://bitbucket.org/StefanoVt/tech_mapping.
[19] Available at https://bitbucket.org/StefanoVt/placeandroute.
[20] Available at https://bitbucket.org/StefanoVt/gatecollector.
[21] Available at https://bitbucket.org/StefanoVt/pfencoding.
[22] https://bitbucket.org/aqcsat/aqcsat.
[23] https://cloud.dwavesys.com/leap/.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

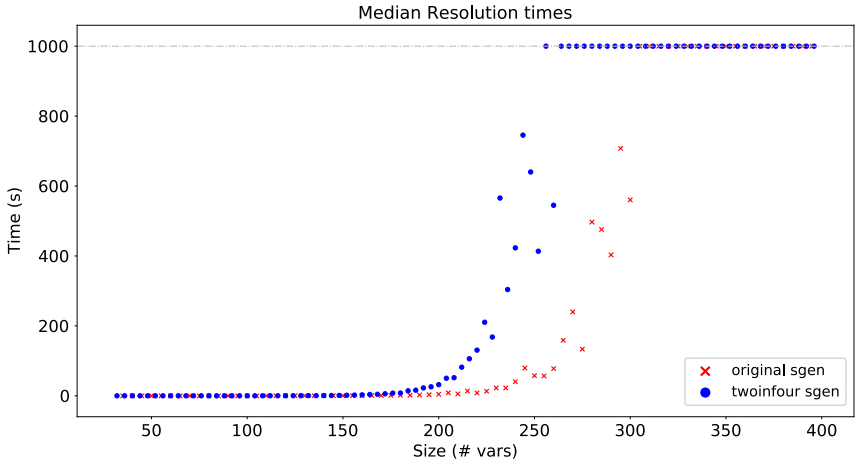


Fig. 11. Median times for the best-performing SLS algorithm on two different variants of the SGEN problem on UBCSAT (SAPS). Timeout is marked with a gray line. The figure report times on a computer with a 8-core Intel® Xeon® E5-2407 CPU, at 2.20 GHz.

Table 1
(a) Number of SATtolsing problem instances (out of 100) solved by the QA hardware using 5 samples [resp. 10 and 20] 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, [resp. 100 µs and 200 µs] of anneal time per instance respectively. 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 2.20 GHz.

D-Wave 2000Q				
Problem size	# solved 5 samples	# solved 10 samples	# solved 20 samples	% optimal samples
32 vars	100	100	100	97.4
36 vars	100	100	100	96.4
40 vars	100	100	100	94.8
44 vars	100	100	100	93.8
48 vars	100	100	100	91.4
52 vars	100	100	100	93.4
56 vars	100	100	100	91.4
60 vars	100	100	100	88.2
64 vars	100	100	100	84.6
68 vars	100	100	100	84.4
72 vars	98	100	100	84.6
76 vars	99	99	100	86.6
80 vars	100	100	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)

3. For satisfiable problem instances, the desired solution contains exactly one true variable for each subset. For each subset we guarantee that at most one variable is true (10 2-CNF clauses).
4. The partition is shuffled. The tool ensures that each new subset contain exactly one true variable, and minimizes the similarity with the previous partition.
5. For each new subset we ensure that at least one variable is true (a single CNF clause).
6. The previous two steps are repeated one more time, further restricting the solution space.

In Fig. 11 (red plot) we can see how UBCSAT SAPS performs on these random SGEN problems. Notice that with $>$ 300 variables the solver reaches the timeout of 1000 s. In our experiments, we modify the tool by using exactly-2-in-4 constraints on partitions with sets of size 4 with exactly two true variables per subset. This kind of constraint has a more efficient embedding and the modified problems are harder (see Fig. 11, blue plot, where UBCSAT reaches the timeout with $>$ 270 variables).

Experiments and results To solve these SAT instances, we encode and embed them as in §4-§5 and then draw a fixed number of samples/instance (5, 10, 20) at an annealing rate of 10 µs per sample. Table 1(a) shows the results from the D-Wave 2000Q QA hardware.

The QA hardware solves almost all problems with 5 samples (i.e. within 50 µs of total anneal time), and all of them with 20 samples (i.e. within 200 µs of total anneal time), and the rates of sampling optimal solutions remain relatively stable at this scale of problem.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Table 2
(a) Number of MaxSATtoIsing 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 2.90 GHz × 4 processor. The solvers gw2sat [95], rots [96], and novelty [97] are as implemented in UBCSAT [9]. 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	69.0
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	13.0
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	0.020	0.018	0.034	0.039
36 vars	0.025	0.022	0.043	0.060
40 vars	0.039	0.029	0.056	0.119
44 vars	0.049	0.043	0.070	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	0.130	0.167	1.884
64 vars	0.527	0.159	0.207	4.272
68 vars	0.652	0.210	0.270	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	0.430	95.057

(b)

In order to evaluate the significance of the testbed, we also report the results of solving the same problems with the UBCSAT SLS SAT solver using SAPS [9]. Remark 1 applies here. 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. (See also Fig. 11.)

7.2. Weighted MaxSAT solving and sampling

Choosing the benchmarks To demonstrate the performance of the QA hardware 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. More precisely:

1. Beginning with the 2-in-4-SAT instances of the previous section, we remove one of the partitions of the variable set, and change one 2-in-4 constraint to 1-in-4. (This makes the SAT problem unsatisfiable: for an n variable problem, the first partition demands exactly $n/2$ true variables, while the second demands exactly $n/2-1$.)
2. We change the SAT problem into a weighted MaxSAT problem by assigning existing constraints a soft weight of 3 and randomly assigning each variable or its negation a soft constraint of weight 1.
3. We repeatedly generate MaxSAT instances of this form, until we find an instance in which the optimal solution has exactly one violated clause of weight 3 and at least $n/3$ violated clauses of weight 1, and at least 200 distinct optimal solutions exist.

As discussed in §3.3, determining an appropriate gap for chains in MaxSAT problems is more complicated than for SAT problems, and finding the smallest viable chain gap may be difficult analytically. However, a gap may be found experimentally by sweeping over a range of values and choosing one that results in optimal performance. Chain gaps that are too small result in a large number of broken chains, while chain gaps that are too large result in gaps for problem constraints that are smaller than the noise levels of the hardware, yielding solutions that are far from optimal. For the MaxSAT experiments in this section, the chosen chain gap was always in the range $g_{chain} \in [2, 6]$ (relative to penalty functions $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ with $\theta_i \in [-2, 2]$, $\theta_{ij} \in [-1, 1]$).

Experiments and results 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-performing SLS MaxSAT solvers. The QA hardware solves almost all problems with 100 samples/instance (i.e. within 1 ms of anneal time). Remark 1 also applies here. 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 QAs will likely come from applications which require many solutions rather than a single optimum.

To this extent, Table 3 considers generating distinct optimal solutions. For each solver and problem size, the table indicates the number of distinct solutions found in 1 second, averaged across 100 problem instances of that size. For the smallest problems, 1 second is sufficient for all solvers to generate all solutions, while the diversity of solutions found varies

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

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 μs 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	607.0	579.9
40 vars	1007.9	922.0
44 vars	1322.6	1066.6
48 vars	1555.4	1111.8
52 vars	3229.0	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	607.0	606.9	606.9	606.8
40 vars	1007.7	1006.3	1005.3	1005.0
44 vars	1313.8	1307.1	1311.7	1255.5
48 vars	1515.4	1510.7	1504.9	1320.5
52 vars	2707.5	2813.0	2854.6	1616.2
56 vars	2021.9	2106.2	2186.6	969.8
60 vars	2845.6	3061.7	3289.0	904.4
64 vars	3100.0	4171.0	4770.0	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	1716.0	2059.2	116.4
80 vars	772.2	1111.1	1363.9	66.7

(b)

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.

Alternative penalty functions Different penalty functions can result in different QA performance, even when those penalty functions have the same gap between ground and excited states. As an example of this, we describe another set of MaxSAT instances which result in better performance on the D-Wave 2000Q processor relative to classical solvers, even though the penalty functions they use are less theoretically justified.

We call these instances "unbiased" to distinguish them from the MaxSAT instances of the previous section. They are generated as follows. Beginning with the SGEN 2-in-4-SAT instances, we first change one 2-in-4 constraint to 1-in-4, making the SAT problem unsatisfiable. We then remove 5 constraints from one partition of the variable set. This increases the total number of optimal solutions. Finally, we treat the resulting constraints as a MaxSAT problem in which each 1-in-4 or 2-in-4 constraint has the same weight. Despite having many solutions, these problems become difficult for MaxSAT solvers with a relatively small number of variables.

When solving these instances, we represent each 2-in-4-MaxSAT constraint by the following penalty function: $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 4 + x_1x_2 + x_1x_4 + x_2x_3 + x_3x_4 - x_1a_1 - x_2a_2 + x_3a_1 + x_4a_2$. This model satisfies:

$$\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = \begin{cases} 0, & \sum_i x_i = 0; \\ 2, & |\sum_i x_i| = 1; \\ 8, & |\sum_i x_i| = 2. \end{cases}$$

Because the unsatisfiable states $|\sum_i x_i| = 1$ and $|\sum_i x_i| = 2$ have different minimal energy configurations, this is not an *exact* penalty function as required for MaxSAT as in (21). Nevertheless, this model performs well in practice, because for the unbiased MaxSAT instances only configurations with $|\sum_i x_i| \leq 1$ are of interest.

Table 4 summarizes the performance of the D-Wave hardware and classical solvers in finding an optimal solution for the unbiased MaxSAT instances. It is instructive to compare these results to the "biased" MaxSAT instances in Table 2. The unbiased instances require more time for the best classical solvers to solve, yet result in better D-Wave hardware performance, despite the fact that the penalty function used is not exact.

8. 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 are 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 quantum annealing 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 functions.* For large Boolean functions, generating penalty functions directly from SMT becomes difficult because the number of constraints grows much more quickly than the number of available parameters. Function

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

Table 4
(a) Number of MaxSATtoIsing 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, for the "unbiased" MaxSAT instances. 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 2.90 GHz × 4 processor. gw2sat [95], rots [96], and novelty [97] are as implemented in UBCSAT [9]. 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	97.5
36 vars	100	95.7
40 vars	100	92.9
44 vars	100	91.1
48 vars	100	88
52 vars	100	86.1
56 vars	100	83.5
60 vars	100	83.1
64 vars	100	80.8
68 vars	100	81
72 vars	100	79.5
76 vars	100	79
80 vars	100	75.1

(a)

MaxSAT solvers: avg time (ms)				
Problem size	g2wsat	rots	maxwalksat	novelty
32 vars	0.018	0.013	0.025	0.012
36 vars	0.024	0.019	0.036	0.018
40 vars	0.037	0.030	0.052	0.024
44 vars	0.049	0.041	0.076	0.038
48 vars	0.070	0.064	0.115	0.056
52 vars	0.102	0.099	0.176	0.080
56 vars	0.153	0.161	0.262	0.117
60 vars	0.217	0.252	0.403	0.171
64 vars	0.303	0.383	0.598	0.241
68 vars	0.434	0.604	0.938	0.362
72 vars	0.620	0.964	1.448	0.551
76 vars	0.914	1.536	2.262	0.829
80 vars	1.364	2.567	3.618	1.312

(b)

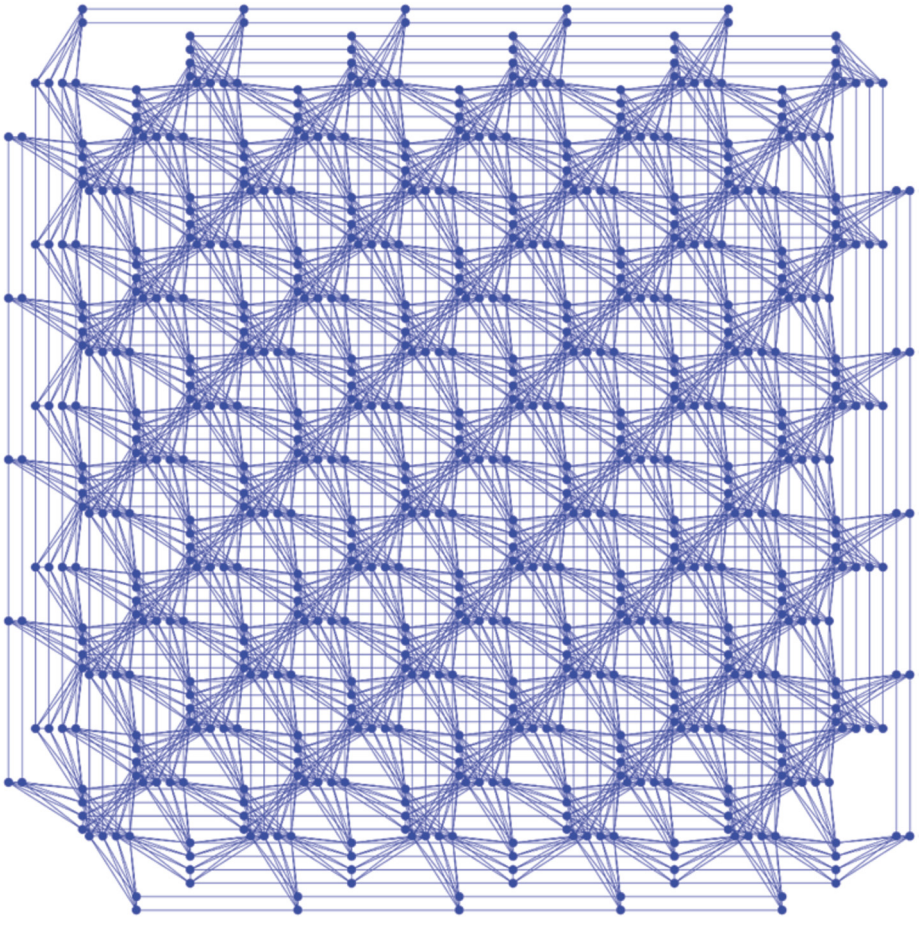


Fig. 12. "Pegasus", the hardware graph of an experimental QA system under development at D-Wave (720-qubit version). Qubits have maximum degree 15 rather than 6, and qubits do not fall into well-defined unit tiles as in Chimera.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

decomposition and chains provide one way around this, but chains limit the resulting 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 function with additional qubits for the purposes of increasing its gap. SMT formulations of these problems have not yet been explored.

- *Solving* (15) *directly.* In the field of automated theorem proving and SMT, novel techniques for solving *quantified* SMT formulas are emerging. It is thus possible to investigate these techniques for solving directly the quantified formulas (15), avoiding thus the expensive Shannon expansion of (16)-(19).
- *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.
- *More connected topologies*. Future QA hardware graphs will be larger, have higher per-qubit connectivity, and have less separation between clusters (tiles) of qubits. An example of a next-generation hardware graph under development at D-Wave is shown in Fig. 12. While these changes will result in the ability to solve larger and more difficult Ising problems, they will also require new encoding strategies. In particular, new methods for problem decomposition, placing small Boolean functions, and penalty modeling that take advantage of additional connectivity will significantly improve the encoding process.

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.[24]

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We wish to thank Patrick Trentin for assistance with OptiMathSAT. Roberto Sebastiani is sponsored by D-Wave Systems Inc. via project *QuASi (Quantum Annealing for SAT Solving)*.

Appendix. Summary of notation

Table 5
Summary of main symbols used in this paper. (Symbols may occur with superscripts and subscripts.) Left: symbol syntax and range. Center: symbol meaning. Right: place where the symbol appeared first.

Symbol	Meaning	Appear
$G \stackrel{\text{def}}{=} \langle V, E\rangle$	HW Graph	§1, (2)
$\{-1, 1\}$, $\{\bot, \top\}$, {false,true}	Boolean values	§1
$\underline{\mathbf{z}} \stackrel{\text{def}}{=} \{z_1, ..., z_{\|V\|}\} \in \{-1, 1\}^{\|V\|}$	generic Boolean variables/qubits	§1, (2)
$H(\underline{\mathbf{z}})$	Ising Hamiltonian	§1, (2)
$\theta_i \in [-2, 2]$ s.t. $1 \le i \le \|V\|$	bias of the i-th qubit	§1, (2)
$\theta_{ij} \in [-1, 1]$ s.t. $1 \le i < j \le \|V\|$	coupling of the (i, j)-th qubits	§1, (2)
$\underline{\mathbf{x}} \stackrel{\text{def}}{=} \{x_1, ..., x_n\} \in \{-1, 1\}^n$	input Boolean variables/qubits	§2.2
$\underline{\mathbf{y}} \stackrel{\text{def}}{=} \{y_1, ..., y_m\} \in \{-1, 1\}^m$	auxiliary Boolean variables	§2.2, (4)
$\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$	Boolean connectives	§2.2
$F(\underline{\mathbf{x}})$	Boolean Function/Formula	§2.2
$\{\langle F_k, c_k\rangle\}_k$, $c_k \in \mathbb{Q}^+$	Weighted MaxSAT problem	§2.2
$\underline{\mathbf{a}} \stackrel{\text{def}}{=} \{a_1, ..., a_h\} \in \{-1, 1\}^h$	ancillary Boolean variables/qubits	§3
$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}\|\boldsymbol{\theta})$, $P_F(\underline{\mathbf{x}}\|\boldsymbol{\theta})$	Penalty Function s.t. $\underline{\mathbf{z}} \stackrel{\text{def}}{=} \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$	§3.1, (5)
$\theta_0 \in (-\infty, +\infty)$	offset	§3.1, (5)
$g_{min} \in (0, +\infty)$	gap	§3.1, (6)
$w_k \in (0, +\infty)$	weights	§3.2, (8)
$\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}}) \in \{-1, 1\}^h$	binary witness variables	§4.2
$g(\underline{\mathbf{a}}_{\mathcal{V}_h})$: $\{\pm1\}^{\|\mathcal{V}_i\|} \rightarrow \mathbb{R}$	factor	§4.2 (23)
$f(\underline{\mathbf{a}}_{\mathcal{V}_h}, a_h)$: $\{\pm1\}^{\|\mathcal{V}_i\|+1} \rightarrow \mathbb{R}$	factor	§4.2 (23)

(*continued on next page*)

[24] See Footnote 22.

Table 5 (*continued*)

Symbol	Meaning	Appear
$\mathcal{F}_i$	bucket (set of factors)	§4.2
$\mathcal{V}_i$	ancillary variables of $\mathcal{F}_i$	§4.2
$m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}\|\underline{\mathbf{x}}) \in \mathbb{R}$	message variable	§4.2
$\underline{\mathbf{v}} \stackrel{\text{def}}{=} (v_1, ..., v_{n+h}), \underline{\mathbf{v}} \in [1, n+h]$	v_i: the vertex where z_i is placed	§4.3
b: $[1, n+h] \longmapsto [-2, 2]$	b(i): bias of the i-th qubit vertex	§4.4
c: $[1, n+h]^2 \longmapsto [-1, 1]$	c(i, j): coupling of the i, j-th edge	§4.4
$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}\|\theta_0, \mathrm{b}, \mathrm{c}, \underline{\mathbf{v}})$	Variable-placing Penalty Function	§4.4
$D \stackrel{\text{def}}{=} \langle V_D, A_D \rangle$	directed acyclic graph	§5.2
M: $V_D \to 2^{V_D}$	mapping of k-feasible cuts	§5.3
$\chi(C)$	characteristic vector of C	§5.4
$T_i \subseteq V$	Steiner tree terminal	§5.4
$C_i, S_i \subseteq V$	Steiner tree of T_i	§5.4
$x_{i,S_i} \in \{0, 1\}$	selection variable for S_i	§5.4

References

[1] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509, https://doi.org/10.1137/S0097539795293172.
[2] L.K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96, ACM, New York, NY, USA, 1996, pp. 212–219.
[3] A. Finnila, M. Gomez, C. Sebenik, C. Stenson, J. Doll, Quantum annealing: a new method for minimizing multidimensional functions, Chem. Phys. Lett. 219 (5) (1994) 343–348, https://doi.org/10.1016/0009-2614(94)00117-0.
[4] T. Kadowaki, H. Nishimori, Quantum annealing in the transverse Ising model, Phys. Rev. E 58 (1998) 5355–5363, https://doi.org/10.1103/PhysRevE.58.5355.
[5] E. Farhi, J. Goldstone, S. Gutmann, M. Sipser, Quantum computation by adiabatic evolution, arXiv preprint, arXiv:quant-ph/0001106.
[6] P.I. Bunyk, E.M. Hoskinson, M.W. Johnson, E. Tolkacheva, F. Altomare, A.J. Berkley, R. Harris, J.P. Hilton, T. Lanting, A.J. Przybysz, J. Whittaker, Architectural considerations in the design of a superconducting quantum annealing processor, IEEE Trans. Appl. Supercond. 24 (4) (2014) 1–10, https://doi.org/10.1109/TASC.2014.2318294.
[7] B. Selman, H. Kautz, B. Cohen, Local search strategies for satisfiability testing, in: Cliques, Coloring, and Satisfiability, in: DIMACS, vol. 26, 1996, pp. 521–532.
[8] W.M. Spears, Simulated annealing for hard satisfiability problems, in: Cliques, Coloring, and Satisfiability, in: DIMACS, vol. 26, American Mathematical Society, 1996, pp. 533–558.
[9] D.A.D. Tompkins, H.H. Hoos, UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT, in: H. Hoos, D. Mitchell (Eds.), Revised Selected Papers from the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), in: Lecture Notes in Computer Science, vol. 3542, Springer, Berlin, Heidelberg, 2005, pp. 306–320.
[10] V.S. Denchev, S. Boixo, S.V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, H. Neven, What is the computational value of finite-range tunneling?, Phys. Rev. X 6 (2016) 031015, https://doi.org/10.1103/PhysRevX.6.031015.
[11] J. King, S. Yarkoni, J. Raymond, I. Ozfidan, A.D. King, M.M. Nevisi, J.P. Hilton, C.C. McGeoch, Quantum annealing amid local ruggedness and global frustration, arXiv preprint, arXiv:1701.04579.
[12] A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009.
[13] C.M. Li, F. Manyà, MaxSAT, hard and soft constraints, Ch. 19, in: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009, pp. 613–631.
[14] F. Massacci, L. Marraro, Logical cryptanalysis as a sat problem, J. Autom. Reason. 24 (1) (2000) 165–203, https://doi.org/10.1023/A:1006326723002.
[15] I. Mironov, L. Zhang, Applications of SAT solvers to cryptanalysis of hash functions, in: A. Biere, C.P. Gomes (Eds.), Theory and Applications of Satisfiability Testing - SAT 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 102–115.
[16] F. Lafitte, J.N. Jr., D.V. Heule, Applications of SAT solvers in cryptanalysis: finding weak keys and preimages, J. Satisf. Boolean Model. Comput. - JSAT 9 (1) (2013) 1–25, https://doi.org/10.3233/SAT190099.
[17] A. Fréchette, N. Newman, K. Leyton-Brown, Solving the station repacking problem, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, pp. 702–709, http://dl.acm.org/citation.cfm?id=3015812.3015917.
[18] C.W. Barrett, R. Sebastiani, S.A. Seshia, C. Tinelli, Satisfiability modulo theories, in: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009, p. 980.
[19] R. Sebastiani, S. Tomasi, Optimization modulo theories with linear rational costs, ACM Trans. Comput. Log. 16 (2) (2015) 12:1–12:43, https://doi.org/10.1145/2699915.
[20] R. Sebastiani, P. Trentin, OptiMathSAT: a tool for optimization modulo theories, J. Autom. Reason. 64 (2020) 423–460, https://doi.org/10.1007/s10817-018-09508-6.
[21] Z. Bian, F. Chudak, W. Macready, A. Roy, R. Sebastiani, S. Varotti, Solving SAT and MaxSAT with a quantum annealer: foundations and a preliminary report, in: C. Dixon, M. Finger (Eds.), Frontiers of Combining Systems, Springer International Publishing, Cham, 2017, pp. 153–171.
[22] R. Harris, J. Johansson, A.J. Berkley, M.W. Johnson, T. Lanting, S. Han, P. Bunyk, E. Ladizinsky, T. Oh, I. Perminov, et al., Experimental demonstration of a robust and scalable flux qubit, Phys. Rev. B 81 (13) (2010), https://doi.org/10.1103/physrevb.81.134510.
[23] M.W. Johnson, M.H.S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A.J. Berkley, J. Johansson, P. Bunyk, E.M. Chapple, C. Enderud, J.P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M.C. Thom, E. Tolkacheva, C.J.S. Truncik, S. Uchaikin, J. Wang, B. Wilson, G. Rose, Quantum annealing with manufactured spins, Nature 473 (7346) (2011) 194–198, https://doi.org/10.1038/nature10012.
[24] T. Lanting, R. Harris, J. Johansson, M.H.S. Amin, A.J. Berkley, S. Gildert, M.W. Johnson, P. Bunyk, E. Tolkacheva, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, E.M. Chapple, C. Enderud, C. Rich, B. Wilson, M.C. Thom, S. Uchaikin, G. Rose, Cotunneling in pairs of coupled flux qubits, Phys. Rev. B 82 (6) (2010) 060512.
[25] M.H. Amin, Searching for quantum speedup in quasistatic quantum annealers, Phys. Rev. A 92 (5) (2015) 052323, https://doi.org/10.1103/PhysRevA.92.052323, arXiv:1503.04216.
[26] M.H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, R. Melko, Quantum Boltzmann machine, Phys. Rev. X 8 (2018) 021050, https://doi.org/10.1103/PhysRevX.8.021050.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

[27] J. Raymond, S. Yarkoni, E. Andriyash, Global warming: temperature estimation in annealers, Front. ICT 3 (2016) 23, https://doi.org/10.3389/fict.2016.00023.
[28] J.P. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, Ch. 4, in: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009, pp. 131–153.
[29] G.S. Tseitin, On the Complexity of Derivation in Propositional Calculus, Springer Berlin Heidelberg, Berlin, Heidelberg, 1983, pp. 466–483.
[30] S.A. Cook, The complexity of theorem proving procedures, in: 3rd Annual ACM Symposium on the Theory of Computation, 1971, pp. 151–158.
[31] S.M. Majercik, Stochastic Boolean satisfiability, Ch. 27, in: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009, pp. 887–925.
[32] A. Cimatti, A. Griggio, B.J. Schaafsma, R. Sebastiani, The MathSAT 5 SMT solver, in: Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13, in: LNCS, vol. 7795, Springer, 2013, pp. 95–109.
[33] R. Sebastiani, P. Trentin, Optimathsat: a tool for optimization modulo theories, in: D. Kroening, C.S. Păsăreanu (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2015, pp. 447–454.
[34] Z. Bian, F. Chudak, R. Israel, B. Lackey, W.G. Macready, A. Roy, Discrete optimization using quantum annealing on sparse Ising models, Front. Phys. 2 (2014) 56, https://doi.org/10.3389/fphy.2014.00056.
[35] V.P. Correia, A.I. Reis, C. Porto, A.R. Brasil, Classifying n-input Boolean functions, in: Proc. IWS, Citeseer, 2001.
[36] Z. Huang, L. Wang, Y. Nasikovskiy, A. Mishchenko, Fast Boolean matching based on NPN classification, in: 2013 International Conference on Field-Programmable Technology (FPT), 2013, pp. 310–313.
[37] V. Choi, Minor-embedding in adiabatic quantum computation: I. The parameter setting problem, Quantum Inf. Process. 7 (5) (2008) 193–209, https://doi.org/10.1007/s11128-008-0082-9.
[38] V. Choi, Minor-embedding in adiabatic quantum computation: II. Minor-universal graph design, Quantum Inf. Process. 10 (3) (2011) 343–353, https://doi.org/10.1007/s11128-010-0200-3.
[39] I. Adler, F. Dorn, F.V. Fomin, I. Sau, D.M. Thilikos, Faster parameterized algorithms for minor containment, in: Proceedings of the 12th Scandinavian Conference on Algorithm Theory, SWAT'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 322–333.
[40] T. Boothby, A.D. King, A. Roy, Fast clique minor generation in Chimera qubit connectivity graphs, Quantum Inf. Process. 15 (1) (2016) 495–508, https://doi.org/10.1007/s11128-015-1150-6.
[41] A. Zaribafiyan, D.J.J. Marchand, S.S. Changiz Rezaei, Systematic and deterministic graph minor embedding for Cartesian products of graphs, Quantum Inf. Process. 16 (5) (2017) 136, https://doi.org/10.1007/s11128-017-1569-z.
[42] J. Cai, W.G. Macready, A. Roy, A practical heuristic for finding graph minors, arXiv preprint, arXiv:1406.2741.
[43] R. Dechter, Bucket Elimination: A Unifying Framework for Probabilistic Inference, Springer Netherlands, Dordrecht, 1998, pp. 75–104.
[44] B.D. McKay, A. Piperno, Practical graph isomorphism, II, J. Symb. Comput. 60 (2014) 94–112, https://doi.org/10.1016/j.jsc.2013.09.003.
[45] Z. Bian, F. Chudak, R.B. Israel, B. Lackey, W.G. Macready, A. Roy, Mapping constrained optimization problems to quantum annealing with application to fault diagnosis, Front. ICT 2016, https://doi.org/10.3389/fict.2016.00014.
[46] J. Su, T. Tu, L. He, A quantum annealing approach for Boolean satisfiability problem, in: 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC), 2016, pp. 1–6.
[47] A. Mishchenko, S. Chatterjee, R. Brayton, Dag-aware AIG rewriting a fresh look at combinational logic synthesis, in: Proceedings of the 43rd Annual Design Automation Conference, DAC '06, ACM, New York, NY, USA, 2006, pp. 532–535.
[48] A. Mishchenko, S. Chatterjee, R. Jiang, R.K. Brayton, FRAIGs: a unifying representation for logic synthesis and verification, Tech. rep., ERL Technical Report, 2005.
[49] N. Een, A. Mishchenko, N. Sörensson, Applying logic synthesis for speeding up sat, in: J. Marques-Silva, K.A. Sakallah (Eds.), Theory and Applications of Satisfiability Testing – SAT 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 272–286.
[50] A. Mishchenko, S. Chatterjee, R. Brayton, X. Wang, T. Kam, Technology mapping with Boolean matching, supergates and choices, https://people.eecs.berkeley.edu/~alanmi/publications/2005/tech05_map.pdf, 2005.
[51] V. Betz, J. Rose, VPR: a new packing, placement and routing tool for FPGA research, in: International Workshop on Field Programmable Logic and Applications, Springer, 1997, pp. 213–222.
[52] A.B. Kahng, J. Lienig, I.L. Markov, J. Hu, VLSI Physical Design: From Graph Partitioning to Timing Closure, Springer Netherlands, Dordrecht, 2011.
[53] W.-J. Sun, C. Sechen, Efficient and effective placement for very large circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 14 (3) (1995) 349–359, https://doi.org/10.1109/43.365125.
[54] T.F. Chan, J. Cong, T. Kong, J.R. Shinnerl, Multilevel optimization for large-scale circuit placement, in: IEEE/ACM International Conference on Computer Aided Design, ICCAD - 2000, 2000, pp. 171–176, IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140).
[55] J.A. Roy, D.A. Papa, S.N. Adya, H.H. Chan, A.N. Ng, J.F. Lu, I.L. Markov, Capo: robust and scalable open-source min-cut floorplacer, in: Proceedings of the 2005 International Symposium on Physical Design, ISPD '05, ACM, New York, NY, USA, 2005, pp. 224–226.
[56] J. Byrka, F. Grandoni, T. Rothvoss, L. Sanità, Steiner tree approximation via iterative randomized rounding, J. ACM 60 (1) (2013) 6:1–6:33, https://doi.org/10.1145/2432622.2432628.
[57] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, J. Vygen, BonnRoute: algorithms and data structures for fast and good VLSI routing, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013) 32:1–32:24, https://doi.org/10.1145/2442087.2442103.
[58] Y. Xu, Y. Zhang, C. Chu, Fastroute 4.0: global router with efficient via minimization, in: Proceedings of the 2009 Asia and South Pacific Design Automation Conference, ASP-DAC '09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 576–581, http://dl.acm.org/citation.cfm?id=1509633.1509768.
[59] J.A. Roy, I.L. Markov, High-performance routing at the nanometer scale, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 27 (6) (2008) 1066–1077, https://doi.org/10.1109/TCAD.2008.923255.
[60] H. Chen, C. Hsu, Y. Chang, High-performance global routing with fast overflow reduction, in: 2009 Asia and South Pacific Design Automation Conference, 2009, pp. 582–587.
[61] M. Cho, K. Lu, K. Yuan, D.Z. Pan, Boxrouter 2.0: architecture and implementation of a hybrid and robust global router, in: 2007 IEEE/ACM International Conference on Computer-Aided Design, 2007, pp. 503–508.
[62] Y.J. Chang, Y.T. Lee, T.C. Wang, NTHU-route 2.0: a fast and stable global router, in: 2008 IEEE/ACM International Conference on Computer-Aided Design, 2008, pp. 338–343.
[63] J. Byrka, F. Grandoni, T. Rothvoß, L. Sanità, An improved LP-based approximation for Steiner tree, in: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5–8 June 2010, pp. 583–592.
[64] P. Klein, R. Ravi, A nearly best-possible approximation algorithm for node-weighted Steiner trees, J. Algorithms 19 (1) (1995) 104–115, https://doi.org/10.1006/jagm.1995.1029.
[65] V.V. Vazirani, Approximation Algorithms, Springer-Verlag, Berlin, Germany, 2001.
[66] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, J. Vygen, BonnRoute: algorithms and data structures for fast and good VLSI routing, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013) 32, https://doi.org/10.1145/2442087.2442103.
[67] D. Müller, K. Radke, J. Vygen, Faster min–max resource sharing in theory and practice, Math. Program. Comput. 3 (1) (2011) 1–35, https://doi.org/10.1007/s12532-011-0023-y.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609

[68] A. Lucas, Ising formulations of many NP problems, Front. Phys. 2 (2014) 5, https://doi.org/10.3389/fphy.2014.00005.
[69] N. Chancellor, S. Zohren, P.A. Warburton, S.C. Benjamin, S. Roberts, A direct mapping of max k-sat and high order parity checks to a Chimera graph, Sci. Rep. 6 (2016) 37107, http://dro.dur.ac.uk/20497/.
[70] S. Pakin, A quantum macro assembler, in: 2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13–15, 2016, 2016, pp. 1–8.
[71] S. Pakin, Performing fully parallel constraint logic programming on a quantum annealer, Theory Pract. Log. Program. 18 (5–6) (2018) 928–949, https://doi.org/10.1017/S1471068418000066.
[72] D. Venturelli, D.J.J. Marchand, G. Rojo, Quantum annealing implementation of job-shop scheduling, arXiv:1506.08479.
[73] G. Rosenberg, P. Haghnegahdar, P. Goddard, P. Carr, K. Wu, M.L. de Prado, Solving the optimal trading trajectory problem using a quantum annealer, in: Proceedings of the 8th Workshop on High Performance Computational Finance, WHPCF '15, ACM, New York, NY, USA, 2015, pp. 7:1–7:7.
[74] R. Dridi, H. Alghassi, Prime factorization using quantum annealing and computational algebraic geometry, Sci. Rep. 7 (1) (2017), https://doi.org/10.1038/srep43048.
[75] A. Perdomo-Ortiz, J. Fluegemann, S. Narasimhan, R. Biswas, V. Smelyanskiy, A quantum annealing approach for fault detection and diagnosis of graph-based systems, Eur. Phys. J. Spec. Top. 224 (1) (2015) 131–148, https://doi.org/10.1140/epjst/e2015-02347-y.
[76] E.G. Rieffel, D. Venturelli, B. O'Gorman, M.B. Do, E.M. Prystay, V.N. Smelyanskiy, A case study in programming a quantum annealer for hard operational planning problems, Quantum Inf. Process. 14 (1) (2015) 1–36, https://doi.org/10.1007/s11128-014-0892-x.
[77] B. O'Gorman, E.G. Rieffel, M. Do, D. Venturelli, J. Frank, Comparing planning problem compilation approaches for quantum annealing, Knowl. Eng. Rev. 31 (5) (2016) 465–474, https://doi.org/10.1017/S0269888916000278.
[78] K.M. Zick, O. Shehab, M. French, Experimental quantum annealing: case study involving the graph isomorphism problem, Sci. Rep. 5 (2015) 11168, http://www.nature.com/srep/2015/150608/srep11168/full/srep11168.html.
[79] Z. Bian, F. Chudak, W.G. Macready, L. Clark, F. Gaitan, Experimental determination of Ramsey numbers, Phys. Rev. Lett. 111 (2013) 130505, https://doi.org/10.1103/PhysRevLett.111.130505.
[80] S. Jiang, K.A. Britt, A.J. McCaskey, T.S. Humble, S. Kais, Quantum annealing for prime factorization, arXiv:1804.02733.
[81] I. Trummer, C. Koch, Multiple query optimization on the d-wave 2x adiabatic quantum computer, Proc. VLDB Endow. 9 (9) (2016) 648–659, https://doi.org/10.14778/2947618.2947621.
[82] E. Andriyash, Z. Bian, F. Chudak, M. Drew-Brook, A.D. King, W.G. Macready, A. Roy, Boosting integer factoring performance via quantum annealing offsets, https://www.dwavesys.com/sites/default/files/14-1002A_B_tr_Boosting_integer_factorization_via_quantum_annealing_offsets.pdf.
[83] C.C. McGeoch, C. Wang, Experimental evaluation of an adiabatic quantum system for combinatorial optimization, in: Proceedings of the ACM International Conference on Computing Frontiers, CF '13, ACM, New York, NY, USA, 2013, pp. 23:1–23:11.
[84] S. Santra, G. Quiroz, G.V. Steeg, D.A. Lidar, Max 2-sat with up to 108 qubits, New J. Phys. 16 (4) (2014) 045006, http://stacks.iop.org/1367-2630/16/i=4/a=045006.
[85] A. Douglass, A.D. King, J. Raymond, Constructing SAT Filters with a Quantum Annealer, Springer International Publishing, Cham, 2015, pp. 104–120.
[86] K.L. Pudenz, G.S. Tallant, T.R. Belote, S.H. Adachi, Quantum annealing and the satisfiability problem, arXiv:1612.07258.
[87] E. Farhi, D. Gosset, I. Hen, A.W. Sandvik, P. Shor, A.P. Young, F. Zamponi, Performance of the quantum adiabatic algorithm on random instances of two optimization problems on regular hypergraphs, Phys. Rev. A 86 (2012) 052334, https://doi.org/10.1103/PhysRevA.86.052334.
[88] I. Hen, A.P. Young, Exponential complexity of the quantum adiabatic algorithm for certain satisfiability problems, Phys. Rev. E 84 (2011) 061152, https://doi.org/10.1103/PhysRevE.84.061152.
[89] V. Choi, Different adiabatic quantum optimization algorithms for the NP-complete exact cover and 3SAT problems, Quantum Inf. Comput. 11 (7–8) (2011) 638–648, http://dl.acm.org/citation.cfm?id=2230916.2230923.
[90] A.D. King, T. Lanting, R. Harris, Performance of a quantum annealer on range-limited constraint satisfaction problems, arXiv:1502.02098.
[91] R. Brayton, A. Mishchenko, ABC: an academic industrial-strength verification tool, in: International Conference on Computer Aided Verification, Springer, 2010, pp. 24–40.
[92] I. Spence, Sgen1: a generator of small but difficult satisfiability benchmarks, ACM J. Exp. Algorithmics 15 (2010) 1.2, https://doi.org/10.1145/1671970.1671972.
[93] A.V. Gelder, I. Spence, Zero-one designs produce small hard SAT instances, in: O. Strichman, S. Szeider (Eds.), Theory and Applications of Satisfiability Testing – SAT 2010, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 388–397, https://doi.org/10.1007/978-3-642-14186-7_37.
[94] I. Spence, Weakening cardinality constraints creates harder satisfiability benchmarks, ACM J. Exp. Algorithmics 20 (2015) 1.4, https://doi.org/10.1145/2746239.
[95] C.M. Li, W.Q. Huang, Diversification and determinism in local search for satisfiability, in: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005, St Andrews, UK, June 19-23, 2005, Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 158–172.
[96] K. Smyth, H.H. Hoos, T. Stützle, Iterated robust tabu search for max-sat, in: Proceedings of the 16th Canadian Society for Computational Studies of Intelligence Conference on Advances in Artificial Intelligence, AI'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 129–144, http://dl.acm.org/citation.cfm?id=1760335.1760351.
[97] D. McAllester, B. Selman, H. Kautz, Evidence for invariants in local search, in: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence, AAAI'97/IAAI'97, AAAI Press, 1997, pp. 321–326, http://dl.acm.org/citation.cfm?id=1867406.1867456.

Please cite this article in press as: Z. Bian et al., Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results, Inf. Comput. (2020), https://doi.org/10.1016/j.ic.2020.104609