D-Wave
The Quantum Computing Company™

Programming with QUBOs

Release 2.4

09-1002A-C

CONTENTS

1 Discrete Optimization Background **1**
1.1 The Ising Model . . . 1
1.2 Quadratic Unconstrained Binary Optimization . . . 1

2 Quantum Annealing Background **3**
2.1 Simulated Annealing . . . 3
2.2 Quantum Annealing . . . 4

3 Ising Problems Addressed Natively by the QPU **7**
3.1 Limitations on Hardware-Realized Ising Models . . . 7
3.2 Circumventing Hardware Limitations . . . 10
3.3 Effects on Problem Difficulty . . . 19

4 Idioms for Discrete Optimization with QUBO Objectives **21**
4.1 Other Modeling Representations . . . 21

5 Bibliography **33**

Bibliography **35**

CHAPTER

ONE

DISCRETE OPTIMIZATION BACKGROUND

Discrete optimization is concerned with the optimization of objective functions defined over search spaces consisting of finite set of objects. It is also commonly called combinatorial optimization. We focus on two classes of optimization objectives: Ising problems defined over strings of $-1/+1$ symbols, or QUBOs defined over strings of $0/1$ symbols. As will be revealed it is important to recognize that this limitation is not in principle constraining, any discrete optimization problem may be brought into this form.

1.1 The Ising Model

The D-Wave™ system can be viewed as a hardware heuristic which minimizes Ising objective functions using a physically realized version of quantum annealing. The Ising energy minimization problem of n variables $s_i \in \{-1, +1\}$ is given by [1]

$$\text{ISING:} \qquad \boldsymbol{s}^\star = \operatorname*{argmin}_{\boldsymbol{s}} E(\boldsymbol{s}|\boldsymbol{h}, \boldsymbol{J}) = \operatorname*{argmin}_{\boldsymbol{s}} \left\{ \sum_{(i,j)\in E} s_i J_{i,j} s_j + \sum_{i\in V} h_i s_i \right\}$$
$$= \operatorname*{argmin}_{\boldsymbol{s}} \Big\{ \langle \boldsymbol{s}, \boldsymbol{J}\boldsymbol{s} \rangle + \langle \boldsymbol{h}, \boldsymbol{s} \rangle \Big\}.$$

Structural information about the Ising problem is captured in its *primal graph* $G = (V, E)$. The set of vertices V of the primal graph correspond to the optimization variables, and edges $(i, j) \in E$ specify which $\boldsymbol{J}_{i,j}$ may be non-zero. Without loss of generality it is convenient to assume that $\boldsymbol{J}$ is upper-triangular. If $J_{i,j} > 0$ the edge is positive, and if $J_{i,j} < 0$ the edge is negative. The primal graph captures independencies between variables which can be exploited to more efficiently minimize the Ising objective.

1.2 Quadratic Unconstrained Binary Optimization

Often it is more convenient to model with 0/1-valued variables than the -1/1-valued Ising variables. Consider for example, maximum independent set (MIS) where the goal is to select the largest subset of vertices (the independent set) of a graph $G = (V, E)$ such that no pair of selected vertices is connected by an edge. Let $x_i = 1$ if vertex i is in the independent set, and $x_i = 0$ otherwise. The constraint that no pair of selected vertices may be connected by an edge is easily represented by penalties of the form Mx_ix_j for all $(i, j) \in E$ where $M > 1$ is the penalty weight. In the -1/1 representation where $s_i = 1$ if i is in the independent set and $s_i = -1$ otherwise, the same penalty is expressed less intuitively as $M(s_is_j + s_i + s_j + 1)/4$. The largest independent set is determined by minimizing $-\sum_{i\in V} x_i$ so

[1] Vectors are indicated in lowercase bold font, *e.g.* $\boldsymbol{s} = [s_1, \cdots, s_n]$. Matrices are indicated in uppercase bold font. The Euclidean inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$.

that the total objective for MIS is

$$\min_{\boldsymbol{x}} \left\{ -\sum_{i \in V} x_i + M \sum_{(i,j) \in E} x_i x_j \right\}.$$

The 0/1 variables are more natural and succinct for this problem.

Quadratic Unconstrained Binary Optimization problems (QUBOs) are of the form

$$\text{QUBO:} \qquad \boldsymbol{x}^\star = \operatorname*{argmin}_{\boldsymbol{x}} E(\boldsymbol{x}|\boldsymbol{Q}) = \operatorname*{argmin}_{\boldsymbol{x}} \sum_{i \geq j} x_i Q_{i,j} x_j = \operatorname*{argmin}_{\boldsymbol{x}} \langle \boldsymbol{x}, \boldsymbol{Q}\boldsymbol{x} \rangle$$

where $x_i \in \{0, 1\}$. The terms linear in x_i arise from the diagonal contribution of $\boldsymbol{Q}$ since $Q_{i,i} x_i^2 = Q_{i,i} x_i$ because $x_i^2 = x_i$ for $x_i = 0$ or 1. Again, we may assume that $\boldsymbol{Q}$ is upper-triangular. In *[Kochenberger2004]* and related papers QUBOs are shown to be an effective representation for modeling and solving a variety of discrete optimization problems.

Ising and QUBO models are simply related to each other through the transformation $\boldsymbol{s} = 2\boldsymbol{x} - \mathbf{1}$ so that

$$\langle \boldsymbol{x}, \boldsymbol{Q}\boldsymbol{x} \rangle = \frac{1}{4} \langle \mathbf{1}, \boldsymbol{Q}\mathbf{1} \rangle + \langle \overbrace{\boldsymbol{Q}\mathbf{1}/2}^{\boldsymbol{h}}, \boldsymbol{s} \rangle + \langle \boldsymbol{s}, \overbrace{(\boldsymbol{Q}/4)}^{\boldsymbol{J}} \boldsymbol{s} \rangle$$

where $\mathbf{1}$ is the vector all of whose components are 1. Thus, $E(\boldsymbol{x}|\boldsymbol{Q}) = \langle \mathbf{1}, \boldsymbol{Q}\mathbf{1} \rangle + E(\boldsymbol{s}|\boldsymbol{Q}\mathbf{1}/2, \boldsymbol{Q}/4)$, and we may freely translate between whichever Ising/QUBO representation is more convenient. In the utility packs we provide code which applies this translation.

The Ising and QUBO problems are NP hard *[Barahona1982]* as is easily seen by expressing maximum independent set (an NP hard problem) as an Ising optimization problem. While the Ising/QUBO problem is hard in the worst case, there are a number of tractable subclasses of Ising/QUBO problems. Perhaps the broadest tractable subclass was defined in *[Bertsimas1999]* where it was shown that for sign-balanced primal graphs the linear programming relaxation is tight (the LP solution is binary valued). An Ising/QUBO problem is sign-balanced if the primal graph does not contain any cycles with an odd number of positive edges. A special case of sign-balanced graphs occurs when all $J_{i,j} \leq 0$ in which case the objective function is submodular and efficient algorithms based on graph cuts may be applied *[Kolmogorov2004]*. [2] Other tractable subclasses include problems where the primal graph satisfies certain structural properties, e.g. planarity *[Schraudolph2009]*, or low tree-width *[Coughlan2009]*.

[2] Though graph cut methods cannot solve problems that are close to submodular, *i.e.* a few positive edges, algorithms may be developed where the optimal values of certain variables in near-submodular problems may be inferred. In the utility packs we provide such functionality.

CHAPTER

TWO

QUANTUM ANNEALING BACKGROUND

The D-Wave system uses a physical quantum annealing (QA) process to provide approximate minimizers for a certain class of Ising objectives. The subclass of Ising problems addressed by the D-Wave hardware remains NP hard as the hardware is scaled. The material of this chapter is not needed to formulate and solve problems on the D-Wave system, but it may be of interest for those seeking to understand the underlying foundations of quantum annealing. Most users of discrete optimization are familiar with the simulated annealing (SA) heuristic. Quantum annealing shares similarities with SA, and we review SA from a perspective that generalizes to QA.

2.1 Simulated Annealing

Simulated annealing, proposed in *[Kirkpatrick1983]* and inspired by a physical process, is an attempt to balance exploration with exploitation. SA searches for the global minimizer $\boldsymbol{x}^\star$ in an indirect and stochastic way. One view of this process simplifies the picture by removing the stochastic elements of the algorithm. Rather than seeking the global minimizer $\boldsymbol{x}^\star$ directly we seek a probability distribution p over the search space. $p(\boldsymbol{x})$ defines the probability associated with element $\boldsymbol{x}$ in the search space. The highest probability p should be associated with $\boldsymbol{x}^\star$. p is determined by minimizing a function balancing explorative and exploitative terms. Physicists call this function free energy $F(p)$, and it is given by

$$F(p) = \mathbb{E}_p(E) - TS(p).$$

$\mathbb{E}_p(E) \equiv \sum_{\boldsymbol{x}} p(\boldsymbol{x})E(\boldsymbol{x})$ is the expected energy under the distribution p and is minimized by the probability distribution [1]

$$p(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = \boldsymbol{x}^\star \\ 0 & \text{for all other } \boldsymbol{x} \end{cases}.$$

The term $S(p) = -\sum_{\boldsymbol{x}} p(\boldsymbol{x}) \ln p(\boldsymbol{x})$ is the entropy; the negative entropy is minimized when equal probability is assigned to all $\boldsymbol{x}$. The inclusion of this term favors exploration of the entire search space. T (for temperature) is a parameter controlling the relative importance of these two terms. Simulated annealing begins at large T which favors widespread exploration of the search space. As the algorithm progresses T is decreased to narrow the focus down to promising regions where $\mathbb{E}_p(E)$ is low. Eventually, T is driven to zero so that the global minimum can be obtained.

For any T the free energy is convex (bowl shaped) as a function of p so that gradient descent with respect to p is guaranteed to locate the globally minimal $p^\star$. In fact, the optimal $p^\star$ at temperature T can be written down immediately as the Boltzmann distribution $p_T(\boldsymbol{x}) = \exp\big(-E(\boldsymbol{x})/T\big)$. This observation is not directly helpful because exponentially many numbers are required to specify p_T (we need to specify the probability of every $\boldsymbol{x}$). As a result, approximations to p_T need to be used. Most commonly, a collection of samples drawn approximately from p_T are used to estimate p_T. It is at this point the stochastic nature of SA returns. Markov chain Monte Carlo (MCMC) is a method that allows for samples to be drawn from a probability distribution. It bears much in common with local search methods for optimization. A proposal distribution $v_T(\boldsymbol{x}|\boldsymbol{x}')$ is defined which for each $\boldsymbol{x}'$ defines a probability over all possible

[1] Assuming the globally minimal state is unique.

next states $\boldsymbol{x}$. Typically, $v_T(\boldsymbol{x}|\boldsymbol{x}')$ is non-zero only for $\boldsymbol{x}$ in the neighborhood of $\boldsymbol{x}'$. An initial distribution $\pi_0(\boldsymbol{x})$ is evolved by v_T into $\pi_1(\boldsymbol{x}) = \sum_{\boldsymbol{x}'} v_T(\boldsymbol{x}|\boldsymbol{x}')\pi_0(\boldsymbol{x}')$. MCMC equilibrates (reaches a fixed point) at a distribution $\pi_\infty(\boldsymbol{x})$ satisfying

$$\pi_\infty(\boldsymbol{x}')v_T(\boldsymbol{x}|\boldsymbol{x}') = \pi_\infty(\boldsymbol{x})v_T(\boldsymbol{x}'|\boldsymbol{x}) \ ,$$

where the flow of probability out of $\boldsymbol{x}'$ into $\boldsymbol{x}$ is the same as the reverse flow. For SA, $v_T(\boldsymbol{x}|\boldsymbol{x}')$ is chosen so that the equilibrium distribution is $\pi_\infty = p_T$. The rate of convergence to π_∞ is governed by the gap between the largest and second largest eigenvalues of $v_T(\boldsymbol{x}|\boldsymbol{x}')$. The smaller the gap the slower the convergence.

The hardness of a problem for SA manifests itself in the sampling problem. The local nature of the MCMC proposal distribution can make large scale changes in $\boldsymbol{x}$ (i.e. far-ranging exploration) difficult. This is why the solution p_{T^+} at a slightly higher temperature $T^+ > T$ is used to bootstrap the sampling at p_T. In hard problems there is typically a particular T where sampling becomes particularly difficult as the character of p_T can change dramatically over a small range of T. This abrupt change in the character of p_T is called a phase transition. The transition in the Ising model from a non-magnetic to a magnetic state at a particular temperature T (the Curie temperature for ferromagnets, and the Neel temperature for antiferromagnets) is an example of a phase transition.

2.2 Quantum Annealing

From a computational point of view there is nothing special about using entropy as the objective to force exploration. Any function that smoothes the probability over the search space can serve the same purpose. In quantum annealing quantum processes are used to encourage exploration.

Quantum annealing was first proposed in *[Kadowaki1998]*. A similar, but more general model of computation was proposed in *[Farhi2000]* and dubbed *adiabatic quantum computation* (AQC). AQC was shown to be equivalent to the older and more familiar gate model of quantum computation *[Mizel2007]*. Though the complexity of quantum annealing is unknown, it is expected that hardware realization of this algorithm will provide dramatic speedups for approximately minimizing Ising objectives. [2] This speedup is due to the physical use of quantum resources, the inherent parallelism of the hardware, and the fast timescales of the underlying physical energy-minimizing dynamics. Having motivated quantum mechanics as an alternative exploration-inducing search mechanism we turn to an overview of QA.

For concreteness, suppose that we are trying to solve problem P represented as a QUBO on n bits. Classical mechanics is generalized to quantum mechanics by generalizing bits to qubits, classical states of 2^n binary configurations to quantum states, and the energy function E to a Hermitian operator H.

The 2^n classical states $\boldsymbol{s}$ form an orthonormal basis of a 2^n-dimensional vector space $\mathcal{V}$ and are denoted by $|\boldsymbol{s}\rangle$ in standard notation. [3] When $n = 1$, the vector space $\mathcal{V}$ has dimension 2, and we can assume that the two classical states, 0 and 1, map to the canonical vectors $|0\rangle = [1 \ \ 0]^\top$ and $|1\rangle = [0 \ \ 1]^\top$. When $n = 2$, the 4 classical states map to the 4 canonical vectors in $\mathcal{V}$ written as $|00\rangle = |0\rangle \otimes |0\rangle$, $|01\rangle = |0\rangle \otimes |1\rangle$, $|10\rangle = |1\rangle \otimes |0\rangle$, and $|11\rangle = |1\rangle \otimes |1\rangle$, that is, the tensor product of the classical states of each of the 2 qubits. In general, the classical state $\boldsymbol{x} = b_1 \dots b_n$, where b_i is the i th bit of $\boldsymbol{x}$, can be written as $|x\rangle = |b_1\rangle \otimes |b_2\rangle \dots \otimes |b_n\rangle$. A quantum state is simply a normalized vector of $\mathcal{V}$. Thus, an arbitrary quantum state ϕ in the basis of classical states can be written as $|\phi\rangle = \sum_{\boldsymbol{x}} \alpha_{\boldsymbol{x}}|\boldsymbol{x}\rangle$, where the 2^n complex numbers $\alpha_{\boldsymbol{x}}$ give the amplitudes in each of the 2^n basis vectors $|\boldsymbol{x}\rangle$, and satisfy $\sum_{\boldsymbol{x}} |\alpha_{\boldsymbol{x}}|^2 = 1$. $\langle\phi|$ is the Hermitian conjugate. Quantum states cannot be measured directly, but when qubits are measured in state $|\phi\rangle$ each classical state $|\boldsymbol{x}\rangle$ will be seen with probability $|\alpha_{\boldsymbol{x}}|^2$. For example, the single qubit state $(|0\rangle + |1\rangle)/\sqrt{2}$ is equally likely to be measured as either 0 or 1.

The operator H maps quantum states into quantum states, and for a system of n qubits can be represented as a matrix of size $2^n \times 2^n$. The energy of quantum state $|\phi\rangle$ is $\langle\phi|H|\phi\rangle$. Minimizing the energy $\langle\phi|H|\phi\rangle$ is accomplished by

[2] Problems with n variables having no structure (a flat energy landscape with a single minimum) can be solved by quantum annealing in time $\mathcal{O}(2^{n/2})$ rather than $\mathcal{O}(2^n)$ as is possible with classical algorithms. Problems difficult for QA *[Reichardt2004]*, and problems for which quantum annealing is exponentially faster than simulated annealing *[Farhi2002]* can both be constructed.

[3] $|\phi\rangle$ is the standard physics notation for the column vector indicated by ϕ. $\langle\phi|$ is the row vector of complex-conjugated elements of $|\phi\rangle$. With this notation the inner product of vectors $|\phi_1\rangle$ and $|\phi_2\rangle$ is $\langle\phi_1|\phi_2\rangle$.

finding the smallest eigenvalue of H, and the corresponding normalized eigenvector(s). For our QUBO problem P, each classical state $\boldsymbol{x}$ has energy $E_P(\boldsymbol{x})$. The operator H_P representing this problem is diagonal in the $\{|\boldsymbol{x}\rangle\}$ basis and satisfies $H_P|\boldsymbol{x}\rangle = E_P(\boldsymbol{x})|\boldsymbol{x}\rangle$. Thus, the classical states that have smallest energy $E_P(\boldsymbol{x})$ are also minimizers of $\langle\phi|H_P|\phi\rangle$ over all quantum states $|\phi\rangle$. Furthermore, the minimum value of the energy $\langle\phi|H_P|\phi\rangle$ is equal to the smallest eigenvalue of H_P which in turn is equal to $\min_{\boldsymbol{x}} E_P(\boldsymbol{x})$. Even though the matrix representing the diagonal operator H_P is exponentially large, the operator H_P can be easily configured in hardware, because it decomposes by qubits (linear terms) and pairs of qubits (quadratic terms).

The goal of quantum annealing is to find a minimizer of H_P through a physical quantum evolution. This is accomplished in a similar way as SA. The process starts by considering a second energy-like operator H_X whose minimal energy state is the quantum state that has equal squared amplitude for all bit configurations so that all classical states are equally likely to be observed when measured. The operator H_X is easily configurable in hardware because it can be applied qubit by qubit (we make each qubit equally likely to be measured as 0 or 1). Furthermore, the quantum system can easily attain this minimal energy state. Unlike H_P, H_X is not diagonal.

We can, then, define a quantum version of a free-energy-like objective

$$\mathcal{F}(\rho) = \mathrm{tr}(H_P\rho) + \tau\,\mathrm{tr}(H_X\rho)$$

where $\rho = |\phi\rangle\langle\phi|$ and $\mathrm{tr}(H\rho) = \langle\phi|H|\phi\rangle$. ρ is a $2^n \times 2^n$ matrix, but acts like a probability in that all its eigenvalues are positive and sum to 1. With this definition the parallels to SA are direct. τ is initialized at a very large value, and minimizing $\mathcal{F}$ with respect to ρ is the same as minimizing H_X alone which makes all $|\boldsymbol{x}\rangle$ equally likely to be observed. Similarly, when $\tau = 0$ minimization yields the solution to the optimization problem encoded by H_P. However, just as we faced a problem in SA due to the exponential size of p, here ρ is exponentially large. The solution to this problem in the present context however is vastly more satisfying, and relies on Nature to implicitly represent ρ.

The minimization of $\mathcal{F}$ (which amounts to diagonalizing $H_P + \tau H_X$ to find the lowest eigenvector) is accomplished by Nature which natively operates on exponentially large quantum states ϕ. Thus, by setting initial conditions to define $\mathcal{F}(\rho)$ and relying on physics we can circumvent the problems associated with the exponential size of ρ. Given that Nature solves our problem why do we need to bother with the H_X contribution encouraging exploration? The answer to this is that it may take an exponentially long time for the physical system to relax to its lowest energy state. We recall that the Ising optimization problems facing Nature are difficult, and even physical evolution can get stuck in local minima for long periods of time. Fortunately, the adiabatic theorem *[Ambainis2004]* can be applied to the system to speed the relaxation to lowest energy states. In simple terms, if the system is initialized at a minimum energy state at large initial τ, and the decrease of τ over time is sufficiently slow, the system will remain at minimum energy state throughout the evolution. In other words, by annealing τ from large values to zero we speed the relaxation to the globally lowest energy state of H_P. The adiabatic theorem relates the rate at which τ can be decreased to the eigenvalues of $H_P + \tau H_X$. More conventionally τ is expressed as $\tau = (1-s)/s$ so that the free-energy-like objective can be restated as $\mathcal{F}(\rho) = \mathrm{tr}\big((sH_P + (1-s)H_X)\rho\big)$ where s now increases from 0 to 1 during quantum annealing. [4]

The parallelism inherent to quantum mechanical evolution can translate into significantly improved optimization. The first quantitative result was derived for unstructured search *[Grover1996]*. Consider a classical energy function $E(\boldsymbol{x}) = -1$ for $\boldsymbol{x} = \boldsymbol{x}^\star$, and $E(\boldsymbol{x}) = 0$ for all other configurations. This problem is extremely difficult as the energy function provides no hints as to what $\boldsymbol{x}^\star$ might be. For strings of length n the best classical search (sampling without replacement) requires $\mathcal{O}(2^n)$ time to locate $\boldsymbol{x}^\star$. Quantum mechanically this can be improved to $\mathcal{O}(2^{n/2})$ using a specially tuned interpolation $g(s)$ from H_X to H_P, i.e. $H = (1-g(s))H_X + g(s)H_P$ *[Roland2002]*. While unstructured search remains exponentially hard, the scaling exponent is halved. Carefully engineered optimization problems can be constructed for which quantum annealing is exponentially faster than its classical counterpart SA *[Farhi2002]* Other energy functions with structure can also be constructed in which QA offers no speed-ups beyond the unstructured search square root increase *[Reichardt2004]*. In general, the efficacy of QA is governed by the minimum gap $\min_s E_1(s) - E_0(s)$ where $E_0(s)$ is the lowest energy eigenvalue of $H(s) = (1-s)H_X + sH_P$, and $E_1(s)$ is the second lowest eigenvalue. However, determining the minimum gap is as difficult as the original problem, and theory offers little practical guidance.

Exploration combining both thermal and quantum mechanisms is possible. In this case the appropriate free energy

[4] The annealing parameter s should not be confused with $\boldsymbol{s}$ the vector of spin values.

function is

$$\mathcal{F}(\rho) = \mathrm{tr}(H\rho) - TS(\rho)$$

where $H = sH_P + (1-s)H_X$ and $S(\rho) = -\mathrm{tr}(\rho \ln \rho)$ is the quantum generalization of entropy. The limits $T \to 0$ and $s \to 1$ remove the explorative effects of the thermal and quantum annealing processes respectively.

2.2.1 Hardware-Realized Quantum Annealing

The qubits used in the D-Wave QPU are loops of superconducting wire cooled to very low temperature. The energy function describing the physics is well approximated by the Ising model. The Ising parameters $\boldsymbol{h}$ and $\boldsymbol{J}$ are tuneable through the application of small magnetic fluxes induced by currents sent to the quantum processing unit (QPU).

The D-Wave QPU embodies a particular operator for H_X, and allows for the expression of a class of QUBO/Ising problems for H_P. A detailed physical description of the underlying hardware components can be found in *[Harris2010]*, *[Johnson2011]*. Due to the underlying physics the weighting of H_X and H_P is not linear in s, but has the form $A(s)H_X + B(s)H_P$, where $A(s)$ and $B(s)$ are referred to as envelope functions. [5] This still allows for controlled damping of the exploratory effects of: math:*H_X* through s.

The QPU, however, is less flexible regarding changes in temperature T. Though slight changes in T are possible, we shall consider the temperature fixed at a low, but non-zero value. Thus, runs of the hardware will not deterministically return the global minimum of $E_P(\boldsymbol{s}|\boldsymbol{h},\boldsymbol{J})$, but sample from a distribution giving maximal probability to the globally lowest energy configuration.

The subclass of Ising objectives solved by the current D-Wave hardware is restricted in a number of ways. In the next chapter we describe these limitations, and show how they may be circumvented.

[5] Our convention will be that $A(s)$ and $B(s)$ will have units of energy, and H_X and H_P will be dimensionless.}

CHAPTER

THREE

ISING PROBLEMS ADDRESSED NATIVELY BY THE QPU

Before describing the interfaces to the D-Wave QPU, we describe the limitations which restrict the class of Ising problems that are addressed natively by the QPU. Having described these limitations we show how they can be circumvented to solve general QUBO/Ising problems. In subsequent chapters we show how general discrete optimization problems may be brought into QUBO form.

3.1 Limitations on Hardware-Realized Ising Models

There are three fundamental limitation imposed by current generation hardware: small qubits numbers, sparse qubit connectivity, and limited parameter precision. We consider each of these constraints.

Qubits are a scarce resource on current hardware. Thus, problem formulations which make most effective use of qubits are highly preferred. Though, in principle 64 qubits might be allocated to represent a single floating point value, it currently makes more sense to consider problems where qubits to represent inherently discrete decision variables.

Different problems require different connectivity, and thus can have different primal graphs. While complete connectivity between qubits would eliminate this connectivity mismatch, the local nature of physical interactions makes complete connectivity difficult. The current hardware architecture provides a particular connectivity driven by engineering feasibility. Future QPU generations are likely to provide improved connectivity somewhat ameliorating this problem. In the longer term we envisage hardware connectivity specially tailored to certain problem domains. However, at present accommodating connectivity mismatch is critical for successful applications.

The hardware may be thought as a programmable analog device exploiting quantum effects. Consequently, there are limitations on the precision to which various parameters may be set. The algorithm designer must be aware of these limitations as they impact the types of problems that may be modeled.

3.1.1 Qubit number

The first limitation imposed by the hardware is the limited number of qubits supported. The number of qubits is a function of the size of the Chimera graph, which depends on the release of the QPU in the system. For example, a 12 x 12 x 4 matrix has 1152 quibits. In the best case, where each qubit is assigned to a different optimization variable, the QPU can solve up to 1152 variables. Problems larger than the number of qubits may still be tackled but only by breaking the larger problem up into a sequence of smaller optimizations.

3.1.2 Connectivity

D-Wave's QPU imposes restrictions on which Ising $J_{i,j}$ parameters may be different from 0. The allowed connectivity is described with a particular primal graph we have a called Chimera (primal graphs are defined in *The Ising Model*). An $M \times N \times L$ Chimera graph consists of an $M \times N$ two-dimensional lattice of blocks, with each block consisting of $2L$ variables for a total of $2MNL$ variables.

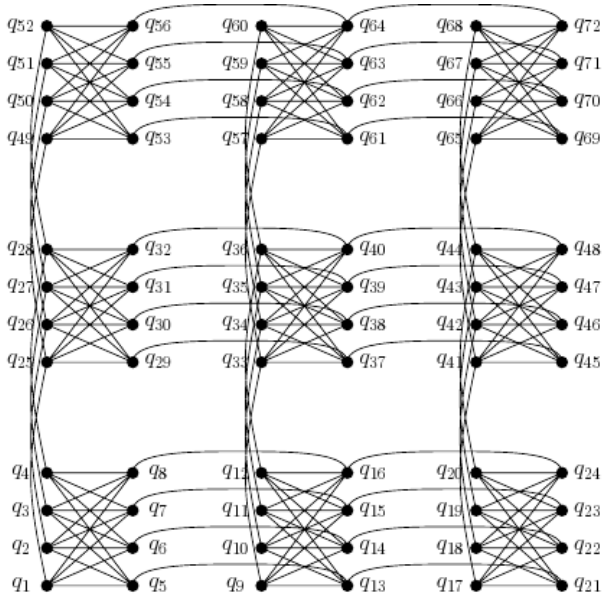

Fig. 3.1: Chimera(3,3,4).

chimeraGraph shows a $3 \times 3 \times 4$ Chimera graph. Nodes in a $M \times N \times L$ Chimera graph represent each of the $2MNL$ qubits (labeled as q_i in *chimeraGraph*). Edges (connections between nodes) in the graph indicate variable connections which may be non-zero. As an example, in *chimeraGraph* $J_{1,8}$ may be non-zero (because there is an edge connecting qubits 1 and 8), but $J_{1,2}$ must always be zero (because there is no edge connecting qubits 1 and 2). The basic repeating block of Chimera (a block of 8 variables with complete bipartite connectivity) may be tiled into an $M \times N$ lattice. The left hand variables within each block are connected vertically, and the right hand variable within the block are connected horizontally. A more formal description of the connectivity follows.

Chimera connections are best expressed using a different indexing notation for qubits. To index any qubit we use four numbers (i, j, u, k). (i, j) indexes the (row,column) of the block. i must be between 0 and $M - 1$ inclusive, and j must be between 0 and $N - 1$ inclusive. $u = 0$ indicates the left hand qubits in the block, and $u = 1$ indicates the right hand qubits. $k = 0, 1, \cdots, L - 1$ indexes qubits within either the left- or right-hand side of a block.

As an example, *chimeraIndexing* shows this 4-tuple indexing for a $5 \times 4 \times 4$ Chimera, and highlights the coordinates of two particular qubits. $u = 0$ nodes are indicated in blue, and $u = 1$ nodes in red. In this notation, the qubits indexed by (i, j, u, k) and (i', j', u', k') are neighbours if and only if

1. $i = i'$ and $j = j'$ and $[(u, u') = (0, 1)$ or $(u, u') = (1, 0)]$ OR
2. $i = i' \pm 1$ and $j = j'$ and $u = u'$ and $u = 0$ and $k = k'$ OR
3. $i = i'$ and $j = j' \pm 1$ and $u = u'$ and $u = 1$ and $k = k'$.

The first rule accounts for connections within a block, the second rule accounts for vertical connections between blocks, and the last rule accounts for horizontal connections between blocks. Variables indexed by a negative i or j are ignored. [1]

A 4-tuple index (i, j, u, k) (as used in *chimeraIndexing*) can be converted to a linear index ℓ (as used in *chimeraGraph*)

[1] The Chimera adjacency matrix $\mathbf{A}$ has matrix elements $A_{i,j,u,k;i',j',u',k'} = \delta_{i,i'}\delta_{j,j'}\delta_{u,u'+1} + \delta_{i,i'\pm1}\delta_{j,j'}\delta_{u,u'}\delta_{u,0}\delta_{k,k} + \delta_{i,i'}\delta_{j,j'\pm1}\delta_{u,u'}\delta_{u,1}\delta_{k,k'}$ where $\delta_{a,b}$ is the Kronecker delta function equal to 1 if $a = b$ and 0 if $a \neq b$. Addition on u is modulo 2. Expressed another way, $\mathbf{A} = \mathbf{I}_M \otimes \mathbf{I}_N \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \mathbf{1}_L + \mathbf{L}_M \otimes \mathbf{I}_N \otimes \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \mathbf{I}_L + \mathbf{I}_M \otimes \mathbf{L}_N \otimes \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \mathbf{I}_L$ where $\otimes$ is the matrix tensor product, $\mathbf{I}_n$ is the $n \times n$ identity, $\mathbf{1}_n$ is the $n \times n$ matrix of 1's, and $\mathbf{L}_n$ is the $n \times n$ adjacency matrix for the graph representing a line of n nodes each connected to its left and right neighbours.

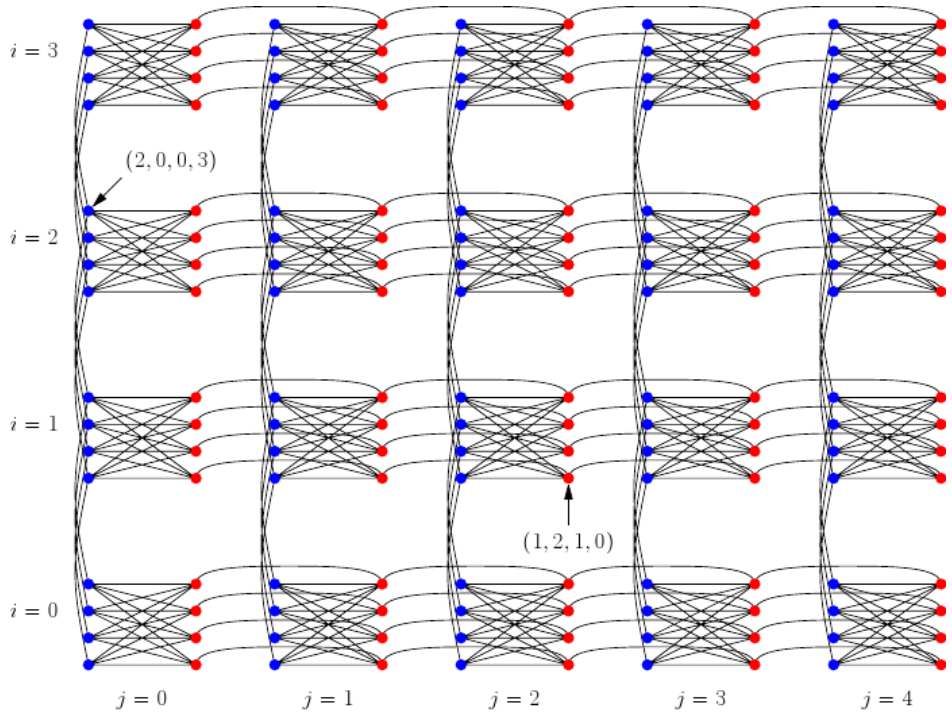

Fig. 3.2: Chimera indexing.

according to the formula

$$\ell(i,j,u,k) = 1 + 2NLi + 2Lj + Lu + k$$

where $M \times N \times L$ is the size of the Chimera lattice. We supply functions to make these translations.

The $M \times N \times L$ Chimera graph is bipartite, non-planar, and has a tree-width equal to $L \min(M, N)$. If $L \min(M, N) \leq 25$ then Chimera problems may be solved using tree decomposition approaches [Coughlan2009]. Larger Chimera lattices however prevent the tree decomposition from being used to solve the optimization exactly, though heuristics based upon the tree decomposition may be constructed. The Chimera software solvers we provide are based on this idea.

If a problem has a primal graph which is not a subgraph of Chimera then this must be worked around in order to solve the problem.

3.1.3 Precision

The precision to which values assigned to each edge $J_{i,j}$ or each vertex h_i can be specified is not arbitrary. Due to the analog manner in which the underlying physical model represents any h_i or $J_{i,j}$ there is always additive noise which contaminates desired problem parameters. This noise cannot be measured directly, but we can quantify it approximately as follows:

The hardware has a limited dynamic range within which parameters may be set, *i.e.* $\underline{h} \leq h_i \leq \overline{h}$ and $\underline{J} \leq J_{i,j} \leq \overline{J}$ (with $\underline{h} < 0, \overline{h} > 0, \underline{J} < 0, \overline{J} > 0$). An arbitrary h or J value within these ranges may be specified, but when realized within the hardware that the actual value seen by the energy minimizing dynamics is a sample from a Gaussian distribution centered on the requested value but having some standard deviation. More precisely, if we wish to realize a particular value $\mathfrak{h}_i$ what is realized on the QPU is a sample from the Gaussian distribution $N(\cdot|\mathfrak{h}_i, \sigma_h)$. Similarly, a sample from a coupling parameter $\mathfrak{J}_{i,j}$ is realized as a draw from $N(\cdot|\mathfrak{J}_{i,j}, \sigma_J)$.

Current generation QPUs have $[\underline{h}, \overline{h}] \approx [-1, 1]$ and $[\underline{J}, \overline{J}] \approx [-1, 1]$, though this differs from QPU to QPU.

Some Ising/QUBO models can be sensitive to limited parameter ranges. This problem can be particularly acute for some constrained optimization problems where large parameter values are necessary to impose penalties on constraint violations, and where fine-grained parameter differences are necessary for the optimization criterion. Constraint satisfaction problems (problems having no optimization criterion) thus tend to require less precision. To make maximal use of limited precision it is suggested that users calling the hardware employ the auto-scale mode of operation which utilizes the largest possible parameter ranges.

3.2 Circumventing Hardware Limitations

Fortunately, the three hardware limitations discussed in the previous sections may be overcome (though there is a cost involved), and the D-Wave hardware can be used, in principle, to address arbitrarily-sized Ising problems. We consider approaches to ameliorate each of the qubits number, connectivity, and precision limitations.

3.2.1 Qubit Number

To solve problems with more variables than qubits there is no choice but to break the large problem into subproblems, solve the subproblems, and somehow reconstruct an answer to the original problem from the subproblem solutions. Divide-and-conquer algorithms have a rich history in computer science for problems of this type and can be adapted to address the limited numbers of qubits available in early versions of QA hardware. We outline three approaches by which large optimization problems can be decomposed. Decomposition approaches tailored to specific problem types can also be developed, but here we consider only general-purpose approaches.

All the decomposition approaches we consider rely on conditioning (or assigning) a subset of variables to particular values in their domain. If A indicates a set of variables that are unconditioned, and $\backslash A$ indicates the remaining variables conditioned to certain values, then an Ising objective $E(\boldsymbol{s})$ can be minimized in two stages:

$$\min_{\boldsymbol{s}} E(\boldsymbol{s}) = \min_{\boldsymbol{s}_A, \boldsymbol{s}_{\backslash A}} E(\boldsymbol{s}_A, \boldsymbol{s}_{\backslash A}) = \min_{\boldsymbol{s}_{\backslash A}} \mathcal{E}(\boldsymbol{s}_{\backslash A})$$

where

$$\mathcal{E}(\boldsymbol{s}_{\backslash A}) = \min_{\boldsymbol{s}_A} E(\boldsymbol{s}_A, \boldsymbol{s}_{\backslash A}) = E\big(\boldsymbol{s}^\star_A(\boldsymbol{s}_{\backslash A}), \boldsymbol{s}_{\backslash A}\big).$$

$\boldsymbol{s}^\star_A(\boldsymbol{s}_{\backslash A}) = \operatorname{argmin}_{\boldsymbol{s}_A} E(\boldsymbol{s}_A, \boldsymbol{s}_{\backslash A})$ is the optimal setting of $\boldsymbol{s}_A$ for a given $\boldsymbol{s}_{\backslash A}$. Conditioning makes the $\min_{\boldsymbol{s}_A}$ problem smaller and simpler than the original problem. The primal graph of this conditioned problem over the vertices $\boldsymbol{s}_A$ of $E(\boldsymbol{s}_A, \boldsymbol{s}_{\backslash A})$ is formed from the primal graph of the original problem by removing all conditioned variables and all edges attached to a conditioned variable. This can make the $\min_{\boldsymbol{s}_A}$ problem dramatically simpler. Of course, we still must identify the lowest energy setting of the conditioned variables. The outer loop optimization determining the conditioned variable settings is the $\min_{\boldsymbol{s}_{\backslash A}} \mathcal{E}(\boldsymbol{s}_{\backslash A})$ problem. The primal graph for the $\mathcal{E}(\boldsymbol{s}_{\backslash A})$ problem has connections between variables in $\boldsymbol{s}_{\backslash A}$ which have been induced though the elimination (minimization) of the $\boldsymbol{s}_A$ variables, i.e. through the dependence on $\boldsymbol{s}^\star_A(\boldsymbol{s}_{\backslash A})$. In summary, we cast the $\min_{\boldsymbol{s}} E(\boldsymbol{s})$ problem as the smaller problem $\min_{\boldsymbol{s}_{\backslash A}} \mathcal{E}(\boldsymbol{s}_{\backslash A})$ where $\mathcal{E}(\boldsymbol{s}_{\backslash A})$ can be evaluated at any $\boldsymbol{s}_{\backslash A}$ using the hardware QA to effect the minimization over $\boldsymbol{s}_A$. The outer optimization over $\boldsymbol{s}_{\backslash A}$ is carried out using conventional software algorithms (and not QA).

In certain situations the smaller $\min_{\boldsymbol{s}_A}$ subproblem may be further reduced in size through techniques which infer values for the unconditioned variables based on either side constraints or bounding information. Different choices for the conditioning sets A (which may vary during the course of the algorithm) and different inference techniques give different algorithms. Most stochastic local search algorithms rely entirely on conditioning and utilize no inference. In contrast, complete search methods like branch-and-bound which sequentially fix variables rely heavily on inference to prune their search spaces.

Cut-Set Conditioning

Cut-set condition is a simple decomposition method that chooses a subset of variables $\boldsymbol{s}_A$ (which remains fixed throughout the course of the algorithm) so that the resulting subproblem $\min_{\boldsymbol{s}_A} E(\boldsymbol{s}_A, \boldsymbol{s}_{\setminus A})$ decomposes into a set of disjoint subproblems. In this case the primal graph of $E(\boldsymbol{s}_A, \boldsymbol{s}_{\setminus A})$ breaks into separate components indexed by α which can be solved independently. Formally:

$$\begin{aligned}
\mathcal{E}(\boldsymbol{s}_{\setminus A}) &= \min_{\boldsymbol{s}_A} E(\boldsymbol{s}_A, \boldsymbol{s}_{\setminus A}) \\
&= \min_{\boldsymbol{s}_A} \left\{ E_{\setminus A}(\boldsymbol{s}_{\setminus A}) + \sum_\alpha E_\alpha(\boldsymbol{s}_{\setminus A}, \boldsymbol{s}_{A_\alpha}) \right\} \\
&= E_{\setminus A}(\boldsymbol{s}_{\setminus A}) + \sum_\alpha \min_{\boldsymbol{s}_{A_\alpha}} E_\alpha(\boldsymbol{s}_{\setminus A}, \boldsymbol{s}_{A_\alpha}) \\
&= E_{\setminus A}(\boldsymbol{s}_{\setminus A}) + \sum_\alpha \mathcal{E}_\alpha(\boldsymbol{s}_{\setminus A})
\end{aligned}$$

where $\boldsymbol{s}_A = \bigcup_\alpha \boldsymbol{s}_{A_\alpha}$ and $\boldsymbol{s}_{A_\alpha} \cap \boldsymbol{s}_{A_{\alpha'}} = \emptyset$ for $\alpha \neq \alpha'$. With a proper choice of the conditioning variables the primal graph is cut into disjoint partitions which are optimized with respect to $\boldsymbol{s}_{A_\alpha}$ independently. The set of conditioned variables which cuts the original problem into pieces is called the cut set. We choose a cut-set such that each of the remaining $\min_{\boldsymbol{s}_{A_\alpha}}$ problems is small enough that it can be solved in the D-Wave hardware. A small example is illustrated in *figCutsetFig* and *figCutsetGraph*.

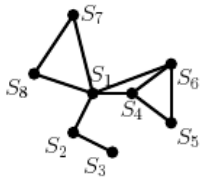

Fig. 3.3: An 8-variable primal graph.

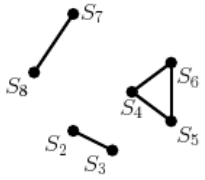

Fig. 3.4: Primal graph after conditioning.

figCutsetFig shows the primal graph of an 8 variable problem. We note that if we conditioned on variable S_1 the primal graph decomposes into 3 smaller subproblems (see *figCutsetGraph*). The remaining variable sets $\boldsymbol{S}_A = \{S_2, S_3\} \cup \{S_4, S_5, S_6\} \cup \{S_7, S_8\}$ define three decoupled subproblems.

The outer optimization problem which determines optimal values for the cut set variables (S_1 in the previous example) is carried out with a classical heuristic, perhaps greedy local search, tabu search, or simulated annealing. Alternatively, a complete search method may be used to give the global minimum of $\mathcal{E}(\boldsymbol{s}_{\setminus A})$. Note however, that because the inner optimizations over $\boldsymbol{s}_A$ come with no proof of optimality we have no guarantees that the global minimum of the original problem is attained.

To make the outer optimization as easy as possible, it is desirable to have the number of conditioned variables as small as possible so that the outer optimization is as small as possible. Nevertheless, we still need to ensure that the choice of cut-set variables shatters the problem into subproblems which are sufficiently small so as to be solved by the hardware. Determining the smallest cut-set is a graph theoretic problem which requires finding a set of vertex separators which

disconnects the original primal graph into sufficiently small disconnected components. Finding a minimal set of vertex separators is NP-hard in general, so, fast heuristics must be employed. [2]

To connect this approach to previous proposals for problem decomposition we comment on an alternative criteria to determine the set $\mathbf{s}_{\setminus A}$ of conditioned variables. Since tree-like primal graphs can be globally minimized in polynomial time, we might also determine the set $\mathbf{s}_{\setminus A}$ so that the resulting graph, once the conditioned variables and their edges are removed is either a tree or collection of disconnected trees. In this case the subproblems are all tree-like and can be solved quickly. This is an old idea and first appeared in a different context *[Dechter1987]*. However, the cutsets need to form trees will typically be larger than the cut sets needed to form hardware-sized subproblems, so there is a significant advantage in using hardware to solve subproblems.

Branch-and-Bound

Branch-and-bound algorithms progressively condition more and more variables to particular values. At a point in the search, variable s_i must be conditioned to either $s_i = -1$ or $s_i = 1$. This branching possibility defines a split at node s_i. At this node, some variables have been conditioned to values and some remain free. Additional splits on free variables define a branching binary tree of possibilities. The leaves of this tree define the 2^n configurations where all variables have been assigned values. At any node, both branches of the split must be explored, but branch-and-bound algorithms prune branches that can be shown not to contain the global optimum leaf node. This pruning is accomplished by maintaining upper and lower bounds to the global minimum of $E(\boldsymbol{s})$. At any point in the search the upper bound is the smallest value $E(\boldsymbol{s})$ that has been seen at the visited leaves. The lower bound at a node in the search is calculated as a lower bound on $\min_{\boldsymbol{s}_A} E(\boldsymbol{s}_A, \boldsymbol{s}_{\setminus A})$ where $\boldsymbol{s}_A$ is the set of as-yet-unconditioned variables. If the lower bound at a node exceeds the current upper bound then there is no need to search deeper than this node as there cannot be a leaf in the subtree rooted at the node which will have lower energy than the current best upper bound. The efficacy of this pruning of subtrees is determined by the tightness of the lower bound approximation, and by the ordering in which variables are progressively conditioned.

Branch-and-bound methods can benefit from the quantum annealing hardware solver by terminating search higher in the tree of variable assignments. Once sufficient variables have been conditioned so that the remaining unconditioned variables can be optimized by the hardware there is no need to explore deeper - simply call the hardware to approximate the best completion from that node. Of course, since the hardware does not come with a proof of optimality, we lose the ability to guarantee that this algorithm returns the global minimum.

An alternative avenue to incorporate the hardware is to use quantum annealing to provide tight lower bound functions at any node in the search tree. Lagrangian relaxation is one way to find these lower bounds.

Lagrangian relaxation relies on a different simplification of the primal graph, and yields a lower bound on the global minimum objective value. In many cases the bound can be made quite tight. Consider a node in the primal graph representing a variable S_i. We split the node in two and define $S_i^{(1)}$ and $S_i^{(2)}$ and add the constraint that $S_i^{(1)} = S_i^{(2)}$. This leaves the problem unchanged. The original objective $E(s_i, \boldsymbol{s}_{\setminus i})$ is modified to $E'(s_i^{(1)}, s_i^{(2)}, \boldsymbol{s}_{\setminus i})$. If we ignore the equality constraints amongst split variables, then with sufficiently many split variables the modified objective E' will decompose into smaller independent problems. To ensure that these two objectives give the same value when the $S_i^{(1)} = S_i^{(2)}$ constraint is satisfied we require $h_i = h_i^{(1)} + h_i^{(2)}$. The advantage of splitting variable S_i comes when the equality constraint is softened and treated approximately.

Introducing a multiplier λ_i for the equality constraint the Lagrangian for the constrained problem is

$$L(s_i^{(1)}, s_i^{(2)}, \boldsymbol{s}_{\setminus i}; \lambda_i) = E'(s_i^{(1)}, s_i^{(2)}, \mathbf{s}_{\setminus i}) + \lambda_i(s_i^{(1)} - s_i^{(2)}).$$

Minimizing this Lagrangian for any fixed value of λ_i provides a lower bound to $\min_{\boldsymbol{s}} E(\boldsymbol{s})$. The dual function is defined as

$$g(\lambda_i) = \min_{s_i^{(1)}, s_i^{(2)}, \boldsymbol{s}_{\setminus i}} L(s_i^{(1)}, s_i^{(2)}, \boldsymbol{s}_{\setminus i}; \lambda_i),$$

and maximizing the dual with respect to λ_i provides the tightest possible lower bound.

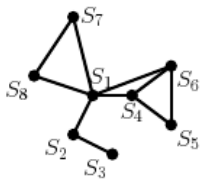

Fig. 3.5: An 8-variable primal graph.

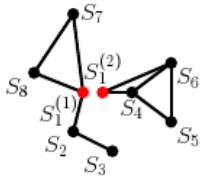

Fig. 3.6: Primal graph after splitting

As an example consider again the small 8 variable problem defined in *figSplitFig*. The Lagrangian relaxed version of the problem obtained by splitting variable S_1 resulting in *figSplitGraph*. The constraint $S_1^{(1)} = S_1^{(2)}$ is treated softly giving two independent subproblems consisting of variable sets $S_1^{(1)}, S_2, S_3, S_7, S_8$ and $S_1^{(2)}, S_4, S_5, S_6$. If either of the subproblems is still too large then they can be decomposed further either through another variable split or through conditioning.

We introduce as many replicated variables as needed in order to generate subproblems sufficiently small so that they can be solved in hardware. These subproblems are solved to give the dual function which is then optimized using a subgradient method to provide the tightest possible lower bound bounds. [3] An alternative to subgradient optimization is proposed in [Johnson2007] where a smooth approximation to dual function is considered.

Since the quantum annealing may return non-optimal results, the lower bounds returned by Lagrangian relaxation may actually be too high in which case branch-and-bound may mistakenly prune branches containing the global minimum. However, viewing this Lagrangian-relaxed branch-and-bound algorithm as a heuristic, we may expect good results if the quantum annealing is effective at locating low energy states.

During the branch-and-bound process we can also take advantage of the cut-set decompositions discussed in *graphDecompSect*. Note that as the search moves between nodes in the search tree, the primal graph for the subproblem which must be solved changes. In particular, the vertices in the graph corresponding to the fixed variables at the search tree node are removed along with all edges connected to these vertices. At some nodes in the search tree the primal graph may become disconnected in which case we solve multiple smaller subproblems. These subproblems and their minimizers may be cached so that when these same subproblems are generated at other nodes the results can be looked up. Results with such and/or depth first search for 0/1 integer programs are presented in *[Marinescu2007]*. In that paper alternative methods for exploring the search tree including dynamic variable orderings and best-first orderings are also considered. All of these improvements may also be applied in this setting.

Large Neighbourhood Local Search

Local search algorithms improve upon a candidate solution $\boldsymbol{s}^t$ available at iteration t by searching for better solutions within some local neighbourhood of $\boldsymbol{s}^t$. Most commonly, the neighbourhood for bitstrings is the Hamming neighbourhood consisting of all bitstrings which differ from $\boldsymbol{s}^t$ in a single bit. By adopting the improvement, if any, found within the neighbourhood as the next configuration, the search progressively finds better and better solutions. Search terminates when there are no better solutions within the neighbourhood. Such configurations are locally optimal with

[2] The ranges $[\underline{h}, \overline{h}]$ and $[\underline{J}, \overline{J}]$ of any QPU may be queried through an application programming interface call to the QPU.

[3] One effective way to find a small set of vertex separators is to apply a graph partition heuristic like Metis http://glaros.dtc.umn.edu/gkhome/views/metis, and then build a cut-set of vertex separators by selecting a vertex cover from the edges that span partitions.

respect to the neighbourhood. After trapping in local optima additional heuristics may be applied to enable escape and continued exploration of the search space.

Quantum annealing hardware can be very simply combined with local search to allow the local search algorithm to explore much larger neighbourhoods than the 1-bit-flip Hamming neighbourhood. As the size of the neighbourhood increases, the quality of the resultant local minima improves. Consider a problem of n Ising variables, and define the neighbourhood around configuration $\boldsymbol{s}^t$ as all states within Hamming distance d (with $d < n$) of $\boldsymbol{s}^t$. Searching within this neighbourhood for a configuration $\boldsymbol{s}^{t+1}$ having energy lower then $E(\boldsymbol{s}^t)$ is a smaller optimization problem than the original, but it still cannot be directly carried out by quantum annealing hardware. Instead we choose one of the $\binom{n}{d}$ subsets of d variables. Call the selected subset A and note that $|A| = d$. Having selected A we determine the best setting for these $\boldsymbol{s}_A$ variables given the fixed context of the conditioned variables $\boldsymbol{s}^t_{\setminus A}$. We select d such that the smaller d-variable problem can be embedded within the quantum annealing hardware. If no improvement is found within the chosen subset another subset of size d may be selected. This algorithm stagnates at a configuration which is a local optimum with respect to all d variable spin flips. However, if d and n are both large there are great many size-d subsets around each configuration $\boldsymbol{s}^t$ so trapping in d-optimal local minima is rarely attained. However, progress can be slow as mainly unpromising neighbourhoods are explored. For Ising problems, it is possible to infer variable subsets that need not be considered, and heuristics may be defined which generate promising subsets. *[Liu2005]* presents promising results for even small neighbourhoods of size $d \leq 4$. With hardware solvers, much larger neighbourhoods may be considered.

3.2.2 Connectivity

Even if the hardware has more qubits than an Ising optimization problem has variables the problem may not be solvable in hardware. Not only is a sufficient number of qubits necessary, but the qubits must also be connected in the same manner as the primal graph of the original Ising problem. Given the fixed Chimera connectivity of the D-Wave hardware and the potentially arbitrary connectivity of real world problems, this seems to be a significant problem. Fortunately, there is a way to mimic arbitrary hardware connectivity at the expense of using qubits as connectors rather than optimization variables.

In Lagrangian relaxation we split an optimization variable s_i into two variables $s_i^{(1)}$ and $s_i^{(2)}$ and added the constraint $s_i^{(1)} = s_i^{(2)}$ (which is subsequently relaxed). A similar trick can be used to obtain hardware connectivity that is not inherent in Chimera. In this case an optimization variable s_i is mapped into a set of one or more qubits $\{q_i^{(1)}, \cdots, q_i^{(k)}\}$ (qubits are assumed to be ± 1 valued). Since all qubits represent the same problem variable we must impose the constraint $q_i^{(j)} = q_i^{(j')}$ for all pairs (j, j') occurring as edges in a spanning tree across the qubits. The equality constraint $q_i^{(j)} = q_i^{(j')}$ can be encoded as the Ising penalty:

$$-M q_i^{(j)} q_i^{(j')}$$

where $M > 0$ is the weight of the penalty. If M is sufficiently large the lowest energy state will always have $q_i^{(j)} = q_i^{(j')}$ because that feasible assignment is $2M$ lower in energy than the infeasible assignment $q_i^{(j)} \neq q_i^{(j')}$. In this way strings of qubits are slaved to each other to effectively create ‘‘wires” which can connect arbitrary Chimera vertices. To create these wire-like connectors we must weigh the penalties large enough so that low energy configurations do not violate the equality constraints. Balancing this we’d like the smallest possible penalty weight which enforce the constraints to prevent precision issues, and to foster efficient exploration of the search space. An iterative procedure which incrementally updates weights until the equality constraints are satisfied is effective, and code is available in the utilities packages to accomplish this.

For example, if qubits q_5 and q_{33} represent problem variables s_1 and s_2 respectively and there is a (s_1, s_2) edge in the problem primal graph, then this coupling can be realized by representing problem variable s_1 with the string of qubits $\{q_5, q_{13}, q_9\}$ (see *qubitPathFig*). Edge weights of strength $-M$ couple qubits along the bold black edges. The string of qubits representing s_1 is now connected to s_2 through the blue edge. Note also that in mapping a single optimization variable to a connected set of qubits in Chimera we can mimic higher degrees of connectivity than are present in Chimera. For example, variable s_1 represented by qubits $\{q_5, q_{13}, q_9\}$ can be connected to up to 12 other

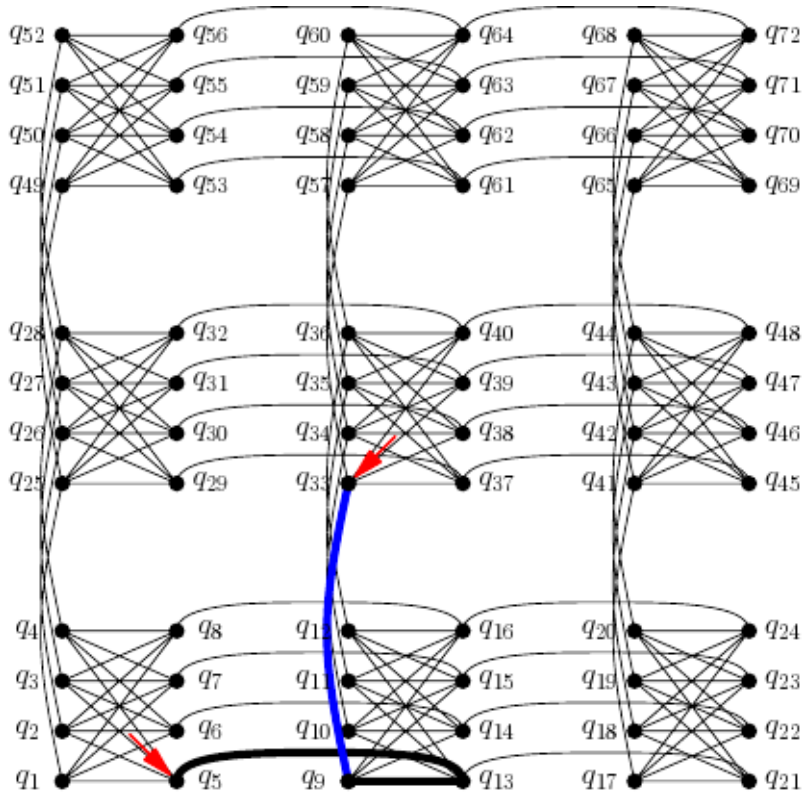

Fig. 3.7: A qubit chain used for connectivity.

qubits $\{q_1, q_2, q_3, q_4, q_{10}, q_{11}, q_{12}, q_{14}, q_{15}, q_{16}, q_{21}, q_{33}\}$, and thus up to potentially 12 other optimization variables (if each qubit represented a distinct optimization variable). Thus, the connectivity can be effectively doubled in this case.

What does this variable to qubit mapping observation say about the general problem of solving an arbitrary problem on Chimera? To answer this question we introduce a notion from graph theory. A *minor* of a graph is formed from a sequence of edge contractions on the graph. An edge contraction is the removal of an edge (v_1, v_2) and fusion of the vertices v_1 and v_2. If the sets of neighbouring vertices of v_1 and v_2 in the original graph are $\mathcal{N}_1$ and $\mathcal{N}_2$ then the neighbours of the fused node in the edge contracted graph are $(\mathcal{N}_1 \cup \mathcal{N}_2) \setminus \{v_1, v_2\}$. With this notion we can say when an Ising problem may be embedded into Chimera. *If G is the primal graph of an Ising problem then it can be embedded into Chimera, if G is isomorphic to a graph minor of Chimera.*

While the graph minor embedding method tells us how to embed problems into Chimera, in principle, there is significant complexity in the process. We need to know both the isomorphism from G into the particular Chimera minor, and we need to know which of the many Chimera minors to select. The choice of minors is important because bad embeddings are wasteful of qubits. Qubits acting as connectors cannot be used as problem variables so the effective size of the Ising problem that can be solved is reduced. Thus, we want embeddings with fewest numbers of qubits wasted as wires.

It is worthwhile to more formally specify the embedding problem as tools are available which can assist in finding embeddings. Let $G = (V, E)$ be a primal graph for a problem we want to embed into hardware. Fundamentally, there are two requirements:

- a given vertex $v \in V$ must be mapped to connected set of qubits $q \in$ Chimera
- all edges $(v_1, v_2) \in E$ must be mapped to at least one edge in Chimera.

To satisfy the first constraint we posit an unknown table EmbedDist(v, q, d) which maps vertex v to qubit q and is distance d from a reference qubit also mapped from v. Additionally, it is convenient to define Embed$(v, q) \leftrightarrow$

$\exists d\,\text{EmbedDist}(v,q,d)$. The constraints on these tables are then: [4]

1. $\forall v,q,d\,\text{EmbedDist}(v,q,d) \;\rightarrow\; \big(d=0\big) \vee \big(\exists v_1,q_1\,\text{Chimera}(q,q_1) \wedge \text{EmbedDist}(v_1,q_1,d-1)\big)$
2. $\forall v_1,v_2\; E(v_1,v_2) \;\rightarrow\; \exists q_1,q_2\,\text{Chimera}(q_1,q_2) \wedge E(v_1,q_1) \wedge \text{Embed}(v_2,q_2)$.

The predicate $\text{Chimera}(q_1,q_2)$ is true if and only if there is a hardware edge between qubits q_1 and q_2. Similarly, $E(v_1,v_2)$ is true if and only if there is an edge in the primal graph between problem variables v_1 and v_2. In addition to these two constraints there are the bookkeeping constraints:

3. Every v is mapped: $\forall v \exists q\,\text{Embed}(v,q)$,
4. At most one v is mapped to a given q: $\forall v,q\,\text{Embed}(v,q) \rightarrow \neg\exists v_1\,(v_1 \neq v) \wedge \text{Embed}(v_1,q)$. Using aggregate predicates, the second bookkeeping constraint can also be specified as $\forall q\,\text{COUNT}\big(v,1,\text{Embed}(v,q)\big) \leq 1$. [5]
5. There is 1 root for each v: $\forall v\,\text{COUNT}(q,1,\text{EmbedDist}(v,q,0)) = 1$.

Note that this specification does not do optimization to minimize the total number of qubits used. However, when quantifying over d (the size of the connected Chimera graphs representing problem variables) we must specify a maximal value D so that $0 \leq d \leq D$. In this way the size of the embeddings may be somewhat controlled. Keep in mind though that for some graphs (specified through $E(v_1,v_2)$) the problem may be infeasible if D is too small.

Embedding Complete Graphs

Since a completely connected graph on n vertices (all vertices connected to all other vertices) can solve all Ising problems defined over n variables, it is natural to ask for the largest complete graph K_n which is a minor of a $M \times N \times L$ Chimera graph. Knowing this complete minor allows for immediate embedding of all problems of n or fewer variables into Chimera. The largest complete minor has $n = 1 + L\min(M,N)$ vertices. [6] *minorFig1* shows this graph minor for a small $2 \times 2 \times 4$ Chimera graph. *minorFig2* shows a similar result for $4 \times 4 \times L$ Chimera.

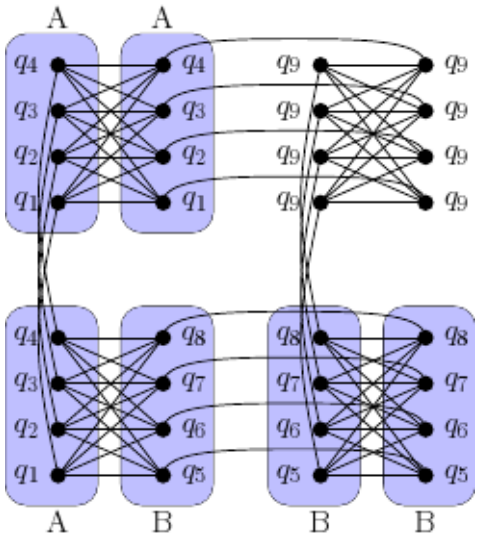

Fig. 3.8: Embedding of K_9 into $2 \times 2 \times 4$ Chimera; A and B indicate two different blocks of 4 variables. Equivalently labelled qubits are connected by edge penalties.

[4] A concise introduction to subgradient methods may be found in the class notes of Stephen Boyd available at *[Boyd2007]*.

[5] This constraint specification is executable, and solvers are available which can solve for the unknown tables $\text{EmbedDist}(v,q,d)$ and $\text{Embed}(v,q)$ from the constraint specification.

[6] The aggregate operator $\text{COUNT}(v,1,\phi(v))$ counts the values in the domain of v for which predicate $\phi(v)$ is true.

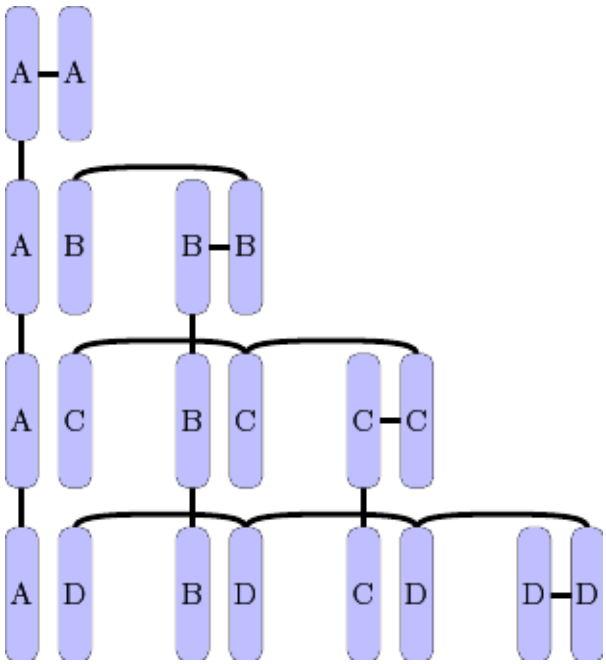

Fig. 3.9: Condensed representation of the complete graph of size K_{4L+1} on $4 \times 4 \times L$ Chimera. Each block A,B,C,D consists of L qubits, and the thick black edges ensure that consistent qubit values are obtained in similarly labeled blocks. Each block is connected to all other blocks, and to a common qubit not drawn in the upper triangular part of the lattice. If this figure were to be expanded, the top left corner consisting of the A and B blocks alone would appear as in *minorFig1* if $L = 4$.

Thus, connectivity issues may be worked around at the cost of larger Chimera lattices. Note also, that problems with specific connectivity requirements can often have problem specific embeddings that make more effective use of qubits than using Chimera as the complete graph on n vertices.

Approximate Embedding of Ising Problems into Chimera

In some cases the Ising objective function $E(\boldsymbol{s})$ is suggestive only, and the exact form of the problem is malleable. One important example arises in learning problems. The true objective in predictive modeling is to minimize the error on future observations so that they are accurately predicted. However, not knowing what points will be observed in the future there is little choice but to minimize the errors on a training set of observed examples. However, minimizing training set error runs the risk of over fitting and additional terms favouring ‘‘simpler” models over complex models are typically added to the objective. Thus, there is latitude in the precise objective chosen. In such cases we can embed problems more simply.

Let $G = (V, E)$ indicate the $M \times N \times L$ Chimera graph. We embed an Ising objective of n variables into Chimera (with $|V| \geq n$) with a mapping $\varphi : \{1, \ldots, n\} \mapsto V$ such that $J_{i,j} \neq 0 \Rightarrow (\varphi(i), \varphi(j)) \in E$. In other words, each nonzero J_{ij} must be assigned to an edge in Chimera. The simplifying assumption we make here requires that each node in the original primal graph is mapped to a single node in Chimera.

The mapping φ is encoded with a set of binary variables $m_{i,q}$ which indicate that problem variable i is mapped to Chimera node q. For a valid mapping, we require $\sum_q m_{i,q} = 1$ for all problem variables i, and $\sum_i m_{i,q} \leq 1$ for all Chimera nodes q. Since the original problem can be altered, we force fit the problem into Chimera by maximizing the total magnitude of $J_{i,j}$ mapped to Chimera edges, i.e. we seek:

$$\boldsymbol{m}^\star = \underset{\boldsymbol{m}}{\operatorname{argmax}} \sum_{i>j} \sum_{(q,q') \in E} |J_{i,j}| m_{i,q} m_{j,q'}$$

where $\boldsymbol{m}$ is subject to the mapping constraints. This problem is a variant of the NP hard quadratic assignment problem. In the interest of linear time embedding we apply a greedy heuristic to approximately maximize the objective.

As a starting point, let $i_1 = \text{argmax}_i \sum_{j<i} |J_{j,i}| + \sum_{j>i} |J_{i,j}|$. So i_1 is the row/column index of J with the highest sum of magnitudes ($\boldsymbol{J}$ is assumed to be upper-triangular). We map i_1 to one of the Chimera vertices of highest degree. For the remaining variables, assume that we have defined φ on $\{i_1, \ldots, i_k\}$ such that $\varphi(i_j) = q_j$ where q_j is some Chimera qubit. Then assign $\varphi(i_{k+1}) = q_{k+1}$, where $i_{k+1} \notin \{i_1, \ldots, i_k\}$ and $q_{k+1} \notin \{q_1, \ldots, q_k\}$ to maximize the sum of all $|J_{i_{k+1},i_j}|$ and $|J_{i_j,i_{k+1}}|$ over all $j \in \{1, \ldots, k\}$ for which (q_j, q_{k+1}) is a Chimera edge. This fast greedy heuristic performs well. As an example, on a class of 128 variable Ising problems generated in a machine learning application, the greedy heuristic managed to map about 11% of the total absolute edge weight $\sum_{i,j} |J_{i,j}|$ of a fully connected graph into Chimera. The run time of the greedy heuristic was milliseconds. For comparison, a tabu heuristic designed for this matching quadratic assignment problem performs marginally better attaining about 11.6% total edge weight after running for 5 minutes.

3.2.3 Precision

The last limitation we address is the finite precision available on $\boldsymbol{h}$ and $\boldsymbol{J}$. An approach to this problem falls out of the solution to connectivity limitations. Recall, the following difficulty: an Ising problem may require a large range of J or h values, and simultaneously the lowest energy states may be sensitive to small misspecifications in J or h. When problems are scaled to lie within the range available in hardware these issues can cause problems. We consider two ways to ameliorate this difficulty. [7]

Limiting Parameter Ranges through Embedding: Minimizing the range of on-QPU J or h values improves hardware results. Improvements to parameter ranges can be simply obtained in some cases by exploiting problem embeddings onto the hardware. Recall that problem embeddings may require that a problem variable is represented by many qubits. For example, let us say problem variable s_i has the largest parameter value h_i, and that s_i is mapped to qubits q_i^1, q_i^2, and q_i^3. Since these three qubits will have the same value in any feasible solution, then we can share h_i across the three qubits, *i.e.* $h_i s_i \rightarrow (h_i/3)(q_i^1 + q_i^2 + q_i^3)$. Now the h_i has been reduced by a factor of 3. In a similar way coupling parameters $J_{i,j}$ may also be shared. In any embedding there may be multiple edges between connected chains of qubits representing problem variables. Again in this case the edge weight between these problem variables may be shared across all edges where coupled qubits represent logical variables. Thus, precision can be enhanced at the cost of using extra qubits to build up the desired interaction strengths. As part of our software utility packages, there is code to split parameter weight across qubits and edges.

Limiting Parameter Ranges by Simplifying the Problem: In some cases large parameter values allow for additional simplifications. Image a problem with interaction $h_i s_i$ where $h_i > 0$ is much much larger than all other problem parameters. It is quite likely that in low energy states $s_i = -1$ (which is $2h$ lower in energy than the alternative $s_i = +1$). In cases like this we may be able to prove (in polynomial time) that in the lowest energy state that certain variables must assume certain values. Such variables may then be eliminated from the problem removing the need to represent the large parameter h_i. For this reason, it is often useful to preprocess a problem to determine if we can infer values about certain variables. Code in the utility libraries does this rapidly. It must be recognized however that for any given problem we may not be able to infer the values for any variables and the problem may not simplify. However, such inferences can be made rapid enough that there is little overhead in attempting to simplify every problem before sending it to the hardware.

Problem Selection Lastly, we comment that precision limitations may also be used to assess how suitable a problem is for hardware QA solution. Problems of a combinatorial nature (e.g. finding a graph with certain properties, find matchings) usually require less precision than quantitative optimization problems. For these problems, precision concerns are rarely as important as the numbers of qubits and connectivity constraints. Likewise, an Ising problem having parameters specified to high precision is likely to be problematic in hardware, as the lowest energy state *may* be sensitive to this precision.

[7] That this is the largest complete minor follows immediately from the tree-width of Chimera being $L \min(M, N)$.

3.3 Effects on Problem Difficulty

The remedies we have outlined above allow for the solution of arbitrary Ising problems in hardware by transforming the problem to an alternative form that preserves the low energy states. In particular, qubits are introduced and constraints added which force sets of qubits to take identical values when measured. These constraints are treated through penalties which penalize infeasible configurations. Such penalties introduce energy barriers and may make exploration of the search space more difficult for quantum annealing. As a result, the effects of these transformations may well affect the efficacy of quantum annealing. As more is learned about these effects certain preferred transformations may be identified, but we will not consider these nuances further here.

CHAPTER

FOUR

IDIOMS FOR DISCRETE OPTIMIZATION WITH QUBO OBJECTIVES

4.1 Other Modeling Representations

Having shown how arbitrary Ising problems may be solved on D-Wave's hardware we now show how other constraint satisfaction and optimization problems may be converted to Ising or QUBO form. To begin, we show another common modeling representation that is simply the Ising model in disguise. This alternative representation is then generalized to variable interactions beyond pairwise which in many case is the most natural way to model some problems. We show how these higher order interactions may be systematically reduced to second order. In particular, constraints involve large groups of variables and we provide a systematic way to reduce complex constraints to pairwise. Linear constraints are a particularly simple class of constraints and we illustrate how integer programming may be solved as a QUBO.

4.1.1 Posiforms

Posiforms are a convenient modeling representation for a broad class of discrete optimization problems. We introduce posiforms through the satisfiability problem, and then define posiforms more generally.

2-Satisfiability

We have seen how QUBO and Ising are two sides of the same problem. Here we introduce a third common representation of this problem.

Satisfiability was the first NP complete problem identified by *[Cook1971]* and *[Levin1973]*. The conjunctive normal form of SAT is a decision problem which asks for an assignment of Boolean variables which satisfies a collection of constraints called clauses. A clause is a disjunction of literals, and is satisfied if any literal in the clause is true. For example, the clause $x_1 \vee \overline{x}_2$ is satisfied if either $x_1 = true = 1$ or $x_2 = false = 0$. An overline on a Boolean variable indicates its negation, i.e. $\overline{x} = \neg x = 1 - x$. A literal is a variable or a negation of a variable. In a 2-SAT problem all clauses contain at most 2 literals. Finding a satisfiable assignment or proving that there is no satisfying assignment for 2-SAT can be done in polynomial time. However, 3-SAT which allows clauses of up to 3 literals is NP hard and there are problem instances which cannot be solved in polynomial time.

A variant of SAT relaxes the decision problem (is there a satisfying assignment or not?) to an optimization problem and seeks the assignment which minimizes the number of unsatisfiable clauses. This problem is called MAX-SAT (MAXimize the number of satisfied clauses). Weighted MAX-SAT generalizes the problem further. Each clause is assigned a positive weight, and the violation of a clause c having weight w_c incurs a cost w_c. We seek to maximize the weight of satisfied clauses, or equivalently minimize the weight of unsatisfied clauses. Typically, weights are integral so that weighted MAX-SAT is really nothing other than MAX-SAT with redundant clauses (if the weight of a clause is w then w replicated clauses are introduced into the MAX-SAT problem). A weighted MAX-SAT problem is represented as a list of weighted clauses, e.g. $\{(x_1 \vee \overline{x}_2; 3), (x_3; 1), (\overline{x}_3 \vee x_2; 4)\}$ represents a problem of three variables consisting of 3 clauses having weights 3, 1, and 4 respectively. If all clauses contain as most 2 literals then we have a weighted MAX-2-SAT problem. Weighted MAX-2-SAT is equivalent to the QUBO and Ising problems.

How might we represent clauses as terms in a QUBO? A weighted clause $(x_1 \vee \overline{x}_2; 3)$ is unsatisfied if $\overline{x}_1 \wedge x_2$ is true, and this conjunction is equivalent to the product of Boolean variables $\overline{x}_1 x_2$. If the clause is unsatisfied then a cost of 3 is incurred, thus the penalty representing the clause is $3\overline{x}_1 x_2$. The weighted MAX-2-SAT problem $\{(x_1 \vee \overline{x}_2; 3), (x_3; 1), (\overline{x}_3 \vee x_2; 4)\}$ is then easily represented as $3\overline{x}_1 x_2 + \overline{x}_3 + 4x_3\overline{x}_2$. This representation is called a posiform. A posiform is a summation of terms where each term is a product of literals multiplied by a positive (usually integral) weight.

The general weighted MAX-2-SAT problem can be written as the posiform:

$$\min_{\boldsymbol{x}} \sum_c w_c \overline{\ell}_{c,1} \overline{\ell}_{c,2}$$

where $\ell_{c,1}$ and $\ell_{c,2}$ are the two literals of clause c, and w_c is its weight. Interestingly, the resolution rule for satisfiability has been generalized to weighted MAX-SAT which enables the simplification of some weighted MAX-2-SAT problems [Bonet2007].

This posiform representation can be cast as a QUBO by writing negated literals $\overline{x}$ as $1 - x$. For example, the QUBO representation of the three variable problem is obtained by expanding $3(1 - x_1)x_2 + (1 - x_3) + 4(1 - x_1)x_2$. The introduction of negative literals means that QUBOs may be written without any negative coefficients. For example, the QUBO contribution $-3x_1x_2$ may be expressed as either $-3 + 3\overline{x}_2 + 3\overline{x}_1 x_2$ or $-3 + 3\overline{x}_1 + 3x_1\overline{x}_2$. In this form it is clear that the lower bound of this contribution is -3.

General Posiforms

Posiforms are a useful representation for pseudo-Boolean functions (functions which map bit-strings to real values) *[Boros2002]*. General posiforms may have terms consisting of products of more than 2 literals. As an example of a posiform model (in this case having only 2 literals per term), the minimum vertex cover of a graph $G = (V, E)$ is determined by:

$$\min_{\boldsymbol{x}} \left\{ \sum_{v \in V} x_v + M \sum_{(v,v') \in E} \overline{x}_v \overline{x}_{v'} \right\}$$

which determines the smallest set of vertices covering all edges in G. The quadratic penalty of weight M term ensures that for all edges (v, v') at least of one of x_v or $x_{v'}$ is zero. Thus, the minimizer determines the smallest set of vertices which cover all edges in G. A related problem is maximum independent set which seeks the largest possible set of vertices no two of which are connected by an edge (a set of non-connected vertices is called an independent set). Maximum independent set is given by the posiform objective:

$$\min_{\boldsymbol{x}} \left\{ \sum_{v \in V} \overline{x}_v + M \sum_{(v,v') \in E} x_v x_{v'} \right\}$$

The first term seeks the minimal number of terms not in the independent set, and the second term penalizes choices which correspond to non-independent sets.

As an example of a problem having higher-order interactions consider an objective function which counts the number of cliques of size 3 in a graph of N vertices. The graph might be represented with a set of $\binom{N}{2}$ Boolean variables $x_{i,j}$ indicating the presence/absence of each edge (i, j). Given this representation vertices $\{1, 2, 3\}$ form a clique if $x_{1,2}x_{1,3}x_{2,3}$. The total number of 3-cliques is:

$$\sum_{(v_1,v_2,v_3) \in \mathcal{S}} x_{v_1,v_2} x_{v_1,v_3} x_{v_2,v_3}$$

where $\mathcal{S}$ is the set of all subsets of size 3 of $\{1, 2, \cdots, N\}$. This is an example of a posiform where each term has degree 3.

Another higher-order posiform arises in weighted MAX-3-SAT. If clauses consist of 3 literals then the analog of *weightedMax2Sat* is:

$$\min_{\boldsymbol{x}} \sum_c w_c \bar{\ell}_{c,1} \bar{\ell}_{c,2} \bar{\ell}_{c,3}.$$

4.1.2 Reducing Higher-Order Interactions

As we have seen, in many cases the natural model for a problem includes interactions between more than pairs of variables. How can such problems be reduced to the pairwise QUBO representation?

One approach to this reduction-to-pairwise problem involves the introduction of ancillary variables that are minimized over. We begin by noting that the constraint $z \Leftrightarrow x_1 \wedge x_2$ which sets Boolean variable z equal to the logical AND of Boolean variables x_1 and x_2 can be represented through a quadratic penalty function: [1]

$$P(x_1, x_2; z) = x_1 x_2 - 2(x_1 + x_2)z + 3z.$$

This penalty function attains the minimal value of 0 at the 4 (x_1, x_2, z) combinations satisfying $z \Leftrightarrow x_1 \wedge x_2$. All other combination of values have $P > 0$. Thus, P is a penalty function enforcing the constraint. We use this result to reduce a triplet interaction to pairwise interaction as follows: [2]

$$x_1 x_2 x_3 = \min_z \{ z x_3 + M P(x_1, x_2; z) \}$$

where $M > 1$ is a penalty weight. The product $x_1 x_2$ is captured by z which is multiplied by x_3 giving the desired interaction. The penalty function P is added to ensure that z is set correctly given x_1 and x_2. In the same manner an interaction involving 4 or more variables can be reduced to pairwise by sequentially introducing new variables and reducing the degree of the interaction by 1 at each step. For a problem of N terms the required number of ancillary variables in polynomial is N (see *[Boros2002]* for further details). As extra ancillary variables limit the size of problem addressable by the D-Wave hardware it is desirable to minimize the number of introduced variables. The reduction-to-pairwise algorithm of *[Boros2002]* is easily modified to minimize the number ancillary variables. Amongst all terms with more than pairwise interactions we select the pair of variables that is most common amongst the terms. The ancillary variable z is introduced to represent the product of the common pair, and the pair is replaced in all terms with z. The process is repeated if terms of 3 or more variables remain after the replacement. Note that subsequent iterations may involve products of ancillary variables if these ancillary pairs are the most common amongst the terms. Code is available in the utilities package to accomplish this reduction.

4.1.3 Constraints as Penalties

Most applications involve associated constraints on variables which restrict bitstrings to certain feasible configurations. The hardware cannot natively impose constraints, but energy penalties can be introduced which penalize infeasible configurations. Penalty functions are also a source of higher order interactions. In this section we consider approaches to solving optimization problems having constraints.

Constraint programming is a declarative framework for modeling problems which require solutions to satisfy a number of constraints *[Dechter2003]*. Problems posed as finite domain constraint satisfaction problems (CSP) can be straightforwardly translated into QUBO optimization tasks. CSPs typically require very little precision when represented as QUBOs, and consequently are excellent problems for solution on early generations of hardware.

A finite domain CSP consists of a set of variables, a specification of the domain of each variable, and a specification of the constraints over combinations of the allowed values of the variables. A constraint $C_\alpha(\boldsymbol{x}_\alpha)$ defined over a subset of variables $\boldsymbol{x}_\alpha$ defines the set of feasible and infeasible combinations of $\boldsymbol{x}_\alpha$. The constraint C_α may be be viewed as a predicate which evaluates to true on feasible configurations and to false on infeasible configurations. For example,

[1] The same penalty function written in terms of -1/1 variables is $P(s_1, s_2; s_z) = 3 + s_1 s_2 - 2(s_1 + s_2)s_z - s_1 - s_2 + 2s_z$.
[2] The penalty weight may be taken to be $M = 1$.

if the domains of variables X_1, X_2, X_3 are all $\{0, 1, 2\}$, and the constraint is $X_1 + X_2 < X_3$ then the feasible set is $\{(0, 0, 1), (0, 0, 2), (0, 1, 2), (1, 0, 2)\}$, and all remaining combinations are infeasible. The variables involved in a constraint define its scope. Typically, constraint satisfaction problems have many constraints of local scope (involving only a small subset of variables), but there may often be global constraints that couple of variables. We indicate the set of constraints as $\mathcal{C} = \{C_\alpha(\boldsymbol{x}_\alpha)\}_{\alpha=1}^{|\mathcal{C}|}$ and we seek a solution $\boldsymbol{x}$ that satisfies all constraints, i.e. $\bigwedge_{C_\alpha \in \mathcal{C}} C_\alpha(\boldsymbol{x})$ is true. Satisfiability is an example CSP.

Reduction to Maximum Independent Set

There are a number of ways that a CSP may be converted into a QUBO. One standard approach defines the dual problem as follows:

Introduce a variable y_α for each constraint C_α. The domain of y_α is the set of feasible solutions to C_α. Coupling between dual variables arises from constraints C_α and C_β having common variables in their scopes. For example, if the scope of C_α is X_1, X_2, the scope of C_β is X_2, X_3 and the feasible sets are $\{(1, 2), (3, 1)\}$ and $\{(1, 4), (2, 1)\}$ respectively, then $y_\alpha \in \{(1, 2), (3, 1)\}$ and $y_\beta \in \{(1, 4), (2, 1)\}$. The two constraints share variable X_2 so whatever value X_2 is assigned it must be the same value in both constraints. This means that of the 4 possible y_α, y_β combinations only $(1, 2), (2, 1)$ and $(3, 1), (1, 4)$ are allowed because the combinations $(1, 2), (1, 4)$ and $(3, 1), (2, 1)$ disagree on the setting of X_2. This dual constraint formulation defines a conflict graph. The nodes of the conflict graph are the possible feasible settings for each y_α, and we form edges between the nodes y_α, y_β if there is any conflicting setting of a shared variables. An example conflict graph is shown in *dualGraph*.

C_1		C_2			
X_1	X_2	X_2	X_3	Y_1	Y_2
1	2	1	2	1,2	1,4
3	1	2	1	3,1	2,1
3	4			3,4	

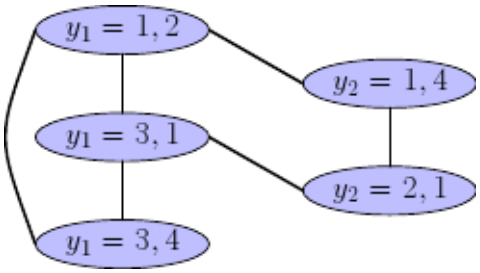

Fig. 4.1: Dual graph of a small CSP consisting of three variables X_1, X_2, and X_3 each having domain $\{1, 2, 3, 4\}$, and two constraints $C_1(X_1, X_2)$, $C_2(X_2, X_3)$ having scopes $\{X_1, X_2\}$ and $\{X_2, X_3\}$ respectively. (a) The feasible sets for C_1 and C_2. (b) The domains of the dual variables y_1 and y_2. (c) The conflict graph defining feasible joint settings for dual variables.

It is not hard to see that if there is an independent set (see *maxIndSet*) in the conflict graph of size $|\mathcal{C}|$ then we have solved the problem. We select each y_α in the independent set which defines values for all the X variables in the scope of C_α. The edge constraint ensures no two elements that are in conflict are in the independent set. As we have seen maximum independent set (MIS) is easily represented as a QUBO. A good discussion of the advantages and disadvantages of modeling in terms of dual variables can be found in [Bacchus2002].

An analogous construction may be used to solve soft constraint satisfaction problems (in other words optimization problems) where the ‘‘constraint” function $C_\alpha(\boldsymbol{x}_\alpha)$ assigns a non-negative weight to each configuration $\boldsymbol{x}_\alpha$. The goal is to find a configuration which minimizes the sum of weights assigned by the ‘‘constraint” functions. In this case the reduction is to weighted MIS where the goal is to find an independent set of least weight. The QUBO representation of weighted MIS for a graph $G = (V, E)$ is simply $\min_{\boldsymbol{x}} \{\sum_v w_v x_v + M \sum_{(v,v') \in E} x_v x_{v'}\}$ where w_v is the weight of vertex v.

Penalty Functions

The CSP → MIS → QUBO conversion can be expensive since the resulting conflict graphs can have great many nodes and edges if each constraint has a large feasible set. Though, there are methods to compress the number of needed nodes, an alternative formulation of the CSP is often preferred. We follow the same approach as was introduced to represent the logical and constraint that was used to reduce higher-order problems to quadratic in *HOSect*. For each constraint $C_\alpha(\boldsymbol{x}_\alpha)$ we introduce a quadratic penalty function $P_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{a}_\alpha)$. The penalty function is defined so that:

$$\begin{aligned} &\min_{\boldsymbol{a}_\alpha} P_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{a}_\alpha) = o \qquad \text{for all feasible } \boldsymbol{x}_\alpha \\ &\min_{\boldsymbol{a}_\alpha} P_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{a}_\alpha) \geq o+1 \qquad \text{for all infeasible } \boldsymbol{x}_\alpha. \end{aligned}$$

Thus, the feasible configurations are encoded as global minima of P_α so $\operatorname{argmin}_{\boldsymbol{x}_\alpha, \boldsymbol{a}_\alpha} P_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{a}_\alpha)$ yields a $\boldsymbol{x}$ for which $C_\alpha(\boldsymbol{x}_\alpha)$ is true. The additional $\boldsymbol{a}_\alpha$ variables are necessary to mimic interactions in C_α that are beyond pairwise. If we have a penalty function for each constraint then minimizing the objective:

$$\min_{\boldsymbol{x},\boldsymbol{a}} P(\boldsymbol{x}, \boldsymbol{a}) = \min_{\boldsymbol{x},\boldsymbol{a}} \left\{ \sum_{\alpha=1}^{|\mathcal{C}|} P_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{a}_\alpha) \right\}$$

gives a feasible solution $\boldsymbol{x}^\star, \boldsymbol{a}^\star$ satisfying all constraints and having objective value $P(\boldsymbol{x}^\star, \boldsymbol{a}^\star) = o$ if the original CSP has a feasible solution $\boldsymbol{x}^\star$. If the original CSP is infeasible then $P(\boldsymbol{x}^\star, \boldsymbol{a}^\star)$ will be at least $o+1$. The utility of this approach depends upon being able to conveniently represent common constraints C_α as QUBO penalties. Fortunately, there are techniques to automatically construct QUBO penalty function from specifications of C_α.

Penalties for Linear Constraints

The simplest constraints are linear equalities and inequalities, and much of the mathematical programming literature is based upon such constraints. These constraints are easily mapped to penalties suitable for QUBO optimization.

m equality constraints on n Boolean variables $\boldsymbol{x}$ may be expressed as $\boldsymbol{C}\boldsymbol{x} = \boldsymbol{c}$ for an $m \times n$ matrix $\boldsymbol{C}$ and $m \times 1$ vector $\boldsymbol{c}$. Since the QUBO allows quadratic interactions, a penalty function for the $\boldsymbol{C}\boldsymbol{x} - \boldsymbol{c} = 0$ constraint is simply $M\|\boldsymbol{C}\boldsymbol{x} - \boldsymbol{c}\|^2$. If we would like to weight each constraint independently we can write a penalty function as:

$$P_=(\boldsymbol{x}) = \sum_k m_k^= \big(\langle \boldsymbol{C}_k, \boldsymbol{x}\rangle - c_k\big)^2$$

where $m_k^= > 0$ is the weight of the k th equality constraint, and $\boldsymbol{C}_k$ is the k th row of $\boldsymbol{C}$. This function is non-negative and equal to zero for feasible $\boldsymbol{x}$. This is clearly quadratic (pairwise) in $\boldsymbol{x}$. Note however, that depending on the structure of $\boldsymbol{C}$ this can couple all the variables in $\boldsymbol{x}$ and potentially create qubit connectivity difficulties.

Inequality constraints of the form $\boldsymbol{D}\boldsymbol{x} \leq \boldsymbol{d}$ are slightly more difficult but can be represented by introducing slack variables to reduce the inequalities to equalities. Consider the i th inequality constraint $\langle \boldsymbol{D}_i, \boldsymbol{x}\rangle - d_i \leq 0$ where $\boldsymbol{D}_i$ is the i th row of $\boldsymbol{D}$. Introducing a non-negative slack variable $\xi_i \geq 0$ the inequality is written as the equality $\langle \boldsymbol{D}_i, \boldsymbol{x}\rangle - d_i + \xi_i = 0$. The slack variable may need to take a value as large as $d_i - \min_{\boldsymbol{x}} \langle \boldsymbol{D}_i, \boldsymbol{x}\rangle = d_i - \sum_j \min(D_{i,j}, 0)$. This slack quantity may be represented with $\lceil d_i - \sum_j \min(D_{i,j}, 0)\rceil$ bits. We write $\xi_i = \langle \boldsymbol{2}, \boldsymbol{a}_i\rangle$ where $\boldsymbol{a}_i$ is the binary representation of ξ_i, and $\boldsymbol{2}$ is a vector whose elements are the powers of 2. This binary representation enforces the requirement that $\xi_i \geq 0$. The i th inequality may be enforced with the penalty $\big(\langle \boldsymbol{D}_i, \boldsymbol{x}\rangle - d_i + \langle \boldsymbol{2}, \boldsymbol{a}_i\rangle\big)^2$ and all constraints can be represented as:

$$P_\leq(\boldsymbol{x}, \boldsymbol{a}) = \sum_k m_k^\leq \big(\langle \boldsymbol{D}_i, \boldsymbol{x}\rangle - d_i + \langle \boldsymbol{2}, \boldsymbol{a}_i\rangle\big)^2$$

where $\boldsymbol{a}$ is the vector collecting all binary slack variables. Note again that this penalty can couple all variables $\boldsymbol{x}$ and ancillary variables $\boldsymbol{a}$.

Penalties for Nonlinear Constraints

We have seen how penalty functions for equality and inequality constraints may be constructed. Inequality constraints require the introduction of ancillary variables to represent the slacks of the inequalities. Further we have noted that these penalties may couple a great many variables. More generally, we ask: how do we develop constraints for nonlinear constraints? How do we generate constraints that have specified variable connectivity?

A few other constraints useful in building Boolean circuits are given in *logicalTable*. $\oplus$ is "exclusive-or" and requires the introduction of a single ancillary variable a. Where did these come from? In this section we outline an algorithm to derive quadratic penalties from specifications of feasible sets, and show how connectivity constraints between variables may be accommodated.

Table 4.1: QUBO penalty functions for capturing the output of simple Boolean operations

Constraint	Penalty
$z \Leftrightarrow \neg x$	$2xz - x - z + 1$
$z \Leftrightarrow x_1 \vee x_2$	$x_1x_2 + (x_1 + x_2)(1 - 2z) + z$
$z \Leftrightarrow x_1 \wedge x_2$	$x_1x_2 - 2(x_1 + x_2)z + 3z$
$z \Leftrightarrow x_1 \oplus x_2$	$2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a$

For penalties whose scope includes a small number of variables (e.g. less than 15) we may construct penalty functions from a specification of the feasible and infeasible configurations. Let F represent a set of feasible configurations, and $\overline{F}$ represent a set of infeasible configurations. We require $\min_{\boldsymbol{a}} P(\boldsymbol{x}, \boldsymbol{a}) = o$ for $\mathbf{x} \in F$ and $\min_{\boldsymbol{a}} P(\mathbf{x}, \boldsymbol{a}) \geq o + 1$ for $\mathbf{x} \in \overline{F}$ for some constant o. Since P must be quadratic, we write the penalty as:

$$P(\boldsymbol{x}, \boldsymbol{a}) = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{a} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}^{x,x} & \boldsymbol{Q}^{x,a} \\ 0 & \boldsymbol{Q}^{a,a} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{a} \end{bmatrix}$$

with

$$\boldsymbol{Q}^{x,x} = \sum_i w_i^{x,x} \boldsymbol{M}_i^{x,x}, \qquad \boldsymbol{Q}^{x,a} = \sum_i w_i^{x,a} \boldsymbol{M}_i^{x,a}, \qquad \boldsymbol{Q}^{a,a} = \sum_i w_i^{a,a} \boldsymbol{M}_i^{a,a}.$$

We seek the parameters $\boldsymbol{w} \equiv \{\boldsymbol{w}^{x,x}, \boldsymbol{w}^{x,a}, \boldsymbol{w}^{a,a}\}$ given the matrices $\boldsymbol{M} \equiv \{\boldsymbol{M}^{x,x}, \boldsymbol{M}^{x,a}, \boldsymbol{M}^{a,a}\}$. We generalize penalties in this way in order to limit required precision and/or to accommodate connectivity constraints as described by the set of $\boldsymbol{M}$. Thus, we'd like to solve the optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w}, o} \|\boldsymbol{w}\|_1 \\ \text{subject to: } &\langle \boldsymbol{w}^{x,x}, \boldsymbol{c}^{x,x}(j) \rangle + \min_k \{ \langle \boldsymbol{w}^{x,a}, \boldsymbol{c}_k^{x,a}(j) \rangle + \langle \boldsymbol{w}^{a,a}, \boldsymbol{c}_k^{a,a} \rangle \} = o \quad \forall j \in F \\ &\langle \boldsymbol{w}^{x,x}, \boldsymbol{c}^{x,x}(\overline{j}) \rangle + \min_k \left\{ \langle \boldsymbol{w}^{x,a}, \boldsymbol{c}_k^{x,a}(\overline{j}) \rangle + \langle \boldsymbol{w}^{a,a}, \boldsymbol{c}_k^{a,a} \rangle \right\} \geq o + 1 \quad \forall \overline{j} \in \overline{F} \end{aligned}$$

where k ranges over the allowed configurations $\boldsymbol{a}_k$ of $\boldsymbol{a}$, and the $\boldsymbol{c}$ vectors have components indexed by j given by:

$$\left[\boldsymbol{c}^{x,x}(j)\right]_i = \boldsymbol{x}_j^\top \boldsymbol{M}_i^{x,x} \boldsymbol{x}_j, \qquad \left[\boldsymbol{c}_k^{x,a}(j)\right]_i = \boldsymbol{x}_j^\top \boldsymbol{M}_i^{x,a} \boldsymbol{a}_k, \qquad \left[\boldsymbol{c}_k^{a,a}\right]_i = \boldsymbol{a}_k^\top \boldsymbol{M}_i^{a,a} \boldsymbol{a}_k.$$

We have also introduced an unknown o representing the objective value of feasible configurations. This unknown value need not be equal to 0. The notation $j \in F$ and $\overline{j} \in \overline{F}$ indicates feasible and infeasible configurations respectively. Minimizing the L_1 norm forces penalty terms to be close to 0, and aides in limiting the required precision. The second constraint can be expressed more simply to give:

$$\begin{aligned} &\min_{\boldsymbol{w}, o} \|\boldsymbol{w}\|_1 \\ \text{subject to: } &\langle \boldsymbol{w}^{x,x}, \boldsymbol{c}^{x,x}(j) \rangle + \min_k \{ \langle \boldsymbol{w}^{x,a}, \boldsymbol{c}_k^{x,a}(j) \rangle + \langle \boldsymbol{w}^{a,a}, \boldsymbol{c}_k^{a,a} \rangle \} = o \quad \forall j \in F \\ &\langle \boldsymbol{w}^{x,x}, \boldsymbol{c}^{x,x}(\overline{j}) \rangle + \langle \boldsymbol{w}^{x,a}, \boldsymbol{c}_k^{x,a}(\overline{j}) \rangle + \langle \boldsymbol{w}^{a,a}, \boldsymbol{c}_k^{a,a} \rangle \geq o + 1 \quad \forall \overline{j} \in \overline{F}, \forall k. \end{aligned}$$

The min in the first constraint makes this a difficult optimization problem. One approach to its solution is to express it as a mixed integer program and use a mixed integer solver.

We introduce Boolean indicator variables $\{\alpha_k(j)\}$ for each $1 \le j \le |F|$ and for each allowed configuration $\boldsymbol{a}_k$. Define the vector $\boldsymbol{\alpha}(j)$ whose components are the different k. The indicator variable $\alpha_k(j)$ picks out the single k at which $\min_k$ in constraint 1 is attained (if there are multiple k attaining the minimal value we pick 1 of these). Thus we have:

$$\langle \mathbf{1}, \boldsymbol{\alpha}(j) \rangle = 1 \quad \forall j \in F$$
$$\sum_{k'} \left\{ \langle \alpha_{k'}(j) \boldsymbol{w}^{x,a}, \boldsymbol{c}_{k'}^{x,a}(j) \rangle + \langle \alpha_{k'}(j) \boldsymbol{w}^{a,a}, \boldsymbol{c}_{k'}^{a,a} \rangle \right\} \le \langle \boldsymbol{w}^{x,a}, \boldsymbol{c}_k^{x,a}(j) \rangle + \langle \boldsymbol{w}^{a,a}, \boldsymbol{c}_k^{a,a} \rangle \quad \forall k, j.$$

Unfortunately, this constraint couples $\alpha_k(j)$ to $\boldsymbol{w}^{x,a}$ and $\boldsymbol{w}^{a,a}$. [3] To break this coupling and obtain a linear problem we introduce $\boldsymbol{v}_k^{x,a}(j) = \alpha_k(j)\boldsymbol{w}^{x,a}$ and $\boldsymbol{v}_k^{a,a}(j) = \alpha_k(j)\boldsymbol{w}^{a,a}$. This requirement is enforced with:

$$\begin{aligned}
\boldsymbol{v}_k^{x,a}(j) &\le \alpha_k(j) \\
\boldsymbol{v}_k^{a,a}(j) &\le \alpha_k(j) M \\
-\boldsymbol{v}_k^{x,a}(j) &\le \alpha_k(j) M \\
-\boldsymbol{v}_k^{a,a}(j) &\le \alpha_k(j) M \\
\sum_k \boldsymbol{v}_k^{x,a}(j) &= \boldsymbol{w}^{x,a} \\
\sum_k \boldsymbol{v}_k^{a,a}(j) &= \boldsymbol{w}^{a,a}.
\end{aligned}$$

The first two of the constraints in each column requires that $\boldsymbol{v}_k^{x,a}(j) = 0$ and $\boldsymbol{v}_k^{a,a}(j) = 0$ if $\alpha_k(j) = 0$. The penalty weight M should be chosen to be larger than the largest absolute value of the optimal values of $\boldsymbol{w}^{x,a}$ and $\boldsymbol{w}^{a,a}$ so that we do not eliminate any solutions.

The resulting mixed integer program can then be solved with any MIP solver. If there is no feasible solution for a given number of ancillary variables or specified connectivity, then these requirements must be relaxed by introducing additional ancillary variables of additional connectivity.

On request we can supply MATLAB code which builds penalty functions using this approach, but the code requires a good mixed integer program solver.

Even if a penalty can be derived through conversion to a linear equality or inequality, it may still be useful to apply the above penalty-deriving machinery to construct a penalty with fewer ancillary variables. For example, the constraint $x_1 \vee x_2 \vee x_3$ (which is useful if we're solving 3-SAT) can always be expressed as the inequality $x_1 + x_2 + x_3 \ge 1$, and converted to the penalty function $(x_1 + x_2 + x_3 - 1 - a_1 - 2a_2)^2$. The two ancillary variables a_1 and a_2 are necessary to represent the possible slack values 0, 1, and 2. However, the following penalty function

$$-3 + (\overline{x}_1 + \overline{x}_2 + \overline{x}_3)(1 + a) + \overline{a} + x_1 x_2 + x_2 x_3 + x_3 x_1$$

represents the same constraint using a single ancillary variable. [4]

4.1.4 Integer Programming

As an example application of the QUBO idioms developed above, we show how any integer program may be converted to QUBO form.

Consider the integer program:

$$\min_{\boldsymbol{y}} \langle \boldsymbol{c}, \boldsymbol{y} \rangle \qquad \text{subject to:} \quad \boldsymbol{A}\boldsymbol{y} = \boldsymbol{a}, \quad \boldsymbol{B}\boldsymbol{y} \le \boldsymbol{b}, \quad y_i \in \{0, D_i - 1\}$$

[3] CPLEX and other commercial grade optimization software can solve problems with quadratic constraints so it may be used to directly address this formulation, but we have not tried that.

[4] In fact, it is this representation which is commonly used to reduce 3-SAT to MAX-2-SAT.

where D_i is the number of allowed values for variable y_i. As a first step in translating this into a QUBO, the non-binary valued y_i must be made binary valued. This can be accomplished either by introducing indicator variables or by writing y_i in binary. We write y_i in binary to limit the number of new Boolean valued variables required. Let $d_i = \lceil \log_2 D_i \rceil$ and $y_i = \langle \mathbf{2}, \boldsymbol{x}_i \rangle$ where $\mathbf{2} = [2^{d_i}, \cdots, 2, 1]$ is the vector of powers of two and $\boldsymbol{x}_i = [x_{i,d_i}, \cdots, x_{i,1}, x_{i,0}]$ represents the bits of y_i. The binary representation may allow for y_i values larger than D_i so we further impose the inequality constraints that $\langle \mathbf{2}, \boldsymbol{x}_i \rangle \leq D_i - 1$. Thus, the original integer program may easily be converted to:

$$\min_{\boldsymbol{x}} \langle \boldsymbol{c}, \boldsymbol{x} \rangle \qquad \text{subject to:} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{a}, \quad \boldsymbol{B}\boldsymbol{x} \leq \boldsymbol{b}, \quad x_i \in \{0, 1\}$$

where the new $\boldsymbol{B}$ and $\boldsymbol{b}$ have the additional inequality constraints appended, and where $\boldsymbol{c}$ has been suitably modified. In this form, the only remaining difficulty is in representing the equality and inequality constraints within the unconstrained QUBO. As we have seen, these are easily translated to quadratic penalties.

Thus, the binary program given by *binaryProgramme* can be written as the QUBO objective:

$$\underset{\boldsymbol{x},\boldsymbol{\xi}}{\operatorname{argmin}} \left\{ \langle \boldsymbol{c}, \boldsymbol{x} \rangle + \sum_k m_k^{=} \big(\langle \boldsymbol{A}_k, \boldsymbol{x} \rangle - a_k \big)^2 + \sum_i m_i^{\leq} \big(\langle \boldsymbol{B}_i, \boldsymbol{x} \rangle - b_i + \langle \mathbf{2}, \boldsymbol{\xi}_i \rangle \big)^2 \right\}$$

where k runs over the equality constraints and i runs over the inequality constraints. The only remaining question is how to set the sizes of the penalty weights $\boldsymbol{m}^{=}$ and $\boldsymbol{m}^{\leq}$. Without problem specific knowledge, we approach this issue the same way it is done in Lagrangian methods by incrementally increasing the weights until the constraints are satisfied. If $\boldsymbol{m} = [\boldsymbol{m}^{=}, \boldsymbol{m}^{\leq}]$ is some setting of the penalty weights and $\boldsymbol{x}(\boldsymbol{m}), \boldsymbol{\xi}(\boldsymbol{m})$ are the optimal variable settings for that choice of $\boldsymbol{m}$, then we update the penalty weights in proportion to the constraint violations:

$$m_k^{=} := m_k^{=} + \alpha \big| \langle \boldsymbol{A}_k, \boldsymbol{x}(\boldsymbol{m}) \rangle - a_k \big| m_k^{\leq} := m_k^{\leq} + \alpha \big| \langle \boldsymbol{B}_k, \boldsymbol{x}(\boldsymbol{m}) \rangle - b_k + \langle \mathbf{2}, \boldsymbol{\xi}(\boldsymbol{m}) \rangle \big|.$$

$\alpha > 0$ determines the rate at which penalty weights are accumulated.

4.1.5 Constraint Satisfaction as Inverse Verification

There is another approach to solving CSPs that exploits the fact that NP hard decision problems can be verified in polynomial time. Recall that we seek a feasible solution for which $C(\boldsymbol{x}) \equiv \bigwedge_{\alpha=1}^{|\mathcal{C}|} C_\alpha(\boldsymbol{x}_\alpha)$ is satisfied. If the CSP corresponding to $C(\boldsymbol{x})$ is in NP then we can write the output $z \Leftrightarrow C(\boldsymbol{x})$ as a Boolean circuit whose size is polynomial in the number of Boolean input variables. The Boolean circuit $z \Leftrightarrow C(\boldsymbol{x})$ can be converted to an optimization objective using the logical primitives defined in *logicalTable* (each gate just contributes the corresponding penalty function). This resultant optimization objective will also have a polynomial number of variables, and at a feasible solution $\boldsymbol{x}^\star$, $z_\star = 1$ the optimization objective will evaluate to 0. The advantage of the optimization form of the verification circuit is that the circuit may be run in reverse inferring inputs from the desired true output. To accomplish this, clamp the output of the circuit to $z = 1$ by adding $-Mz$ (for sufficiently large M) to the objective, and then minimize with respect to all variables. [5] If we obtain a solution for which the objective evaluates to 0 then we have found the feasible solution. In *factoringSect* we show how this may be used to solve factoring problems as QUBOs, and that this formulation yields low requirements on both connectivity and precision constraints making it suitable for quantum annealing hardware.

Factoring

We wish to factor an ℓ bit integer p as a product of a pair of integers a and b. p, a, and b are all represented in binary, i.e. $p = \langle \mathbf{2}, \boldsymbol{p} \rangle$, $a = \langle \mathbf{2}, \boldsymbol{a} \rangle$, and $b = \langle \mathbf{2}, \boldsymbol{b} \rangle$ where $\mathbf{2}$ is a vector of the powers of two of appropriate length. Factors a and b are assumed to be ℓ_a and ℓ_b bits respectively so that $\ell = \ell_a + \ell_b$ or $\ell = \ell_a + \ell_b - 1$.

A natural first step is to cast this task as an optimization problem. A reasonable objective is to minimize the difference:

$$\underset{\boldsymbol{a},\boldsymbol{b}}{\operatorname{argmin}} \big(p - \langle \boldsymbol{a}, \mathbf{2} \rangle \langle \mathbf{2}, \boldsymbol{b} \rangle \big)^2.$$

[5] Alternatively, we may substitute $z = 1$ into the objective and eliminate z from the problem.

This problem involves quartic interactions in the optimization variables so ancillary variables will need to be introduced to reduce the objective to quadratic. More problematically, there are a large number of interactions between pairs of variables and the magnitudes of these interactions varies from 1 to $2^{2\ell_a+2\ell_b-4}$. Thus, both connectivity and precision requirements are severe in this formulation of the problem.

To ameliorate these difficulties, we adapt the ‘‘inverse verification” idea to this factoring context. Representing factoring as a constraint satisfaction problem greatly reduces precision requirements, and Boolean circuits enable us to tuneably define the connectivity between optimization variables.

Multiplication Circuits as QUBOs

For factorization the verification circuit is formed from the circuit which multiplies the inputs $\boldsymbol{a}$ and $\boldsymbol{b}$. We then fix the output of the circuit to the known bit string $\boldsymbol{p}$, and minimize the energy of the penalty formulation of the circuit to find $\boldsymbol{a}$ and $\boldsymbol{b}$. The multiplication circuit is built from AND gates, half adders and full adders. The AND gate which realizes $t = x_1 \wedge x_2$ is given by the penalty $P_\wedge(x_1, x_2; t) = x_1x_2 - 2(x_1 + x_2)r + 3t$. The half adder takes two input bits x_1 and x_2 and gives two output bits s and c representing the sum and carry of the two bits. These are defined through $s+2c = x_1+x_2$. Clearly, then the penalty function for this equality constraint is $P_{2A}(x_1, x_2; s, c) = (s+2c-x_1-x_2)^2$. Similarly, the full adder adds three inputs bits and returns the sum and carry as $s + 2c = x_1 + x_2 + x_3$. The full adder penalty function is $P_{3A}(x_1, x_2, x_3; s, c) = (s + 2c = x_1 + x_2 + x_3)^2$. The penalty functions result in fully connected groups of 3, 4, and 5 variables respectively. Moreover, the range of coupling coefficients is small. The full multiplication circuit will be built from repeated use of these simple elements, and total precision requirements will remain small.

The multiplication circuit parallels standard multiplication circuits with logical gates replaced by the corresponding penalty functions. A simple circuit architecture is inspired by the explicit multiplication table. Consider the multiplication of two 4 bit integers. Schematically, the product is formed from the multiplication table given in *multTable*.

Table 4.2: 4-bit integer multiplication table

				a_3	a_2	a_1	a_0
				b_3	b_2	b_1	b_0
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Multiplication of two 4-bit integers (a_3, a_2, a_1, a_0) and (b_3, b_2, b_1, b_0) form the 8-bit product $(p_7, p_6, p_5, p_4, p_3, p_2, p_1, p_0)$.

Each of the products a_ib_j is the logical AND of the two bits. We introduce the ancillary variables $t_{i,j} \equiv a_i \wedge b_j$, s_b^l representing partial sums contributing to the b th output bit at level l, and c_b^l representing carries from adding contributions to the b th output bit at level l.

One circuit representation for the multiplication is formed from adding up the contributions to each output bit p_b as in *multTable*. As an example, the following QUBO objective realizes the multiplication of two 4-bit integers a and b into

their product p.

bit p_0: $P_\wedge(a_0, b_0; p_0)$

bit p_1: $P_\wedge(a_1, b_0; t_{1,0}) + P_\wedge(a_0, b_1; t_{0,1}) + P_{2A}(t_{1,0}, t_{0,1}; p_1, c_1^1)$

bit p_2: $P_\wedge(a_0, b_2; t_{0,2}) + P_\wedge(a_1, b_1; t_{1,1}) + P_\wedge(a_2, b_0; t_{2,0}) + P_{2A}(t_{2,0}, t_{1,1}; s_2^1, c_2^1) + P_{3A}(t_{0,2}, s_2^1, c_1^1; p_2, c_2^2)$

bit p_3: $P_\wedge(a_0, b_3; t_{0,3}) + P_\wedge(a_1, b_2; t_{1,2}) + P_\wedge(a_2, b_1; t_{2,1}) + P_\wedge(a_3, b_0; t_{3,0}) + P_{2A}(t_{2,1}, t_{3,0}; s_3^1, c_3^1) +$
$P_{3A}(t_{1,2}, s_3^1, c_2^1; s_3^2, c_3^2) + P_{3A}f(t_{0,3}, s_3^2, c_2^2; p_3, c_3^3)$

bit p_4: $P_\wedge(a_1, b_3; t_{1,3}) + P_\wedge(a_2, b_2; t_{2,2}) + P_\wedge(a_3, b_1; t_{3,1}) + P_{2A}(t_{2,2}, t_{3,1}; s_4^1, c_4^1) +$
$P_{3A}(t_{1,3}, s_4^1, c_3^1; s_4^2, c_4^2) + P_{3A}(s_4^2, c_3^2, c_3^3; p_4, c_4^3)$

bit p_5: $P_\wedge(a_2, b_3; t_{2,3}) + P_\wedge(a_3, b_2; t_{3,2}) + P_{3A}(t_{2,3}, t_{3,2}, c_4^1; s_5^2; c_5^2) + P_{3A}(s_5^2, c_4^2, c_4^3; p_5, c_5^3)$

bit p_6: $P_\wedge(a_3, b_3; t_{3,3}) + P_{3A}(t_{3,3}, c_5^2, c_5^3; p_6, p_7)$

bit p_7: the carry from the p_6 sum

This QUBO objective has energy 0 if and only if the binary encodings of a and b satisfy $p = ab$.

The graphical representation of these QUBO contributions is perhaps simpler, and is shown in *multFig*.

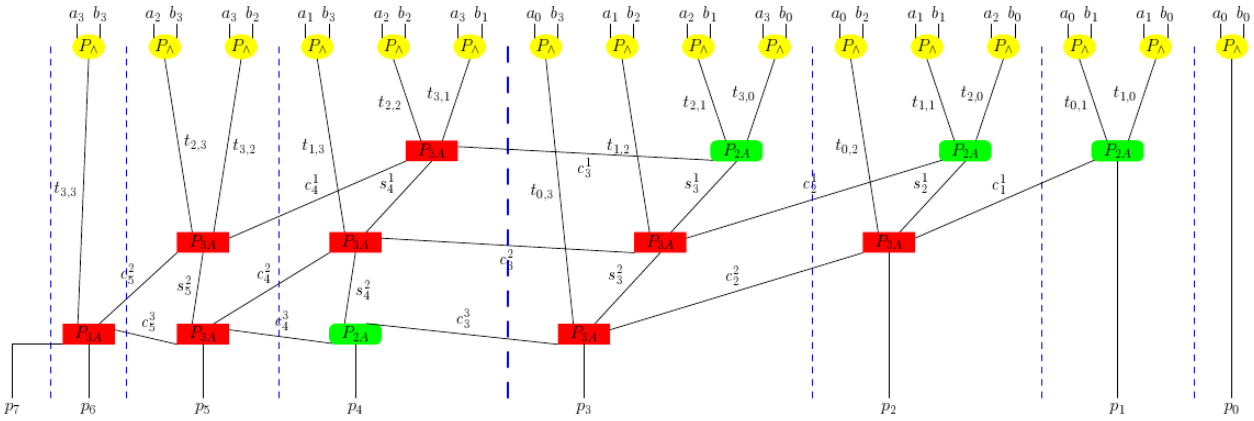

Fig. 4.2: Graphical representation of the multiplication circuit for two 4 bit integers. The structure reflects *multTable*. See text for the explicit QUBO form of the circuit.

Yellow ellipses represent $\wedge$ penalties, green rounded boxes represent half-adder penalties, and red boxes represent full-adder penalties. Variables are represented by edges and connect appropriate penalty terms. The contributions to each bit are separated by a blue dashed line. The thick dashed blue line separating output bits p_3 and p_4 separates parts of the circuit having different connectivity rules.

This circuit can be run in ‘‘reverse” by fixing the output bits $(p_7, \cdots, p_0)$ to the values defined by the number to be factored, and then optimizing over the remaining $\boldsymbol{a}$ and $\boldsymbol{b}$ variables as well as the intermediate sum and carry variables. As is visible from *multFig* the connectivity between variables is much sparser than the naive $(p - \langle \boldsymbol{a}, \boldsymbol{2}\rangle\langle \boldsymbol{2}, \boldsymbol{b}\rangle)^2$ objective. Connectivity is defined by the sparse connections between small completely connected subgraphs of size 3, 4, and 5.

A Regularly Structured Factoring Lattice

To scale to large integers we need a regular circuit structure. One way to do this is to embed the ANDing of variables directly within the adder gates. Consequently, we define the new adder penalties $\tilde{P}_{2A}(x_1^1, x_1^2, x_2; s, c)$ which realizes the constraint $s + 2c = x_1^1 x_1^2 + x_2$ and $\tilde{P}_{3A}(x_1^1, x_1^2, x_2, x_3)$ which realizes $s + 2c = x_1^1 x_1^2 + x_2 + x_3$. By running the

method described in *derivingSect*, we rapidly find that the following penalties realize these constraints:

$$\tilde{P}_{2A}(x_1^1, x_1^2, x_2; s, c) = \begin{bmatrix} x_1^1 & x_1^2 & x_3 & s & c \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & -2 & -4 \\ 0 & 0 & 2 & -2 & -4 \\ 0 & 0 & 1 & -4 & -6 \\ 0 & 0 & 0 & 3 & 6 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1^1 \\ x_1^2 \\ x_2 \\ s \\ c \end{bmatrix}$$

$$\tilde{P}_{3A}(x_1^1, x_1^2, x_2, x_3; s, c) = \begin{bmatrix} x_1^1 & x_1^2 & x_2 & x_3 & s & c \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 2 & -2 & -4 \\ 0 & 0 & 2 & 2 & -2 & -4 \\ 0 & 0 & 1 & 4 & -4 & -8 \\ 0 & 0 & 0 & 1 & -4 & -8 \\ 0 & 0 & 0 & 0 & 3 & 8 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \begin{bmatrix} x_1^1 \\ x_1^2 \\ x_2 \\ x_3 \\ s \\ c \end{bmatrix}.$$

Note that no additional ancillary variables are needed to represent these non-linear constraints. Thus, we can eliminate all the $t_{i,j}$ variables representing the $\wedge$ s of $a_i b_j$. The price paid for this variable reduction is additional connectivity. The $\tilde{P}_{2A}$ and $\tilde{P}_{3A}$ gates are now connected graphs of size 5 and 6.

A regularly structured constraint/optimization network representing the multiplication of two 8-bit numbers using the new gates is shown in *mult8Circuit*.

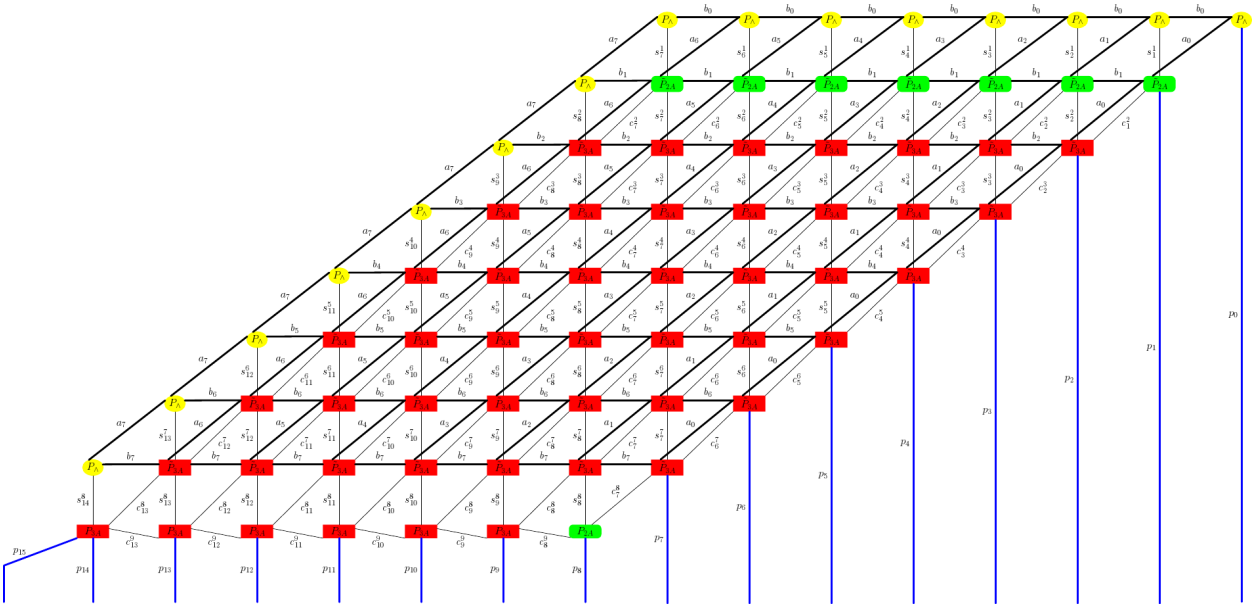

Fig. 4.3: The constraint network representing the multiplication of two 8 bit numbers.

The dark black lines represent the desired $\boldsymbol{a}$ and $\boldsymbol{b}$ variables which thread through the lattice. From the product of two ℓ bit integers into an integer of at most 2ℓ bits, the circuit has $\mathcal{O}(\ell^2)$ variables and the maximal connectivity of any variable is $\mathcal{O}(\ell)$ (this maximal connectivity occurs for the $\boldsymbol{a}$ and $\boldsymbol{b}$ variables). Again the $\boldsymbol{p}$ variables will be fixed to the bits of the number to be factored.

Further, circuit improvements are possible but this example shows a complete example of solving a problem with QUBOs mindful of the constraints imposed by the underlying QUBO-solving hardware. As we have done here, it is generally true that in any given problem, precision, connectivity, and number of qubits may all be traded off against each other.

This discussion while providing an efficient QUBO representation for factoring says nothing about solving the resulting QUBO problem. Factoring has been used as a generator of difficult optimization problems [Burges2002], and there are expected to be a great many local minima in the final QUBO.

CHAPTER

FIVE

BIBLIOGRAPHY

BIBLIOGRAPHY

[Ambainis2004] An Elementary Proof of the Quantum Adiabatic Theorem, Andris Ambainis and Oded Regev, 2004. Available at http://arxiv.org/PS_cache/quant-ph/pdf/0411/0411152v2.pdf.

[Barahona1982] F. Barahona, On the Computational Complexity of Ising Spin Glass Models, *J. Phys. A*, 15, 3241-3253, 1982.

[Bertsimas1999] D. Bertsimas and C.-P. Teo and R. Vohra, On dependent randomized rounding algorithms}, *Oper. Res. Lett.*, 24(3), 105-114, May 1996. Available at http://www.mit.edu/~dbertsim/papers/ApproximationAlgorithms/On dependent randomized rounding algorithms.pdf.

[Boros2002] E. Boros and P. L. Hammer}, Pseudo-boolean optimization, *Discrete Appl. Math.*, 123, 155-225 2002. Available at http://rutcor.rutgers.edu/~boros/Papers/2002-DAM-BH.pdf.

[Burges2002] C.J.C. Burges}, Factoring as Optimization, Microsoft Research TR-2002-83, 2002. Available at http://research.microsoft.com/en-us/um/people/cburges/tech_reports/tr-2002-83.pdf.

[Boyd2007] Stephen Boyd and Almir Mutapcic, Subgradient Methods. Available at http://www.stanford.edu/class/ee364b/lectures/subgrad_method_notes.pdf.

[Cook1971] S. Cook, The Complexity of Theorem-Proving Procedures, *Proceedings of 3rd annual ACM Symposium on Theory of Computing*, 151-158, 1971,

[Coughlan2009] James Coughlan, A tutorial introduction to belief propagation}, 2009. Available at http://www.ski.org/Rehab/Coughlan_lab/General/TutorialsandReference/BPtutorial.pdf.

[Dechter1987] R. Dechter and J. Pearl, The cycle-cutset method for improving search performance in AI applications, *Proceedings of the Third IEEE on Artificial Intelligence Applications*, 224-230, 1987.

[Dechter2003] Rina Dechter, Constraint Processing, Morgan Kaufmann, 2003.

[Farhi2000] Edward Farhi and Jeffrey Goldstone and Sam Gutmann and Michael Sipser}, Quantum Computation by Adiabatic Evolution, 2000. Available at http://arxiv.org/abs/quant-ph/0001106.

[Farhi2002] Edward Farhi and Jeffrey Goldstone and Sam Gutmann, Quantum Adiabatic Evolution Algorithms versus Simulated Annealing, 2002. Available at http://arxiv.org/abs/quant-ph/0201031.

[Grover1996] Lov K. Grover, A fast quantum mechanical algorithm for database search, *Proceedings, 28th Annual ACM Symposium on the Theory of Computing*, p212, May 1996. Pedagogical review available at http://arxiv.org/PS_cache/quant-ph/pdf/0109/0109116v1.pdf.

[Harris2010] R. Harris et al., Experimental Investigation of an Eight Qubit Unit Cell in a Superconducting Optimization Processor, *Phys. Rev. B* 82, 024511 2010. Available at http://arxiv.org/abs/1006.0028.

[Johnson2007] 10. (a) Johnson et al. Lagrangian relaxation for MAP estimation in graphical models. *45th Annual Allerton Conference on Communication, Control and Computing*. September 2007. Available at http://ssg.mit.edu/~jasonj/jmw-allerton07.pdf.

[Johnson2011] M. W. Johnson et al. Quantum annealing with manufactured spins *Nature* 473, 194-198, May 12 2011.

[Kadowaki1998] Tadashi Kadowaki and Hidetoshi Nishimori, Quantum Annealing in the Transverse Ising Model, *Phys. Rev. E* 58, 5355-5363, 1998. Available at http://arxiv.org/PS_cache/cond-mat/pdf/9804/9804280v1.pdf.

[Kirkpatrick1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by Simulated Annealing, *Science*, 671-680, 1983. Available at http://home.gwu.edu/~stroud/classics/KirkpatrickGelattVecchi83.pdf.

[Kochenberger2004] G. Kochenberger, F. Glover, B. Alidaee, and C. Rego, A Unified Modeling and Solution Framework for Combinatorial Optimization Problems, *OR Spectrum*, 2004, 26, 237-250. Available at http://leeds-faculty.colorado.edu/glover/fred%20pubs/333%20-%20xQx%20-%20Unified%20modeling%20and%20solution%20framework%20%28short%29.doc.

[Kolmogorov2004] [V. Kolmogorov and R. Zabih, What energy functions can be minimized via graph cuts?, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26, 65-81, 2004. Available at http://www.cs.cornell.edu/rdz/Papers/KZ-ECCV02-graphcuts.pdf.

[Levin1973] Leonid Levin, Universal search problems, 1973. Translated into english by B. A. Trakhtenbrot, A survey of Russian approaches to perebor (brute-force searches) algorithms, *Annals of the History of Computing* 6(4), 384-400, 1984.

[Liu2005] Wei Liu and Dawn Wilkins and Bahram Alidaee}, A Hybrid Multi-Exchange Local Search for Unconstrained Binary Quadratic Program, University of Mississippi, Hearin Center for Enterprise Science, HCES-09-05, 2005. Available at http://hces.bus.olemiss.edu/reports/hces-09-05.pdf.

[Marinescu2007] R. Marinescu and R. Dechter, Best-First AND/OR Search for 0-1 Integer Linear Programming, *Proceedings of the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, 2007. Available at http://www.ics.uci.edu/~radum/papers/cpaior07-aobf.pdf.

[Mizel2007] A. Mizel, D. A. Lidar, and M. Mitchell Simple proof of equivalence between adiabaticquantum computation and the circuit model, *Physical Review Letters*, 2007, vol 99, 070502. Available at http://www.arimizel.com/images/PRL070502.pdf.

[Roland2002] Jeremie Roland and Nicolas J. Cerf, Quantum search by local adiabatic evolution, *Phys. Rev. A*, 65, 042308, 2002. Available at http://arxiv.org/PS_cache/quant-ph/pdf/0107/0107015v1.pdf.

[Reichardt2004] Ben Reichardt, The quantum adiabatic optimization algorithm and local minima, *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004.

[Schraudolph2009] Nicol N. Schraudolph and Dmitry Kamenetsky}, Efficient exact inference in planar Ising models, *Advances in Neural Information Processing Systems 21*, MIT Press, 2009. Available at http://users.rsise.anu.edu.au/~dkamen/nips08.pdf.