Overview

This document provides guidance on programming the D-Wave system, including how to formulate problems, embed, and configure parameters.
Notice and Disclaimer

D-Wave Systems Inc. (D-Wave), its subsidiaries and affiliates, makes commercially reasonable efforts to ensure that the information in this document is accurate and up to date, but errors may occur. NONE OF D-WAVE SYSTEMS INC., its subsidiaries and affiliates, OR ANY OF ITS RESPECTIVE DIRECTORS, EMPLOYEES, AGENTS, OR OTHER REPRESENTATIVES WILL BE LIABLE FOR DAMAGES, CLAIMS, EXPENSES OR OTHER COSTS (INCLUDING WITHOUT LIMITATION LEGAL FEES) ARISING OUT OF OR IN CONNECTION WITH THE USE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED OR REFERRED TO IN IT. THIS IS A COMPREHENSIVE LIMITATION OF LIABILITY THAT APPLIES TO ALL DAMAGES OF ANY KIND, INCLUDING (WITHOUT LIMITATION) COMPENSATORY, DIRECT, INDIRECT, EXEMPLARY, PUNITIVE AND CONSEQUENTIAL DAMAGES, LOSS OF PROGRAMS OR DATA, INCOME OR PROFIT, LOSS OR DAMAGE TO PROPERTY, AND CLAIMS OF THIRD PARTIES.

D-Wave reserves the right to alter this document and other referenced documents without notice from time to time and at its sole discretion. D-Wave reserves its intellectual property rights in and to this document and its proprietary technology, including copyright, trademark rights, industrial design rights, and patent rights. D-Wave trademarks used herein include D-Wave®, D-Wave 2X™, D-Wave 2000Q™, Leap™, and the D-Wave logos (the D-Wave Marks). Other marks used in this document are the property of their respective owners. D-Wave does not grant any license, assignment, or other grant of interest in or to the copyright of this document, the D-Wave Marks, any other marks used in this document, or any other intellectual property rights used or referred to herein, except as D-Wave may expressly provide in a written agreement. This document may refer to other documents, including documents subject to the rights of third parties. Nothing in this document constitutes a grant by D-Wave of any license, assignment, or any other interest in the copyright or other intellectual property rights of such other documents. Any use of such other documents is subject to the rights of D-Wave and/or any applicable third parties in those documents.

All installation, service, support, and maintenance of and for the D-Wave System must be performed by qualified factory-trained D-Wave personnel. Do not move, repair, alter, modify, or change the D-Wave System. If the equipment is used in a manner not specified by D-Wave, the protection provided by the equipment may be impaired. Do not provide access to the customer site to anyone other than authorized and qualified personnel. Failure to follow these guidelines may result in disruption of service, extended downtime, damage to equipment (customer’s, D-Wave’s, and/or third parties’), injury, loss of life, or loss of property.
# Contents

1 Introduction
   1.1 About this Document ......................................................... 1
   1.2 Intended Audience ............................................................. 2
   1.3 Notation ........................................................................... 3
   1.4 Abbreviations .................................................................... 3
   1.5 Key Terminology .................................................................. 3
   1.6 Related Documents .............................................................. 5

2 Two Illustrative Examples ........................................................... 7
   2.1 Simple Map Coloring Problem .................................................. 7
      2.1.1 Formulating the Problem .................................................. 8
      2.1.2 Minor-Embedding the Problem .......................................... 11
      2.1.3 Decomposing ................................................................. 16
      2.1.4 Direct Formulations ......................................................... 16
   2.2 Simple Circuit Fault Diagnosis Problem .................................... 17
      2.2.1 Formulating the Problem .................................................. 17
      2.2.2 Minor-Embedding the Problem .......................................... 27
      2.2.3 Scalable, Tool-Assisted Formulation and Minor-Embedding ........ 31

3 Stating the Problem ...................................................................... 35
   3.1 Discrete (Combinatorial) Optimization Problems and CSPs ..................... 36
      3.1.1 Job-Shop Scheduling ......................................................... 36
      3.1.2 Fault Diagnosis ............................................................... 37
      3.1.3 Additional Problems ......................................................... 37
   3.2 Machine Learning Problems ..................................................... 38
      3.2.1 Probabilistic Sampling: RBM ............................................. 39
      3.2.2 Energy-Based Models ....................................................... 39
      3.2.3 Support Vector Machines ................................................ 40
   3.3 Opportunities and Challenges ................................................... 41

4 Reformulating a Problem ............................................................. 43
   4.1 Native Formulations: Ising and QUBO (and MIS) ................................. 43
      4.1.1 Illustrative Example: Map-Coloring Objective in Ising Formulation .... 44
   4.2 Weighted MAX-2-SAT to Ising/QUBO ......................................... 45
      4.2.1 Illustrative Example: Three-Variable MAX-2-SAT ..................... 46
   4.3 Non-Quadratic (Higher-Degree) Polynomials to Ising/QUBO ..................... 47
      4.3.1 Illustrative Example: Polynomial Reductions ............................ 47
   4.4 CSP to MIS ........................................................................ 49
   4.5 Soft CSP to MIS ................................................................... 50
   4.6 CSP Reformulation with Penalty Functions ....................................... 51
      4.6.1 Linear Equality Constraints ............................................... 52
      4.6.2 Illustrative Example: Map-Coloring with Linear Equality Constraints ... 52
      4.6.3 Linear Inequality Constraints .............................................. 53
      4.6.4 Nonlinear Constraints ....................................................... 54
   4.7 CSP Reformulation with Inverse Verification ....................................... 55
   4.8 Elementary Boolean Operations to QUBO ....................................... 56
Quantum computing has the potential to help solve some of the most complex technical, scientific, national defense, and commercial problems that organizations face. For quantum computing, as for classical, the first step in solving a problem is to express it in a mathematical formulation compatible with the underlying physical hardware.

D-Wave systems solve problems that can be mapped onto an Ising model or a quadratic unconstrained binary optimization (QUBO) problem.

**Ising:**

$$E(s|h,J) = \left\{ \sum_{i=1}^{N} h_i s_i + \sum_{i<j}^{N} J_{i,j} s_i s_j \right\} \quad s_i \in \{-1, +1\}$$

is an objective function of $N$ variables $s = [s_1, ..., s_N]$ corresponding to physical Ising spins, where $h_i$ are the biases and $J_{i,j}$ the couplings between spins.

**QUBO:**

$$E(x|Q) = \sum_{i \leq j}^{N} x_i Q_{i,j} x_j \quad x_i \in \{0, 1\}$$

is an objective function of $N$ binary variables represented as an upper-diagonal matrix $Q$, where diagonal terms are the linear coefficients and the nonzero off-diagonal terms the quadratic coefficients.

The mapped problem must be formulated to be compatible with the constraints of the physical system and as robust as possible to analog control errors.

### 1.1 About this Document

This document provides guidance on the steps needed to solve a given problem (the problem instance) on the D-Wave system. It is structured, as shown in Table 1.1, to align with that workflow.

The steps in this workflow are designed to formalize the process of mapping problems onto the quantum processing unit (QPU) architecture and integrating solutions from the QPU into algorithms that solve given problems. It also provides references to technical documents, examples of prototype applications, and links to helpful software tools.
Table 1.1: Problem Solving Workflow

<table>
<thead>
<tr>
<th>Step in Workflow</th>
<th>Chapter</th>
<th>Software Tools</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Formulate the problem</td>
<td>Stating the Problem</td>
<td>State as a CSP or MAX-2-SAT (optimization); define an RBM (machine learning)</td>
</tr>
<tr>
<td>2</td>
<td>Map to a supported formulation</td>
<td>Reformulating a Problem</td>
<td>Reformulate an integer problem to use binary variables; convert a non-quadratic (high-order) polynomial to a QUBO</td>
</tr>
<tr>
<td>3</td>
<td>Decompose¹ and allocate to classical and quantum resources</td>
<td>Decomposing a Large Problem</td>
<td>Use branch-and-bound methods to divide a large problem into smaller parts</td>
</tr>
<tr>
<td>4</td>
<td>Embed on QPU</td>
<td>Minor-Embedding a Problem</td>
<td>Configure repeated elements and connect those with chained qubits</td>
</tr>
<tr>
<td>5</td>
<td>Configure the QPU</td>
<td>Solving a Problem on the QPU</td>
<td>Submitting Problems to the QPU Use spin-reversal transforms to reduce errors; examine the annealing with reverse anneal</td>
</tr>
</tbody>
</table>

A useful approach to this guide is to first scan through to see “the lay of the land.” New users may benefit from the introductory problems of the Two Illustrative Examples chapter, which walk through simple examples, before diving into the more abstract descriptions of latter chapters. Otherwise, find the problem closest to your own field of work and follow that rather than peruse whole chapters.

1.2 Intended Audience

This document is for developers and researchers solving problems on the D-Wave system.

It is assumed that new users have first read the Getting Started with the D-Wave System guide.

The problems solved on quantum computers are often highly complex and some problems may belong to research fields that require specific domain knowledge. Although a full understanding of the formulations presented in this guide can require a strong grounding in combinatorial methods, logic, and so on, you may be able to solve a given problem with just one or two relevant techniques. Additionally, there are software tools that take care of the underlying algorithms.

¹ For problems that require more qubits than available.
1.3 Notation

Unless specified otherwise:

- Vectors are indicated in lowercase bold font; e.g., \( \mathbf{s} = [s_1, \ldots, s_n] \).
- Matrices are indicated in uppercase bold font; e.g., \( \mathbf{Q} \).
- Euclidean inner product of vectors is indicated with angular brackets; e.g., for \( \mathbf{a} \) and \( \mathbf{b} \), it is \( \langle \mathbf{a}, \mathbf{b} \rangle \).

1.4 Abbreviations

Table 1.2 defines some of the abbreviations that are used throughout this document.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQP</td>
<td>Binary quadratic program</td>
</tr>
<tr>
<td>CNF</td>
<td>Conjunctive normal form</td>
</tr>
<tr>
<td>CSP</td>
<td>Constraint satisfaction problem</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to analog converters</td>
</tr>
<tr>
<td>EBM</td>
<td>Energy-based model</td>
</tr>
<tr>
<td>ICE</td>
<td>Integrated control errors</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed integer program</td>
</tr>
<tr>
<td>MIS</td>
<td>Maximum independent set</td>
</tr>
<tr>
<td>QA</td>
<td>Quantum annealing</td>
</tr>
<tr>
<td>QMI</td>
<td>Quantum machine instruction</td>
</tr>
<tr>
<td>QPU</td>
<td>Quantum processing unit</td>
</tr>
<tr>
<td>QUBO</td>
<td>Quadratic unconstrained binary optimization</td>
</tr>
<tr>
<td>SAPI</td>
<td>Solver Application Programming Interface (API)</td>
</tr>
<tr>
<td>SAT</td>
<td>Satisfiability (problem)</td>
</tr>
<tr>
<td>SSVM</td>
<td>Structured support vector machines</td>
</tr>
<tr>
<td>SVM</td>
<td>Support vector machines</td>
</tr>
<tr>
<td>VFYC</td>
<td>Virtual full-yield Chimera</td>
</tr>
</tbody>
</table>

1.5 Key Terminology

It is assumed that new users have first read the *Getting Started with the D-Wave System* guide, which provides a full description of the following terminology that is used in this guide too. This section is meant as a reminder only.

**Bias:** the programmable quantity that controls the external magnetic field applied to a qubit. This field tilts the double-well potential, increasing the probability of the qubit ending up in the lower well.

**Chimera:** architecture of sets of connected unit cells, each with four horizontal qubits con-
nected to four vertical qubits via couplers. Unit cells are tiled vertically and horizontally with adjacent qubits connected, creating a lattice of sparsely connected qubits. (The D-Wave 2000Q QPU supports a C16 Chimera graph: its 2048 qubits are logically mapped into a 16x16 matrix of unit cells of 8 qubits.) Figure 1.1 depicts the top leftmost 2x2 cells of a Chimera graph.

![Figure 1.1: 2x2 unit cells of a Chimera graph.]

**Coupler:** can make two qubits tend to end up in the same state—both 0 or both 1—or it can make them tend to be in opposite states.

**Ground state:** lowest energy state during the anneal.

**Hamiltonian (classical):** a mathematical description of some physical system in terms of its energies. Input any particular state of the system, and the Hamiltonian returns the energy for that state.

**Hamiltonian (quantum):** a function that maps certain states, called eigenstates, to energies. Only when the system is in an eigenstate of the Hamiltonian is its energy well defined and called the eigenenergy. When the system is in any other state, its energy is uncertain. The collection of eigenstates with defined eigenenergies make up the eigenspectrum.

**Minimum gap:** The minimum distance between the ground state and the first excited state throughout any point in the anneal.

**Minor embedding:** the process of mapping logical qubits to physical qubits.

**Objective function:** a mathematical expression of the energy of a system as a function of binary variables representing the qubits.

**Quantum annealing:** harnesses the natural evolution of quantum states: you initialize the system in a delocalized state, gradually turn on the description of the problem you wish to solve, and quantum physics allows the system to follow these changes. The configuration at the end corresponds to the answer you are trying to find.

**Qubits:** quantum bits.
1.6 Related Documents

See also the following related documents:

- *Getting Started with the D-Wave System*
- *Technical Description of the D-Wave Quantum Processing Unit*
This chapter, building on the introductory descriptions of the *Getting Started with the D-Wave System* guide, illustrates the main steps of solving problems on the D-Wave system through two “toy” problems.

The first follows the intuitive approach of the *Getting Started with the D-Wave System* guide to further familiarize new users with the technology.

The second exploits the techniques of this guide but on a small-scale problem and favoring clarity over efficiency, while pointing out ways to expand the techniques used to larger problems.

### 2.1 Simple Map Coloring Problem

The map-coloring problem is to assign a color to each region of a map such that any two regions sharing a border have different colors.

![Figure 2.1: Coloring a map of Canada with four colors.](image)

A solution to a map-coloring problem for a map of Canada with four colors is shown in Figure 2.1.

The following is a simple example that demonstrates some of the workflow steps for formulating a given problem that can be solved on the D-Wave system. This section serves as an introduction to the techniques used in other sections.
2.1.1 Formulating the Problem

Generally, the first step in solving a problem is to state it mathematically. In this example, an intuitive approach is to think of the regions as variables representing the possible set of colors, the values of which must be selected from some numerical scheme, such as natural numbers. The selection function must express the problem’s constraints:

- Each region is assigned one color only, of \( C \) possible colors.
- The color assigned to one region cannot be assigned to adjacent regions.

Solving the problem means finding a permissible value for each of the variables.

Solving such a problem on the D-Wave system necessitates additional steps due to the following considerations:

- The mathematical formulation must use binary variables because the solution is implemented physically with qubits, and so must translate to spins \( s_i \in \{-1, +1\} \) or equivalent binary values \( x_i \in \{0, 1\} \).
- Relationships between variables must be reducible to quadratic (e.g., a QUBO) because the problem’s parameters are represented by qubits’ weights and couplers’ strengths on a QPU.
- The mathematical formulation should be sparing in its number of variables because a QPU has a limited number of qubits and couplers \(^1\).
- Alternative formulations may have different implications for performance.

In particular, the D-Wave system solves problems by minimizing an objective function, which expressed as a QUBO in scalar notation is

\[
E(a_i, b_{ij}; q_i) = \sum_i a_i q_i + \sum_{i<j} b_{ij} q_i q_j,
\]

where \( a_i \) are the biases or weights for each qubit, \( q_i \), and \( b_{ij} \) the coupler strength between qubits \( i \) and \( j \). The map coloring problem is solved by setting biases \( a_i \) and coupler strengths \( b_{ij} \) such that the qubits \( q_i \) in the minimized objective satisfy the constraints on coloring for all the regions.

Translating to Binary

The Reformulating a Problem chapter describes techniques to map problems with integer variables to binary formulations, and the Formulating an Integer CSP as a QUBO Problem section gives a larger example.

This example simply maps the \( C \) possible colors to a unary encoding (rather than using natural numbers). Each region is represented by \( C \) qubits, one for each possible color, which is set to value 1 if selected, while the remaining \( C - 1 \) qubits are 0.

Table 2.1 shows two schemes for representing colors for a map with \( C = 4 \) possible colors, where \( q_B \) is a qubit representing blue, \( q_G \) green, \( q_R \) red, and \( q_Y \) yellow.

----

\(^1\) The Decomposing a Large Problem chapter discusses this topic and approaches for solving problems with many variables on the D-Wave system.
Table 2.1: Translating Color to Binary.

<table>
<thead>
<tr>
<th>Color</th>
<th>Naturals</th>
<th>Unary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>1</td>
<td>( q_B, q_G, q_R, q_Y = 1, 0, 0, 0 )</td>
</tr>
<tr>
<td>Green</td>
<td>2</td>
<td>( q_B, q_G, q_R, q_Y = 0, 1, 0, 0 )</td>
</tr>
<tr>
<td>Red</td>
<td>3</td>
<td>( q_B, q_G, q_R, q_Y = 0, 0, 1, 0 )</td>
</tr>
<tr>
<td>Yellow</td>
<td>4</td>
<td>( q_B, q_G, q_R, q_Y = 0, 0, 0, 1 )</td>
</tr>
</tbody>
</table>

Expressing a Constraint

The Reformulating a Problem chapter describes various methods for incorporating constraints into the formulation of a problem. This example’s simple constraints are formulated intuitively as follows.\(^2\)

Consider for a two-color problem the constraint that a region be assigned a single color only. Here, the \( C = 2 \) colors are unary encoded by \( q_B, q_G \), with allowed states being either blue selected, \( q_B, q_G = 1, 0 \), or green selected, \( q_B, q_G = 0, 1 \), but not both or neither selected. The objective in QUBO format for two qubits, each with its bias and a single coupler between them, is

\[
E(a_i, b_{ij}; q_i) = a_B q_B + a_G q_G + b_{B,G} q_B q_G.
\]

Table 2.2 shows all the possible states for a single-region, two-color problem. In this table, column \( q_B \) and \( q_G \) are all possible states of the two qubits encoding the region's two colors; column **Constraint** shows whether or not the state of the qubits meet the constraint that one color is selected; column \( E(a_i, b_{ij}; q_i) \) is the energy calculated for the state from the objective function.

Table 2.2: States for a Constraint Selecting One of Two Colors.

<table>
<thead>
<tr>
<th>( q_B )</th>
<th>( q_G )</th>
<th>Constraint</th>
<th>( E(a_i, b_{ij}; q_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>Violates</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td>Meets</td>
<td>( a_G )</td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td>Meets</td>
<td>( a_B )</td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>Violates</td>
<td>( a_B + a_G + b_{B,G} )</td>
</tr>
</tbody>
</table>

Coefficients that minimize the objective for this constraint can be guessed from looking at Table 2.2 and the following considerations:

- Both colors are equally preferred, so setting \( a_B = a_G \) ensures identical energy for both states that meet the constraint (\( q_B, q_G = 0, 1 \) and \( q_B, q_G = 1, 0 \)).
- The state \( q_B, q_G = 0, 0 \), which violates the constraint, has energy 0, so to ensure that the non-violating states have lower energy, set \( a_B = a_G < 0 \).
- Having set the minimum energy to negative for the valid states, setting \( a_B + a_G + b_{B,G} = 0 \) for non-valid state \( q_B, q_G = 1, 1 \) ensures that state has a higher energy.

One solution\(^3\) is to set \( a_B = a_G = -1 \) and \( b_{B,G} = 2 \). Setting these values for the qubit biases and coupler strength sets the minimum energy for this objective to \(-1\) for both valid states.

---

\(^2\) This is explained in more detail for a simple two-qubit example in Getting Started with the D-Wave System.\(^3\) Getting Started with the D-Wave System discusses the effects of different choices of values in the section on
and zero for the states that violate the constraint; that is, this objective has lowest energy if just one color is selected (only one qubit is 1), where either color (qubit) has an equal probability of selection.

The two-color formulation above can be easily expanded to a three-color problem: \( C = 3 \) possible colors are encoded as \( q_B, q_G, q_R \). The objective to minimize in QUBO format is

\[
E(a_i, b_{ij}, q_i) = a_Bq_B + a_Gq_G + a_Rq_R + b_{B,G}q_Bq_G + b_{B,R}q_Bq_R + b_{G,R}q_Gq_R.
\]

Coefficients that minimize the objective can be guessed by applying the previous considerations more generically. All colors are equally preferred, so set \( a_i = a \) to ensure identical energy for states that meet the constraint. Simplify by setting \( b_{ij} = b = 2 \) and \( a = -1 \) for an objective

\[
E(a, b; q_i) = a(q_B + q_G + q_R) + b(q_Bq_G + q_Bq_R + q_Gq_R)
= -(q_B + q_G + q_R) + 2(q_Bq_G + q_Bq_R + q_Gq_R).
\]

Table 2.3 enumerates all states of the objective for three possible colors. The meanings of the columns in this table are identical to those for Table 2.2 above.

<table>
<thead>
<tr>
<th>( q_B, q_G, q_R )</th>
<th>Constraint</th>
<th>( E(a = -1, b = 2; q_i) )</th>
</tr>
</thead>
</table>
| 0,0,0                | Violates   | \(- (0) + 2 
| 0,0,1                | Meets      | \(- (1) + 2 
| 0,1,0                | Meets      | \(- (1) + 2 
| 0,1,1                | Violates   | \(- (2) + 2 
| 1,0,0                | Meets      | \(- (1) + 2 
| 1,0,1                | Violates   | \(- (2) + 2 
| 1,1,0                | Violates   | \(- (2) + 2 
| 1,1,1                | Violates   | \(- (3) + 2 |

Table 2.3: States for a Constraint Selecting One of Three Colors.

Again, the minimum energy for this objective is \(-1 \) for valid states and higher for states that violate the constraint; that is, this objective has lowest energy if just one color is selected (one qubit is 1), where the three colors (qubits) have equal probability of selection.

For the four-color map problem, the one-color-per-region constraint can likewise be simplified to a QUBO objective, with \( b_{ij} = b = 2 \) and \( a_i = a = -1 \),

\[
E(a, b; q_i) = -(q_B + q_G + q_R + q_Y) + 2(q_Bq_G + q_Bq_R + q_Bq_Y + q_Gq_R + q_Gq_Y + q_Rq_Y).
\]

The second constraint, that adjacent regions have different colors, can be expressed by weighting the qubits of each region and the couplers between regions to minimize an objective function when different colors are selected for adjacent regions. That is, if \( q_B = 1 \) (blue is selected) on one side of a shared border, for example, the objective has lower energy if \( q_B = 0 \) (blue is not selected) on the other side. In this way, a state of \( q_B, q_G, q_R, q_Y = 1,0,0,0 \) (blue selected) in one region and, for example, \( q_B, q_G, q_R, q_Y = 0,1,0,0 \) (green selected) in an adjacent region has lower energy than the same state (the same color selected) in both.

By configuring both types of constraint together, as shown in the following sections, the combined objective has lowest energy states when simultaneously each region is assigned a single color and adjacent regions do not select the same colors.

problem scaling. See also the considerations discussed in the Overcoming Imprecisions of Qubit Biases and Coupling Strengths section.
2.1.2 Minor-Embedding the Problem

The D-Wave QPU minimizes the energy of an Ising spin configuration whose pairwise interactions lie on the edges of a $M, N, L$ Chimera graph. To solve a given Ising spin problem with arbitrary pairwise interaction structure, you *minor embed* its graph into a Chimera graph by using qubits to represent missing edges. Figure 2.2 depicts the map-coloring problem as a graph, where each region is a node (vertex) and adjacent borders are edges.

![Map of Canada](image)

*Figure 2.2: Map of Canada represented graphically with regions as connected nodes.*

The *Minor-Embedding a Problem* chapter describes various methods for embedding a given problem. This example illustrates an embedding for the simple four-color map problem.

**Minor-Embed a Region**

Similar to the simplified approach to constraints in the previous section, consider first embedding in the context of a single region with a unary encoding scheme for color. Figure 2.3 shows the constraints of the previous $C=2, 3, 4$ map-coloring problems as graphs.

![Color constraints](image)

*Figure 2.3: Color constraints represented graphically for two, three, and four colors with unary encoding.*

The constraints of a four-color map problem require full connectivity between the four qubits encoding the color for each region. This example accomplishes that by embedding each region in a unit cell (8 qubits and 16 couplers on the D-Wave 2000Q QPU), shown in Figure 2.4.
Logical and Physical Qubits

To map a complete four-vertex graph to a Chimera unit cell, it helps to introduce a distinction between the logical qubits \( q_B, q_G, q_R, q_Y \) and couplers of the previous sections and the QPU’s physical qubits \( q_0, q_1, \ldots \) and couplers: minor-embedding maps each logical qubit to one or more connected physical qubits in a chain. To embed a graph into the Chimera topology (i.e., to implement a coupler between logical qubits with a shared edge), find a physical coupler between any physical qubits in the two chains representing the logical qubits.

This example embeds the four logical qubits \( q_B, q_G, q_R, q_Y \) representing four colors for a region in four chains of two physical qubits each: the blue logical qubit is a coupling of the top two physical qubits in the D-Wave unit cell; green, red, and yellow are rows 2, 3, and 4 respectively. As shown in Figure 2.5, representing each logical qubit as a chain of two physical qubits enables full connectivity between the four.

Figure 2.5: Embedding a four-color unary-encoded region in a unit cell on the D-Wave 2000Q QPU. On the left, each of the logical qubits \( q_B, q_G, q_R, q_Y \) is represented by a colored dot and the logical couplers between them by arrows. On the right, the 8 physical qubits of a single unit cell are represented by dots and the 16 couplers by lines. The 4 colored areas represent logical qubits, each representing a chain of two physical qubits.
Expressing a Chain

A chain is formulated as a constraint: all its member physical qubits represent a single logical qubit; i.e., all the qubits in a chain should have identical spins \( s_i \in \{-1, +1\} \), in an Ising formulation, or binary values \( x_i \in \{0, 1\} \), in a QUBO formulation.

Similar to the single-color-per-region constraint formulated above for a two-color problem, the constraint that a two-qubit chain represents a logical qubit for a color (e.g., \( q_B \), the logical qubit representing blue shown in Figure 2.5) has an objective of the form:

\[
E(a_i, b_i; q_B) = a_1 q_{B1} + a_2 q_{B2} + b_{1,2} q_{B1} q_{B2},
\]

where \( q_{B1} \) are the two physical qubits of the logical qubit for blue. For example, in a region embedded in the topmost left unit cell, these are physical qubits \( q_0 \) and \( q_4 \) shown in Figure 2.6.

![Figure 2.6: Top-left part of the Chimera graph.](image)

Table 2.4 enumerates the possible states of the objective for a constraint that formulates a logical qubit for blue. The meanings of the columns in this table are identical to those for Table 2.2 above.

<table>
<thead>
<tr>
<th>( q_{B1} )</th>
<th>( q_{B2} )</th>
<th>Constraint</th>
<th>( E(a_i, b_i; q_B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Meets</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Violates</td>
<td>( a_2 )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Violates</td>
<td>( a_1 )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Meets</td>
<td>( a_1 + a_2 + b_{1,2} )</td>
</tr>
</tbody>
</table>

Coefficients that minimize the objective for this constraint are guessed from looking at Table 2.4 and the following considerations:

- The valid state \( q_{B1}, q_{B2} = 0, 0 \) (blue is not selected) has energy 0, so to ensure that the violating states have higher energy, set \( a_1, a_2 > 0 \).
- The chain’s constraint should not bias whether the color blue is selected or not—it’s task is solely that the two physical qubits represent a single logical qubit—so in the context of this constraint, both valid states are equally preferred. Setting \( a_1 + a_2 + b_{1,2} \)
\[ b_{1,2} = 0 \text{ for valid state } q_{B1}, q_{B2} = 1, 1 \text{ (blue is selected) ensures that state has the same energy as } q_{B1}, q_{B2} = 0, 0 \text{ (blue is not selected). This requires that } b_{1,2} = -|a_1 + a_2|. \]

Again one solution is to set \( a_1 = a_2 = 1 \) and \( b_{1,2} = -2 \). Configuring these values for the qubit biases and coupler strength sets the minimum energy for this objective to 0 for both valid states and 1 for the states that violate the constraint; that is, this objective has lowest energy if the two qubits in the chain are aligned.

Similar constraints create chains for logical qubits \( q_G \) (green), \( q_R \) (red), and \( q_Y \) (yellow).

Having aligned the two physical qubits of a chain (e.g., \( q_0, q_4 \)) of a logical qubit for each color (e.g., \( q_B \)), the four logical qubits \( (q_B, q_G, q_R, q_Y) \) in each region must be assigned their individual biases and coupling strengths. For the four-color map problem, the constraint on one color per region was formulated in the previous section with an objective having the following coefficients:

- Equal biases of \( a = -1 \) for the logical qubits.
- Coupler strengths of \( b = 2 \).

**Coupling Two Regions**

In this simple example, the structures of the problem and the Chimera topology align well, and it’s easy to couple the logical qubits of two adjacent regions to embed the constraint that adjacent regions select different colors. For example, to find a physical coupler connecting British Columbia’s red qubit to Alberta’s red qubit, look at a slightly larger portion of the Chimera graph shown in Figure 2.7: black arcs highlight the physical couplers connecting two adjacent unit cells. Clearly these couplers are ideally positioned to penalize the state where the chains representing the same color are turned on in both unit cells, thereby implementing the constraint that adjacent regions select different colors.

Denoting British Columbia’s red qubit as \( q_{BC}^R \) and Alberta’s red qubit as \( q_{AB}^R \), the constraint that adjacent regions have different colors can be formulated similarly to the objective previously constructed to ensure two coupled qubits have different values:

\[ E(a_i, b_{i,j}; q_i) = a_{BC}^R q_{BC}^R + a_{AB}^R q_{AB}^R + b_{BC, AB}^R q_{BC}^R q_{AB}^R \]

Configuring \( a_{BC}^R = a_{AB}^R = 0 \) and \( b_{BC, AB}^R = 1 \) sets a higher energy for the state \( q_{BC}^R = q_{AB}^R \) (red selected in both British Columbia and Alberta), penalizing that invalid state, while leaving the valid three states with an energy of 0.

**Coupling Several Regions**

The structural alignment between the Chimera graph and the four-color problem is ideal for British Columbia and Alberta but does not extend to the three-way neighbor relation
Coupling two four-color, unary-encoded regions, British Columbia and Alberta, each represented by a unit cell on the D-Wave 2000Q QPU. On the right, the black lines represent physical couplers between logical qubits, represented as colored areas containing two physical qubits in a chain.

of British Columbia, Alberta, and the Northwest Territories (shown on the Canadian map of Figure 2.1). Because unit cells are configured in a two-dimensional checkerboard array, these three regions cannot be mapped to unit cells while preserving the three-way neighbor relation. For example, assigning British Columbia and Alberta to unit cells that neighbor each other horizontally, as in Figure 2.7, leaves the Northwest Territories unit cells to the left of British Columbia, to the right of Alberta, or below either, but never as a direct neighbor of both. In such a configuration, no physical couplers are available to ensure that the same color qubits in these three regions are not simultaneously activated.

Introducing clones—in this case, multiple unit cells represent a region—overcomes this in a way analogous to the chaining of physical qubits to represent a logical qubit. Just as chains extend the footprint of a logical qubit in the graph of physical qubits, clones, by extending the footprint of a single region in the array of unit cells, add neighbors to the cloned region. This enables the enforcement of constraints to cases where neighbors on the map do not transfer directly to adjacent unit cells.

Figure 2.8 shows the use of cloned regions to enable coupling of all neighboring regions in Canada’s map.

Figure 2.8: Mapping of Canada’s 13 regions to unit cells in the Chimera graph of a D-Wave 2000Q; regions and clones are labeled with standard two-letter postal codes, shown in Figure 2.2.
Set the strengths of intercell couplers so the same color is selected for all clones of a region, in the same way as for chains of physical qubits representing a logical qubit. For example, for the physical coupler connecting the red qubits of the top and bottom clones of Alberta, set a strength of $-2$ and add a weight of 1 to the two physical qubits connected via that coupler. Repeat for each of the other three colors, and do the same for the cloned British Columbia and Northwest Territories.

2.1.3 Decomposing

The previous sections of this example explain the steps of solving a small problem: coloring a map of Canada’s 13 regions using four colors, as shown in Figure 2.1. This section briefly looks at how this solution can scale to larger problems, for example to maps with more regions, more colors, or both.

The Decomposing a Large Problem chapter describes some techniques for decomposing large problems into smaller ones that can be solved on the QPU. To solve the map-coloring problem on a map of the United States, use a divide-and-conquer technique: divide the US map into chunks; process the first chunk (find valid colorings for the first set of states); use these colorings to bias the second chunk; repeat until the solution is completed.

Figure 2.9 shows a solution to coloring a map of the United States, which ran on the D-Wave 2000Q in two manageable chunks.

![Figure 2.9: A map of the US colored through division into two smaller parts. The larger of the two, on the left, is just small enough to fit on the D-Wave 2000Q QPU.](image)

2.1.4 Direct Formulations

The previous sections formulated the map-coloring problem in several easily grasped steps. Experienced users might formulate a more direct approach as follows.

For graph $G(V, E)$ of the map problem—no two vertices, $V$, connected by an edge, $E$, should select the same color from set $C$—construct a cost function with binary variables, $x_{v,c} = 1$ when $v \in V$ selects color $c \in C$, by implementing two constraints:

\[ \left( \sum_c x_{v,c} - 1 \right)^2, \]
which has minimum energy (zero) when vertices select one color only, and

$$\sum_c \sum_{v_a, v_b \in E} x_{v_a, c} x_{v_b, c},$$

which adds a penalty if the vertices of an edge select the same color.

These constraints give a QUBO,

$$E(x_{v_a, v_b}) = \sum_v (\sum_c x_{v_c} - 1)^2 + \sum_c \sum_{v_a, v_b \in E} x_{v_a, c} x_{v_b, c}.$$

The minima (ground states) of this QUBO have zero energy for viable solutions. This formulation of the generic problem must still be applied to the map and color set and then embedded.

This section solved the map-coloring problem using a technique of formulating a problem as a constraint satisfaction problem (CSP) using penalty functions. The CSP Reformulation with Penalty Functions section describes this technique and demonstrates it in detail on a simple two-color, two-region part of this map-coloring problem in the Example of CSP reformulation with penalty functions section.

Further Information

- [Dwave4] is a whitepaper on the map coloring problem.
- [Rie2014] describes mappings of planning problems to QUBOs, including map coloring.
- [Bis2017] describes a formulation of this problem.

## 2.2 Simple Circuit Fault Diagnosis Problem

Fault diagnosis is the combinational problem of quickly localizing failures as soon as they are detected in systems. Circuit fault diagnosis (CFD) is the problem of identifying a minimum-sized set of gates that, if faulty, explains an observation of incorrect outputs given a set of inputs.

A simple circuit is shown in Figure 2.10. The following example demonstrates some of the techniques available to formulate a given problem so it can be solved on the D-Wave system.

### 2.2.1 Formulating the Problem

As in the Simple Map Coloring Problem section above, this example solves the CFD problem by configuring the objective function expressed as a scalar QUBO,

$$E(a_i, b_{i,j}; q_i) = \sum_i a_i q_i + \sum_{i<j} b_{i,j} q_i q_j,$$

where $a_i$ are the biases or weights for each qubit, $q_i$, and $b_{i,j}$ the coupler strength between qubits $i$ and $j$. The problem is configured by setting biases $a_i$ and coupler strengths $b_{i,j}$ such that the qubits $q_i$ of the minimized objective function represent the minimum number of faults given the input and output values of the circuit. CFD solutions are those configurations that have the lowest energy levels.
Figure 2.10: The correct output of this circuit for inputs $x_1, x_2, x_3 = 1, 1, 1$ is $z_1, z_2 = 0, 0$. The circuit fault diagnosis problem is to find a minimum number of faulty gates that explains an incorrect output such as $z_1, z_2 = 1, 0$.

Figure 2.11: A small example to understand the connection between energy levels and the CFD solution: three serial NOT gates.

A Short Digression: An Intuitive Explanation of the CFD Solution

To understand the relationship between the CFD solutions and low-energy configurations, it’s helpful to look at an even smaller example.

The circuit in Figure 2.11 has three NOT gates in series, with an input $q_0$, an output $q_3$, and intermediate input/outputs $q_1$ and $q_2$. The following sections of the CFD example configure an objective function that has a higher energy level when a gate is faulty. Assuming that this serial NOT circuit has such an objective function, where the energy level increases by a value of 1 for any malfunctioning gate in the circuit, energy levels for all possible circuit states with an incorrect output are shown in Table 2.5.

In this table, the first column, Circuit Input ($q_0$), is the binary input value applied to the circuit in testing, and the second column, Incorrect Output ($q_3$), is the measured output of the circuit with an error detected. Columns $q_1$ and $q_2$ are all possible intermediate input/outputs of the circuit’s gates. Column Faulty Gate shows which gates are malfunctioning given the input and output states. Column Energy is the energy level of the objective function.
Table 2.5: Serial NOT Energy Levels

<table>
<thead>
<tr>
<th>Circuit Input (q₀)</th>
<th>Incorrect Output (q₃)</th>
<th>q₁</th>
<th>q₂</th>
<th>Faulty Gate</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,2,3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1,2,3</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

In Table 2.5, for the first and last rows, all three gates are malfunctioning, each increasing the energy level by 1 for a total energy level of 3:

First Row: \( (q₁ = 0) \neq \neg(q₀ = 0) \) \( (q₂ = 0) \neq \neg(q₁ = 0) \) \( (q₃ = 0) \neq \neg(q₂ = 0) \)

Last Row: \( (q₁ = 1) \neq \neg(q₀ = 1) \) \( (q₂ = 1) \neq \neg(q₁ = 1) \) \( (q₃ = 1) \neq \neg(q₂ = 1) \)

For the second to seventh rows, the circuit has a single malfunctioning gate, increasing the energy level by 1; for example, in the second row, the gate 1 is the only malfunctioning gate:

Second Row: \( (q₁ = 0) \neq \neg(q₀ = 0) \) \( (q₂ = 1) = \neg(q₁ = 0) \) \( (q₃ = 0) = \neg(q₂ = 1) \)

The minimum-sized set of gates that, if faulty, explains the observation of incorrect outputs given a set of inputs, are the single malfunctioning gates of rows 2–7. These have the lowest non-zero energy for the serial NOT circuit.

Intuitively, imagine that an objective function has been minor embedded on the QPU such that \( qᵢ \) are physical qubits representing the inputs and outputs of the circuit and its NOT gates. To mimic the behavior of a NOT operation, the relationship between each sequential pair of qubits should resemble anti-ferromagnetic coupling: the likely opposite binary states (0/1) of a NOT operation, such as that of gate 1, is represented on the QPU by positively coupling physical qubits \( q₀ \) and \( q₁ \), meaning the lowest energy states have opposite spins (-1/1 or 1/-1). For a functioning circuit, the four qubits line up “head-to-tail”; but when the circuit’s output, represented by \( q₃ \), is clamped in a faulty state, at least one sequential pair of qubits must have their spins aligned despite the anti-ferromagnetic coupling, raising the energy level.

Return to the CFD Example

The Simple Map Coloring Problem section above followed an intuitive approach to finding an objective function. This example exploits the methods of the Reformulating a Problem and Minor-Embedding a Problem chapters to formulate and embed the objective function as follows:

1. Using the techniques of CSP Reformulation with Penalty Functions and Elementary Boolean Operations to QUBO, the Boolean operation of each gate is reformulated as a penalty function that penalizes its incorrect operation; that is, configurations that represent a gate producing incorrect output for its given inputs have higher energy levels than valid configurations. The greater the number of malfunctioning gates, the
higher the energy level of the objective function, which sums all the penalty functions. By finding configurations with the lowest energies, the D-Wave system can discover the configurations with minimum faults that produce incorrect outputs.

2. Penalty functions are normalized so any single fault results in the same penalty value. (If faulty states have different penalties, the combined penalty on two faults might produce a lower energy level than a single fault, in which case the set of minimum-fault configurations is not the set of configurations with lowest energies.)

3. The penalty functions are embedded on the QPU. The embedding demonstrated in this example exploits the method of Pre-embedding Local Constraint Structures for locally-structured embeddings described in the Minor-Embedding a Problem chapter. These embeddings are chosen for their explanatory advantages, not their efficiency; see the General Considerations for considerations used to select embeddings in actual problems.

4. The use of software tools to automate the previous steps is discussed.

Representing Boolean Gates as Penalty Functions

This example maps the circuit’s gates to penalty functions using the formulations of the Elementary Boolean Operations to QUBO section. These penalty functions increase the energy level of the problem’s objective function by penalizing non-valid states of the Boolean operations, which represent malfunctioning gates.

1. The NOT gate

   \[ z \leftrightarrow \neg x : 2xz - x - z + 1. \]

   Table 2.6 shows that this function penalizes states that represent a malfunctioning gate while no penalty is applied to a functioning gate. In this table, column in is all possible states of the gate’s inputs; column out_valid is the corresponding output values of a functioning gate while out_fault is the corresponding output values of a malfunctioning gate, which are simply \( \text{out\_fault} = \neg \text{out\_valid} \); column \( P_{\text{valid}} \) is the value the penalty function adds to the energy of the objective function when the gate is functioning (zero, a functioning gate must not be penalized) while column \( P_{\text{fault}} \) is the value the penalty function adds when the gate is malfunctioning (nonzero, the objective function must be penalized with a higher energy).

<table>
<thead>
<tr>
<th>in</th>
<th>out_valid</th>
<th>out_fault</th>
<th>P_valid</th>
<th>P_fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

   For example, the state \( \text{in}, \text{out\_valid} = 0,1 \) of the first row is represented by the penalty function with \( x = 0 \) and \( z = 1 = \neg x \). For this functioning gate, the value of \( P_{\text{valid}} \) is

   \[ 2xz - x - z + 1 = 2 \times 0 \times 1 - 0 - 1 + 1 = -1 + 1 = 0, \]
not penalizing the valid configuration. In contrast, the state \( \text{in}, \text{out}_{\text{fault}} = 0, 0 \) of the first row is represented by the penalty function with \( x = 0 \) and \( z = 0 \neq \neg x \). For this malfunctioning gate, the value of \( P_{\text{fault}} \) is

\[
2xz - x - z + 1 = 2 \times 0 \times 0 - 0 - 0 + 1 = 1,
\]

adding an energy cost of 1 to the incorrect configuration.

2. The AND gate

AND can be represented as penalty function

\[
z \iff x_1 \land x_2 : x_1 x_2 - 2(x_1 + x_2)z + 3z.
\]

Table 2.7 shows that this function penalizes states that represent a malfunctioning gate while no penalty is applied to a functioning gate. The meanings of the columns in this table are identical to those for the NOT gate above.

Table 2.7: Boolean AND Operation as a Penalty.

<table>
<thead>
<tr>
<th>in</th>
<th>out_{valid}</th>
<th>out_{fault}</th>
<th>( P_{\text{valid}} )</th>
<th>( P_{\text{fault}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0, 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1, 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1, 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

For example, the state \( \text{in} = 0, 0; \text{out}_{\text{valid}} = 0 \) of the first row is represented by the penalty function with \( x_1 = x_2 = 0 \) and \( z = 0 = x_1 \land x_2 \). For this functioning gate, the value of \( P_{\text{valid}} \) is

\[
x_1 x_2 - 2(x_1 + x_2)z + 3z = 0 \times 0 - 2 \times (0 + 0) \times 0 + 3 \times 0 = 0,
\]

not penalizing the valid configuration. In contrast, the state \( \text{in} = 0, 0; \text{out}_{\text{fault}} = 1 \) of the same row is represented by the penalty function with \( x_1 = x_2 = 0 \) and \( z = 1 \neq x_1 \land x_2 \). For this malfunctioning gate, the value of \( P_{\text{fault}} \) is

\[
x_1 x_2 - 2(x_1 + x_2)z + 3z = 0 \times 0 - 2 \times (0 + 0) \times 1 + 3 \times 1 = 3,
\]

adding an energy cost of 3 to the incorrect configuration.

3. The XOR gate

XOR can be represented as penalty function

\[
z \iff x_1 \oplus x_2 : 2x_1 x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a,
\]

where \( a \) is an ancillary variable as described in the Non-Quadratic (Higher-Degree) Polynomials to Ising/QUBO section of chapter Reformulating a Problem (and shown below).

Table 2.8 shows that no penalty is applied to a functioning gate when this penalty function is minimized (when this function is part of an objective
function being minimized, its value is zero for at least one of the possible values of ancillary variable $a$). In this table, column $\text{in}$ is all possible states of the gate’s inputs $x_1, x_2$; column $\text{out}$ is the corresponding output values of a functioning gate $x_1 \oplus x_2$; columns $P_{a=0}$ and $P_{a=1}$ are the values of the penalty function if the ancillary variable $a = 0$ or $a = 1$ respectively; and the final column, $\min_a P$, is the value the penalty function adds to the minimized objective function (the minimum penalty of $P_{a=0}$ or $P_{a=1}$).

Table 2.8: Boolean XOR Operation as a Penalty: Functioning Gate.

<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
<th>$P_{a=0}$</th>
<th>$P_{a=1}$</th>
<th>$\min_a P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0,1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1,0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1,1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For example, the state $\text{in} = 0, 0$ of the first row is represented by the penalty function with $z = 0 \iff x_1 \oplus x_2 = 0 \oplus 0 = 0$. For this functioning gate, the value of $P$ for each possible value of ancillary variable $a$ is

$$a = 0: \quad 2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a$$
$$= 2 \times 0 \times 0 - 2 \times (0 + 0) \times 0 - 4 \times (0 + 0) \times 0 + 4 \times 0 + 0 + 0 + 0 + 4 \times 0$$
$$= 0$$

$$a = 1: \quad 2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a$$
$$= 2 \times 0 \times 0 - 2 \times (0 + 0) \times 0 - 4 \times (0 + 0) \times 1 + 4 \times 1 \times 0 + 0 + 0 + 0 + 4 \times 1$$
$$= 4.$$  

When this penalty function is part of an objective function being minimized, the solution with $a = 0$, which contributes less to the total energy level, is preferred, giving a penalty of zero for the functioning gate.

Table 2.9 shows that a non-zero penalty is applied to a malfunctioning gate.  
In this table, column $\text{out}$ is the output values of a malfunctioning gate $\text{out} = \neg(x_1 \oplus x_2)$; the other columns are identical to those for Table 2.8 above.

Table 2.9: Boolean XOR Operation as a Penalty: Malfunctioning Gate.

<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
<th>$P_{a=0}$</th>
<th>$P_{a=1}$</th>
<th>$\min_a P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>0,1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For example, the state $\text{in} = 0, 0$ of the first row is represented by the penalty function with $z = 1 \iff x_1 \oplus x_2 = 0 \oplus 0 = 0$. For this malfunctioning gate,
the value of $P$ for each possible value of ancillary variable $a$ is

- $a = 0$:
  \[2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a = 1\]
  \[2 \times 0 \times 0 - 2 \times (0 + 0) \times 1 - 4 \times (0 + 0) \times 0 + 4 \times 0 \times 1 + 0 + 0 + 1 + 4 \times 0 = 1\]

- $a = 1$:
  \[2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a = 9\]
  \[2 \times 0 \times 0 - 2 \times (0 + 0) \times 1 - 4 \times (0 + 0) \times 1 + 4 \times 1 \times 1 + 0 + 0 + 1 + 4 \times 1 = 9\]

When this penalty function is part of an objective function being minimized, the solution with $a = 0$, which contributes less to the total energy level, is preferred, giving a penalty of 1 for the malfunctioning gate.

### Normalizing the Penalty Functions

To solve the CFD problem by finding a minimum-sized set of faulty gates, a search for lowest energy levels of the objective function should identify all single faults. However, if functions impose different penalties for faulty states—if some faults are penalized to a higher value than others—two faults with low penalties might produce a lower energy level than a strongly penalized single fault.

The penalty functions of the previous section impose a penalty of 1 on all gate faults with one exception: Table 2.7 shows that when the state $in = 0, 0$ of the AND gate incorrectly outputs $out = 1$, the penalty imposed is $P_{\text{fault}} = 3$. This section aims to normalize the penalty functions; that is, to adjust the penalty functions so any single fault has an identical penalty, contributing equally to the objective function of the CFD problem.

One approach to normalizing the penalty function of the AND gate is to add a penalty function that imposes a reward on (a negative penalty to compensate for) the state $in = 0, 0; out = 1$ with its higher penalty value. The additional penalty function, $P^*$, rewards this state with a negative penalty of value,

\[P_{\text{fault}}(0, 0; 1) + P^*(0, 0; 1) = 3 + P^*(0, 0; 1) = 1\]

Because both penalties are activated together in state $in = 0, 0; out = 1$, the total penalty they impose is 1, identical to the penalties of other single faults for the circuit’s gates.

The binary equivalence,

\[in_1, in_2, out = 0, 0, 1 \leftrightarrow \bar{in}_1, \bar{in}_2, out = 1, 1, 1,\]

makes it straightforward to guess a penalty function, $P^*$, that activates a reward of $-2$ for the AND gate state of $\bar{in}_1, \bar{in}_2, out = 1, 1, 1$:

\[P^* = -2\bar{in}_1\bar{in}_2out.\]

Clearly this penalty function is non-zero only for faulty state $in_1, in_2, out = 0, 0, 1$, in which case it is equal to $-2$. Adding this penalty to the problem’s objective function normalizes the penalty function for the AND operation.
For example, recalculating the penalty given as an example in the previous section, the state \( \text{in} = 0, 0; \text{out}_{\text{valid}} = 0 \) of the first row of Table 2.7 is represented by the penalty function with \( x_1 = x_2 = 0 \) and \( z = 0 = x_1 \land x_2 \). For this functioning gate, the value of \( P_{\text{valid}} + P^* \) is

\[
\begin{align*}
\frac{P_{\text{valid}}}{x_1 x_2 - 2(x_1 + x_2)z + 3z + -2x_1 x_2 z} + \\
\frac{P^*}{0 + -2 \times 1 \times 1 \times 0} \\
= 0,
\end{align*}
\]

not penalizing the valid solution. In contrast, the state \( \text{in} = 0, 0; \text{out}_{\text{fault}} = 1 \) of the same table row is represented by the penalty function with \( x_1 = x_2 = 0 \) and \( z = 1 \neq x_1 \land x_2 \). For this malfunctioning gate, the value of \( P_{\text{fault}} + P^* \) is

\[
\begin{align*}
\frac{P_{\text{fault}}}{x_1 x_2 - 2(x_1 + x_2)z + 3z + -2x_1 x_2 z} + \\
\frac{P^*}{3 + -2 \times 1 \times 1 \times 1} \\
= 3 - 2 \\
= 1,
\end{align*}
\]

reducing the energy cost of the original AND gate from 3 to 1 for the incorrect configuration. For all other states, both valid and faulty, the normalizing penalty, \( P^* \), is zero and has no other affect on the penalty function for AND.

**Reformulating Non-Quadratic Penalties as Quadratic**

The normalizing penalty introduced in the previous section, \( P^* = -2 \overline{in}_1 \overline{in}_2 \overline{out} \), is cubic and must be reformulated as quadratic to be mapped onto the native (Ising or QUBO) quadratic formulations used by the D-Wave system.

This example reformulates the normalizing penalty function using the technique of reduction by minimum selection, as described in section *Non-Quadratic (Higher-Degree) Polynomials to Ising/QUBO*, with the substitution,

\[ axyz = aw(x + y + z - 2) \quad a < 0, \]

where \( a \) is just a constant that must be negative (and for \( P^* \) equals \(-2\)), \( x, y, z \) are the relevant variables (here \( \overline{in}_1, \overline{in}_2, \overline{out} \)), and \( w \) an ancillary variable \(^4\) used by this technique to reduce the cubic term, \( xyz \), to a summation of quadratic terms (e.g., \( aux \)).

Applying that to \( P^* \), with ancillary variable \( w \) of the general formula renamed to \( b \), gives \( P^q \), a quadratic formulation of the cubic normalizing penalty function:

\[ P^* = -2\overline{in}_1 \overline{in}_2 \overline{out} \quad \rightarrow \quad P^q = -2b(\overline{in}_1 + \overline{in}_2 + \overline{out} - 2) \]

Table 2.10 shows that the cubic and quadratic formulations of the normalizing penalty function produce identical results. In this table, column \( \overline{in}_1, \overline{in}_2, \overline{out} \) is all possible states

\(^4\) Similar to ancillary variable \( a \) used in the previous section for the penalty function of the AND gate.
of the gate’s inputs and output; column \( P^* \) is the corresponding values of the cubic normalizing penalty function, \( P^* = -2\overline{m}_1\overline{m}_2out \); column \( P^q|_{b=0} \) the values of the quadratic substitute if the ancillary variable \( b = 0 \) and column \( P^q|_{b=1} \) the values if \( b = 1 \); column \( \min_b P^q \) is the value the penalty function adds to the minimized objective function (the minimum penalty of \( P^q|_{b=0} \) or \( P^q|_{b=1} \)).

Table 2.10: Quadratic Normalizing Penalty Function.

| \( \text{in}_1, \text{in}_2, \text{out} \) | \( P^* \) | \( P^q|_{b=0} \) | \( P^q|_{b=1} \) | \( \min_b P^q \) |
|---|---|---|---|---|
| 0,0,0 | 0 | 0 | 0 | 0 |
| 0,0,1 | -2 | 0 | -2 | -2 |
| 0,1,0 | 0 | 0 | 2 | 0 |
| 0,1,1 | 0 | 0 | 0 | 0 |
| 1,0,0 | 0 | 0 | 2 | 0 |
| 1,0,1 | 0 | 0 | 0 | 0 |
| 1,1,0 | 0 | 0 | 4 | 0 |
| 1,1,1 | 0 | 0 | 2 | 0 |

For example, in the table’s first row the AND gate’s inputs are \( \text{in}_1, \text{in}_2 = 0,0 \), its output is \( \text{out} = 0 \), and the cubic penalty function has a value of zero (as it does for all states except \( \text{in}_1, \text{in}_2, \text{out} = 0,0,1 \))

\[
P^* = -2\overline{m}_1\overline{m}_2out
\]
\[
= -2 \times 1 \times 1 \times 0
\]
\[
= 0
\]

The quadratic substitute has a value of zero regardless of the value of \( b \)

\[
P^q = -2 \times b \times (\overline{m}_1 + \overline{m}_2 + \text{out} - 2)
\]
\[
= -2 \times b \times (1 + 1 + 0 - 2)
\]
\[
= 0
\]

In the second row, the AND gate’s inputs are \( \text{in}_1, \text{in}_2 = 0,0 \), its output is \( \text{out} = 1 \), and the cubic penalty function’s value of \( -2 \) is activated:

\[
P^* = -2\overline{m}_1\overline{m}_2out
\]
\[
= -2 \times 1 \times 1 \times 1
\]
\[
= -2
\]

The value of the quadratic reformulation depends on the value of \( b \):

\[
P^q = -2b(\overline{m}_1 + \overline{m}_2 + \text{out} - 2)
\]
\[
P^q|_{b=0} = -2 \times 0 \times (1 + 1 + 1 - 2) = 0
\]
\[
P^q|_{b=1} = -2 \times 1 \times (1 + 1 + 1 - 2) = -2 \times (3 - 2) = -2
\]

When the objective function is minimized, \( b \) is set to \( b = 1 \) to minimize this term in the objective, resulting in a value of \( \min_b P^q = -2 \).
Constructing an Objective Function

The (normalized) penalty functions, described in the previous sections, for the three types of gates in the example circuit are:

\[ P_{\text{AND}} = x_1 x_2 - 2(x_1 + x_2)z + 3z - 2b(x_1 + x_2 + y_1 - 2) \]
\[ P_{\text{NOT}} = 2xz - x - z + 1 \]
\[ P_{\text{XOR}} = 2x_1 x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a \]

The circuit for this example is shown again in Figure 2.12 with the output formulas of all gates also shown. The CFD objective function is built by summing the penalty functions of all gates in the circuit:

\[ P_1 = x_1 x_2 - 2(x_1 + x_2)y_1 + 3y_1 - 2b_1(x_1 + x_2 + y_1 - 2) \]
\[ P_2 = y_1 x_3 - 2(y_1 + x_3)y_2 + 3y_2 - 2b_2(y_1 + x_3 + y_2 - 2) \]
\[ P_3 = y_1 x_3 - 2(y_1 + x_3)y_3 + 3y_3 - 2b_3(y_1 + x_3 + y_3 - 2) \]
\[ P_4 = 2y_2 x_3 - 2(y_2 + x_3)z_1 - 4(y_2 + x_3)a + 4az_1 + y_2 + x_3 + z_1 + 4a \]
\[ P_5 = 2y_3 z_2 - y_3 - z_2 + 1 \]

where \( P_i \) is the penalty function for gate \( i \) and ancillary variables are as follows:

- \( y_i \) is the output of AND gate \( i \).
- \( a \) is the ancillary variable of the generic penalty function for XOR, used here by the penalty function for XOR gate 4.
- \( b_i \) is the ancillary variable of the generic cubic-to-quartic reduction formula, \( b \), used here by the normalized penalty functions for AND gates 1 to 3.

A few minor adjustments can order these penalty functions in exactly the QUBO form used throughout this guide, \( E(a_i, b_{ij}; q_i) = \sum_i a_i q_i + \sum_{i<j} b_{ij}q_i q_j \).

1. Drop any freestanding constants. These do not affect the energy gap between valid and invalid configurations. In \( P_5 \), for example, the +1 term can be removed,

\[ P_5 = 2y_3 z_2 - y_3 - z_2 \]
2. Substitute \( \neg x \iff (1 - x) \). In \( P_1^N \), for example, use this equivalence to reformulate \( x_1 \) and \( x_2 \) introduced by \( P^* \),

\[
-2b_1(x_1 + x_2 + y_1 - 2) = -2b_1[(1 - x_1) + (1 - x_2) + y_1 - 2] \\
= -2b_1(-x_1 - x_2 + y_1) \\
= 2b_1x_1 + 2b_1x_2 - 2b_1y_1
\]

3. Map the complete set of variables to logical qubits\(^5\). This section, for example, orders the variables as \( q_i = \{x_1, x_2, x_3, z_1, z_2, y_1, y_2, y_3, a, b_1, b_2, b_3\} \)\(^{12} \).

The reordered penalty functions now have the familiar QUBO form:

\[
P_1 = 3y_1 + x_1x_2 - 2x_1y_1 + 2x_1b_1 - 2x_2y_1 + 2x_2b_1 - 2y_1b_1 \\
P_2 = 3y_2 + x_3y_1 - 2x_3y_2 + 2x_3b_2 - 2y_1y_2 + 2y_1b_2 - 2y_2b_2 \\
P_3 = 3y_3 + x_3y_1 - 2x_3y_3 + 2x_3b_3 - 2y_1y_3 + 2y_1b_3 - 2y_3b_3 \\
P_4 = x_3 + z_1 + y_2 + 4a - 2x_3z_1 + 2x_3y_2 - 4x_3a - 2z_1y_2 + 4z_1a - 4y_2a \\
P_5 = -z_2 - y_3 + 2z_2y_3
\]

where, for example, the linear coefficients of \( P_1 \) are \( a_6 = 3 \) (and \( a_i = 0 \) for \( i \neq 6 \) because \( 3y_1 \) is the only non-zero term of \( \sum_i a_i d_i \) and \( y_1 \) is \( 6^{th} \) in the mapping order used here for \( q_i \)) and the quadratic coefficients are \( b_{1,2} = 1, b_{1,6} = -2, \ldots \) and so on.

The objective function for this CFD problem is given by,

\[
E(a_i, b_{ij}; q_i) = P_1 + P_2 + P_3 + P_4 + P_5.
\]

To solve the circuit of this example, that is, to find a minimum-sized set of gates that, if faulty, explains the observation of incorrect outputs, this objective function must be minimized while holding constant the input values \( x_1, x_2, x_3 = 1, 1, 1 \) and incorrect output values \( z_1, z_2 = 1, 0 \).

### 2.2.2 Minor-Embedding the Problem

The Simple Map Coloring Problem example took advantage of the perfect alignment between the structures of problem and Chimera graph in the case of a four-color map problem. Typically these structures do not fortuitously align to such a degree but may do so to a lesser degree for problems with repetitive elements. For CFD problems, as it happens, graph minors of elementary binary operations are embeddable in single Chimera unit cells.

The constituents of the objective function created in the previous sections of this example derived from discrete binary operations plus, for the ANDs, a simple normalizing penalty function. These are all embeddable in single unit cells.

For example, the penalty function for NOT gate \( 5, P_5 \), can be represented as a fully connected \( K_2 \) graph that can be can be minor embedded onto two logical qubits \( z_2 \) \(\rightarrow\) \( q_1 \) and

---

\(^5\) This is similar to the logical qubits that represented colors (e.g., \( q_B \), the blue qubit) or regions (e.g., \( q_C^R \), the red qubit for British Columbia) in the Simple Map Coloring Problem section. For CFD, these logical qubits represent the values of inputs and outputs to the circuit’s gates. As before, a logical qubit, \( q_i \), might be minor embedded to a single physical qubit \( q_j \) on the QPU or to a chain of qubits.

\(^6\) This ordering of \( q_i \) is arbitrary; for a different ordering, a different \( a_i \) would be 3.
$y_3 \rightarrow q_2$ of a unit cell,\(^7\)

$$P_3 = -z_2 - y_3 + 2z_2y_3$$

$$E_{\text{NOT}}(a_1, b_{1,2}, q_1) = -q_1 - q_2 + 2q_1q_2$$

Represented graphically, this function has two variables, $q_i$, that correspond to two vertices, $V_i$, each with a bias, $a_i$, corresponding to the linear coefficient, $-1$, and between them an edge, $E_{12}$, with a coupling strength, $b_{1,2}$, that corresponds to the quadratic coefficient, $+2$.

To minor embed the NOT operation onto a QPU, configure two connected qubits with biases $a_1, a_2 = -1, -1$ and a coupling strength $b_{1,2} = 2$.\(^8\)

Figure 2.13 shows a minor embedding of a NOT gate into a unit cell of a D-Wave 2000Q QPU, in this case, the topmost left cell of the Chimera graph.

![Figure 2.13: A NOT gate minor embedded into the topmost left unit cell of a D-Wave 2000Q QPU.](image)

Logical qubits $q_1, q_2$, noted in bold, are minor embedded as physical qubits $q_0, q_4$, represented as a 0 and 4 inside a blue circle, respectively. Biases $a_1, a_2 = -1, -1$ and coupling strength $b_{1,2} = 2$ are also shown.

Similarly the normalized penalty function for AND gates can be minor embedded in different ways. Considering the usefulness of the generic AND gate, one approach might separate the generic AND functionality (one of the formulations provided in the Elementary Boolean Operations to QUBO section of the Reformulating a Problem chapter) from the normalizing penalty, $P_q$, used for this CFD problem. The generic penalty function for the AND operation can be represented as a fully connected $K_3$ graph; $P_q$ can be represented as a sparsely connected $K_4$ graph.

For example, returning to the penalty functions before they were reordered in the familiar QUBO form, and mapping variables to qubits, $x_3 \rightarrow q_1, y_1 \rightarrow q_2, y_2 \rightarrow q_3, b_2 \rightarrow q_4$, the

---

\(^7\) Note the similarity to the objective function developed in the Simple Map Coloring Problem example, in the two-color case, for a constraint that a region be assigned a single color only. There, the two logical qubits representing colors needed to have different values, which for binary variables means opposite 0/1 states. In other words, that constraint enforced a NOT relationship between the qubits.

\(^8\) As described in the Getting Started with the D-Wave System guide, the QPU has an allowed range of values for its biases and strengths. When configuring these values, the values calculated for the problem are scaled to fit those restrictions and other considerations (e.g., the values for variables of the objective function might also be scaled to accommodate the chaining of physical qubits as logical qubits). Consequently, the qubits for a NOT operation might be configured with values $a_1 = a_2 = -0.5; b_{1,2} = 1$ for example.
penalty function for AND gate 2 is given by,

\[
P_2 = y_1x_3 - 2(y_1 + x_3)y_2 + 3y_2 - 2b_2(y_1 + x_3 + y_2 - 2)
\]

The biases and coupling strengths are \(a_3, b_{1,2}, b_{1,3}, b_{1,3} = 3, 1, -2, -2\) for the generic AND part and \(b_{1,4}, b_{2,4}, b_{3,4} = 2, 2, -2\) for the normalizing part.

Figure 2.14 shows one option for minor embedding the penalty function representing an AND gate and its normalizing penalty function into a Chimera unit cell.

\[
E(a_i, b_{ij}, q_i) = q_2q_1 - 2q_2q_3 - 2q_1q_3 + 3q_3 + -2q_4[(1-q_2) + (1-q_1) + q_3 - 2]
\]

\[
E = 3q_3 + q_1q_2 - 2q_2q_3 - 2q_1q_3 + 2q_1q_4 + 2q_2q_4 - 2q_3q_4
\]

\[
E = 3q_3 + q_1q_2 - 2q_2q_3 - 2q_1q_3 + 2q_1q_4 + 2q_2q_4 - 2q_3q_4
\]

Figure 2.14: A normalized AND gate represented as a \(K_3\) graph for the generic AND part and its embedding into a Chimera unit cell, on the left, and the normalizing penalty function part and its minor embedding into a Chimera unit cell, on the right. The thick blue line represents a chain of qubits with the biases of the logical qubit divided among the physical qubits as described in the Simple Map Coloring Problem example (that example also explains how to formulate a constraint to create a qubit chain). The common qubits \(q_1, q_2, q_3\) must also be chained across the two unit cells. For readability, the black dots representing logical qubits are labeled with only the bias, \(a_i\), without the matching \(q_i\).

To embed the normalized AND functionality one must chain\(^9\) shared qubits \(q_1, q_2, q_3\) be-

\(9\) The Simple Map Coloring Problem example explains chaining for the similar case of cloned regions.
tween the two unit cells to ensure identical values.

An alternative way to embed the AND gate is to represent the entire normalized penalty as a fully connected $K_4$ graph, which can be minor embedded onto a single Chimera unit cell, as shown in Figure 2.15. Similar to the previous embedding of the two parts into two unit cells, the variables are mapped to four logical qubits, $x_3 \rightarrow q_1, y_1 \rightarrow q_2, y_2 \rightarrow q_3, b_2 \rightarrow q_4$. The objective function is merely a rearranged formulation of the previous one,

$$P_2 = 3y_2 + x_3y_1 - 2x_3y_2 + 2x_3b_2 - 2y_1y_2 + 2y_1b_2 - 2y_2b_2$$

$$E_{\text{AND}}(a_i, b_{ij}; q_i) = 3q_3 + q_1q_2 - 2q_1q_3 + 2q_1q_4 - 2q_2q_3 + 2q_2q_4 - 2q_3q_4$$

Figure 2.15: A normalized AND gate represented as a $K_4$ graph, on the left, and its minor embedding into a Chimera unit cell, on the right. The thick blue line represents a chain of qubits. For readability, these graphics are only partly labeled.

Analogously to these two ways of minor embedding AND gate 2, minor embedding a CFD problem can differently group the parts of its objective function. The intuitively simplest way might be to minor embed each gate in a unit cell and then chain shared variables. For example, the output of AND gate 2 feeds into the input of NOT gate 5 as $y_2$. Mapping variables to logical qubits,

NOT: $z_2 \rightarrow q_{1N}^N$, $y_3 \rightarrow q_{1N}^N$

AND: $x_3 \rightarrow q_{1A}^A$, $y_1 \rightarrow q_{2A}^A$, $y_3 \rightarrow q_{3A}^A$, $b_2 \rightarrow q_{4A}^A$

where superscripts $A$ and $N$ denote the AND and NOT parts, respectively (this is the only change to the following objective functions from their previous appearances in this section). The objective functions for the combined AND gate 2 and NOT gate 5 is given by,

$$P_5 = -z_2 - y_3 + 2z_2y_3$$

$$E_{\text{NOT}}(a_i^N, b_{ij}^N, q_i^N) = -q_1^N - q_2^N + 2q_1^Nq_2^N$$

The embedding needs to ensure that $y_3 = q_3^N = q_3^A$.

Figure 2.16 shows a possible minor embedding of AND gate 3 and NOT gate 5 with each in its own Chimera unit cell.
Figure 2.16: A minor embedding of AND gate 3 and NOT gate 5 with each in its own Chimera unit cell. AND gate 3 is minor embedded in the top leftmost unit cell of a D-Wave 2000Q QPU; NOT gate 5 is minor embedded in the neighboring unit cell. Qubits $q^N_2$ and $q^N_3$ are chained into a single logical qubit.

However, a less wasteful way to minor embed these gates is to combine them in a single unit cell, as shown in Figure 2.17.

2.2.3 Scalable, Tool-Assisted Formulation and Minor-Embedding

The manual formulations and minor embeddings of the preceding sections can be effective but also time-consuming for large problems. As described in the Software Environment and Tools chapter, D-Wave offers tools that can automate some of the solution steps to speed and scale up the process.

For example, penalty functions can be created using an SMT solver. Depending on the software used, deriving a penalty function for NOT gate 5 in this example may include commands like the following:

```python
import penaltymodel as pm
import penaltymodel_maxgap as maxgap
import networkx as nx
import dwave_networkx as dnx

FAULT_GAP = .5
NOT_5 = fault_gate({(-1, +1): 0, (+1, -1): 0}, FAULT_GAP)

G5 = dnx.chimera_graph(1)
G5 = nx.relabel_nodes(G5, {4: 'z2', 5: 'a3', 0: 'f5'})

spec = pm.Specification(G5, ['a3', 'z2', 'f5'], NOT_5, pm.SPIN)
pmodel = maxgap.get_penalty_model(spec)
```
Figure 2.17: A minor embedding of AND gate 3 and NOT gate 5 in a single Chimera unit cell, the top leftmost unit cell of a D-Wave 2000Q QPU. AND gate 3 is minor embedded in the lower 6 qubits; NOT gate 5 is minor embedded in the top two qubits. Qubits $q_N^2$ and $q_A^3$ are chained into a single logical qubit.

print('h:', pmodel.model.linear)
print('J:', pmodel.model.quadratic)
print('classical_gap:', pmodel.classical_gap)

In this particular code snippet, Ising is the chosen native formulation, the output of NOT gate 5, $z_2$, is assigned to node 4 of the graph minor to be embedded, and an ancillary variable representing the gate’s input (equivalent to $y_3$ in the previous sections), $a_3$, to node 5. Variable $f_5$, assigned to node 0, indicates whether the gate is faulty or functioning. The model is configured with valid states representing a NOT operation in spins: Boolean $in = 0 \rightarrow out = 1$ is $(−1, +1)$ and Boolean $in = 1 \rightarrow out = 0$ is $(+1, −1)$. Any other state is therefore a fault.

The output from this particular tool may look like the following:

```
h: {1: -0.265625, 2: -1.0, 3: -0.9, 6: -0.1875, 7: 0.903125, 'z2': -0.865625, 'f5': -0.740625, 'a3': -0.13125}
J: {(2, 7): -1.0, (2, 6): -0.0125, (3, 7): 0.103125, (2, 'z2'): 0.990625, (2, 'a3'): -0.996875, (3, 'a3'): 0.03125, (7, 'f5'): -1.0, (3, 'z2'): 1.0, (1, 6): -0.003125, (1, 'z2'): -0.2625, (3, 6): -0.996875, (1, 7): -1.0, (6, 'f5'): 1.0, ('z2', 'f5'): 0.1375, (1, 'a3'): 1.0}
classical_gap: 2.5
```

In the Ising model, $h_i$ is a linear coefficient representing the bias of a qubit and $J_{ij}$ is a quadratic coefficient representing a coupling between two qubits. The output $h : \{1 : −0.265625, 2 : −1.0\}$, means that qubit 1 has a bias of about $−0.265$, qubit 2 a bias of $−1$ and so on; the output $J : \{(2,7) : −1.0, (2, 6) : −0.0125\}$, means the coupler between qubits 2 and 7 has a strength of $−1$, the coupler between qubits 2 and 6 a strength of $−0.0125$ and so on.

The software tool used for this CFD example created for each gate an Ising penalty function suited for minor embedding in a single Chimera unit cell and optimized for a large energy
gap. The tool then used chains of qubits to interconnect qubits representing the problem’s variables across unit cells. Figure 2.18 shows this minor embedding in the Chimera of a D-Wave 2000Q QPU.

**Figure 2.18:** A minor embedding of this example’s CFD problem derived from a software tool. Each Boolean gate is minor embedded into a unit cell; the interconnecting qubit chains are highlighted in blue.
This chapter presents a catalog of problems that can be used as templates representing types of problems with similar solution steps on the D-Wave system.

Keep in mind that there are different ways to model a given problem; for example, a constraint satisfaction problem (CSP) can have various domains, variables, and constraints. Model selection can affect solution performance, so it may be useful to consider various approaches.

Quantum annealing processors naturally return low-energy solutions; some applications require the real minimum energy and others require good low-energy samples. This approach is best suited to solving discrete optimization problems and probabilistic sampling problems.

**Optimization problems.** In an optimization problem, we search for the best of many possible combinations. Optimization problems include scheduling challenges, such as “Should I ship this package on this truck or the next one?” or “What is the most efficient route a traveling salesperson should take to visit different cities?”

Physics can help solve these sorts of problems because we can frame them as energy minimization problems. A fundamental rule of physics is that everything tends to seek a minimum energy state. Objects slide down hills; hot things cool down over time. This behavior is also true in the world of quantum physics. Quantum annealing simply uses quantum physics to find low-energy states of a problem and therefore the optimal or near-optimal combination of elements.

**Sampling problems.** Sampling from many low-energy states and characterizing the shape of the energy landscape is useful for machine learning problems where we want to build a probabilistic model of reality. The samples give us information about the model state for a given set of parameters, which can then be used to improve the model.

Probabilistic models explicitly handle uncertainty by accounting for gaps in our knowledge and errors in data sources. Probability distributions represent the unobserved quantities in a model (including noise effects) and how they relate to the data. The distribution of the data is approximated based on a finite set of samples. The model infers from the observed data, and learning occurs as it transforms the prior distribution, defined before observing the data, into the posterior distribution, defined afterward. If the training process is successful, the learned distribution resembles the distribution that generated the data, allowing predictions to be made on unobserved data. For example, when training on the famous MNIST dataset of handwritten digits, such a model can generate images resembling handwritten digits that are consistent with the training set.

Sampling from energy-based distributions is a computationally intensive task that is an excellent match for the way that the D-Wave system solves problems; that is, by seeking low-energy states.
3.1 Discrete (Combinatorial) Optimization Problems and CSPs

Discrete optimization, also known as combinatorial optimization, is the optimization of an objective function defined over a set of discrete values such as Booleans.

The D-Wave system can be viewed as a hardware heuristic that minimizes Ising objective functions using a physically realized version of quantum annealing. The Reformulating a Problem chapter provides techniques to reformulate various problems as Ising (or QUBO) objective functions.

Further Information

- [Jue2016] discusses quantum annealing for Boolean satisfiability problems.
- [Koc2004] and related papers show that QUBOs are an effective representation for modeling and solving a variety of discrete optimization problems.
- [Bar1982] shows that Ising and QUBO problems are NP-hard.

3.1.1 Job-Shop Scheduling

The job-shop scheduling problem is to maximize priority or minimize schedule length (known as a makespan, the time interval between starting the first job and finishing the last) of multiple jobs done on several machines, where a job is an ordered sequence of tasks performed on particular machines, with constraints that a machine executes one task at a time and must complete started tasks.

Workflow Steps

1. State this problem as a CSP instance (the Two Illustrative Examples chapter gives introductory examples to this step).
2. Translate integer variables to binary (the Simple Map Coloring Problem section gives an example).
3. Reformulate this CSP instance as a QUBO (using the technique described in the CSP Reformulation with Penalty Functions section, for example, if you are scheduling for priority maximization).
4. If the problem is too large to fit on the QPU, decompose the problem (using methods such as those in the Decomposing a Large Problem chapter).
5. Follow the remaining steps of the general workflow of Table 1.1.

Further Information

- [Ven2015] describes an implementation of job-shop scheduling on the D-Wave system, which includes formulating the problem, translating to QUBO, and applying

1 Scheduling for makespan minimization needs either ancillary variables, also discussed in the Reformulating a Problem chapter, or a hybrid approach, such as discussed in the Decomposing a Large Problem chapter.
2 Typically, recasting from integer variables creates a large number of binary variables, many of which represent impossible combinations and so are unnecessary. For this type of problem, decomposition is likely needed.
variable reduction techniques. It also talks about direct embedding of local constraints.

### 3.1.2 Fault Diagnosis

Fault diagnosis attempts to quickly localize failures as soon as they are detected in systems such as sensor networks, process monitoring, and safety monitoring. Circuit fault diagnosis attempts to identify failed gates during manufacturing, under the assumption that gate failure is rare enough that the minimum number of gates failing is the most likely cause of the detected problem.

**Further Information**

- The *Simple Circuit Fault Diagnosis Problem* section in the *Two Illustrative Examples* chapter shows the steps of solving a circuit fault diagnosis problem on the D-Wave system.
- [Bia2016] discusses embedding fault diagnosis CSPs on the D-Wave system.
- [Bis2017] discusses a problem of diagnosing faults in an electrical power-distribution system.
- [Pap1976] discusses decomposing complex systems for the problem of generating tests for digital-faults detection.

### 3.1.3 Additional Problems

- **Protein Folding**
  Protein folding refers to the way protein chains structure themselves in the context of providing some biological function. Although their constituent amino acids enable multiple configurations, proteins rarely misfold (such proteins are a cause of disease) because the standard configuration has lower energy and so is more stable.
  - [Per2012] discusses using the D-Wave system to find the lowest-energy configuration for a folded protein.

- **Traffic-Flow Optimization**
  One form of the traffic-flow optimization problem is to minimize the travel time of a group of vehicles from their sources to destinations by minimizing congestion on the roads being used.
  - [Flo2017] describes work done by Volkswagen to map a traffic-flow optimization problem on the D-Wave system.

- **Factoring**
  The factoring problem is to decompose a number into its factors. There is no known method to quickly factor large integers—the complexity of this problem has made it the basis of public-key cryptography algorithms.
  - The *Factoring* section in the *Extended Examples* chapter is an example of factoring on the D-Wave system.
  - [Dwave3] discusses integer factoring in the context of using the D-Wave Anneal Offsets feature; see also the *Setting Anneal Offsets* section.
• Portfolio Optimization

Portfolio optimization is the problem of optimizing the allocation of a budget to a set of financial assets.


• Database Query Minimization (SAT Filters)

A satisfiability (SAT) filter is a small data structure that enables fast querying over a huge dataset by allowing for false positives (but not false negatives).

- [Dou2015] discusses uses of SAT filters with a quantum annealer.
- [Wea2014] describes the SAT filter.

3.2 Machine Learning Problems

Machine learning algorithms operate by constructing a model with parameters that can be learned from a large amount of example input so that the trained model can make predictions about unseen data. Most implementations to-date have been based on deterministic machine learning models such as feed-forward neural networks. The real world, however, is nondeterministic and filled with uncertainty. Probabilistic models explicitly handle this uncertainty by accounting for gaps in our knowledge and errors in data sources.

In probabilistic models, probability distributions represent the unobserved quantities in a model (including noise effects) and how they relate to the data. The distribution of the data is approximated based on a finite set of samples. The model infers from the observed data, and learning occurs as it transforms the prior distributions, defined before observing the data, into posterior distributions, defined afterward. If the training process is successful, the learned distribution resembles the actual distribution of the data to the extent that the model can make correct predictions about unseen situations—correctly interpreting a previously unseen handwritten digit, for example.

A Boltzmann distribution is a discrete energy-based distribution that defines probability, $p$, for each of the discrete states in a binary vector. For a set of $N$ binary random variables, $x$, the Boltzmann distribution defines a probability for each possible state that $x$ can take using

$$p(x) = \frac{1}{Z} \exp(-E(x; \theta)),$$  \hspace{1cm} (3.1)

where $E(x; \theta)$ is an energy function parameterized by $\theta$ and

$$Z = \sum_x \exp(-E(x; \theta))$$  \hspace{1cm} (3.2)

is the normalizing coefficient, also known as the partition function, that ensures that $p(x)$ sums to 1 over all the possible states of $x$; that is,

$$\sum_x p(x) = 1.$$  \hspace{1cm} (3.3)

$\beta$ is omitted from this equation because usually, in the context of machine learning, it is assumed to be 1.
Note that because of the negative sign for energy, $E$, the states with high probability correspond to states with low energy.

The energy function $E(x; \theta)$ can be represented as a QUBO: the linear coefficients bias the probability of individual binary variables in $x$ and the quadratic coefficients represent the correlation weights between the elements of $x$. The D-Wave architecture, which natively processes information through the Ising/QUBO models (linear coefficients are represented by qubit biases and quadratic coefficients by coupler strengths), can help discrete energy-based machine learning.

Further Information


### 3.2.1 Probabilistic Sampling: RBM

A Boltzmann machine is a stochastic recurrent neural network that defines a probability distribution over binary visible and hidden variables. A restricted Boltzmann machine has no intra-connections among the visible and hidden variables.

An RBM is shown in Figure 3.1.

![Figure 3.1: Bipartite neural net, with a layer of visible variables connected to a layer of hidden variables.](image)

Further Information

- [Sal2007] describes RBMs used to model tabular data, such as users’ ratings of movies.
- [Dwave2] provides an example of an RBM.
- [Hin2012] is a tutorial on training RBMs.
- [Ben2017] discusses sampling on the D-Wave system.

### 3.2.2 Energy-Based Models

Machine learning with energy-based models (EBMs) minimizes an objective function by lowering scalar energy for configurations of variables that best represent dependencies for probabilistic and nonprobabilistic models.
Workflow Steps

1. For an RBM as a generative model, for example, where the gradient needed to maximize log-likelihood of data is intractable (due to the partition function for the energy objective function), instead of using the standard Gibbs’s sampling, use samples from the D-Wave system. The training will have steps like these:
   (a) Initialize variables.
   (b) Teach visible nodes with training samples.
   (c) Sample from the D-Wave system.
   (d) Update and repeat as needed.

2. For embedding, it may be possible to map variables (of the reduced problem) directly to qubits (e.g., pixels to qubits for image inputs), as shown in [Ben2017].

3. Follow the remaining steps of the general workflow of Table 1.1.

Further Information

• [Lec2006] describes EBMs.
• [Dum2013] discusses implementing an RBM using physical computation.

3.2.3 Support Vector Machines

Support vector machines (SVM) find a hyperplane separating data into classes with maximized margin to each class; structured support vector machines (SSVM) assume structure in the output labels; for example, a beach in a picture increases the chance the picture is of a sunset.

Workflow Steps

1. Initialize \( \mathbf{w} = 0 \) where \( \mathbf{w} = [w_1, w_2, \ldots]^T \) are weights to be learned from the training data.

2. Repeat until \( \mathbf{w} \) converges:
   a. Minimize a regularized objective function for each training example, recording optimal solutions \( \hat{y}_i \), where \( \hat{y}_i \) is the maximizer for
      \[
      \max_{y \in \{0,1\}} \left\{ \Delta(y, y_i) + \langle \mathbf{w}, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right\}
      \]
   with features \( \Psi(x, y) \) defining a compatibility function \( f(x, y) = \langle \mathbf{w}, \Psi(x, y) \rangle \) on inputs \( x \) and labels \( y \).

Your objective function might look something like this:

\[
F(w) = \frac{1}{d} \sum_{i=1}^{d} \left[ \max_{y \in \{0,1\}} \left\{ \Delta(y, y_i) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right\} \right] + \lambda \frac{||w||^2}{2}
\]

where \( \lambda \) is a regularization term to prefer simple models (avoid over-fitting) and be minimized using a subgradient such as:

\[
\partial_w F(w) = \frac{1}{d} \sum_{i=1}^{d} [\Psi(x_i, \hat{y}) - \Psi(x_i, y_i)] + \lambda w.
\]
Submit this subgradient, formulated as a QUBO (see the Refor-
multating a Problem chapter), to the D-Wave system for minimiza-
tion.

b. Iterate over variables: calculate $\Delta w_j = \eta \cdot (y_i - \hat{y}_i)x_j$, where $\eta$ is the learning rate, and set the new weight $w_j \leftarrow w_j + \Delta w_j$.

Further Information
- [Dwave1] demonstrates how the D-Wave system can be applied within the context of structured prediction.
- [Boy2007] gives a concise introduction to subgradient methods.

3.3 Opportunities and Challenges

As noted in the Introduction chapter, quantum computing has the potential to help solve some of the most complex problems that organizations face. However, this cutting-edge technology still presents formidable challenges to potential users. The Technical Description of the D-Wave Quantum Processing Unit guide describes many of these in detail. They include, integrated control errors (ICE) such as neighbor interference by qubits, flux noise, quantization errors in configuring values of biases and coupler strengths, errors due to the finite bandwidth of the system’s I/O, and physical variations between qubits. Additionally, system performance may be affected by temperature, high-energy photons, and other factors.

This is typical of young, ground-breaking technologies. Persistence in overcoming such childhood problems to exploit such technologies has frequently been rewarded. Quantum computing poses some difficult challenges to adoption but offers great opportunities.
This chapter briefly presents several techniques for mapping a given problem to a formulation that the D-Wave system supports, Ising or QUBO, known as its native formulations. If a technique seems applicable to the given problem, see the Problem Reformulation section in the Derivations and Further Details chapter for more theoretical background and details, and the Problem Reformulation section in the Software Environment and Tools chapter for available software tools to automate reformulation. Detailed information is available in the literature and referenced papers to guide actual implementation.

Further Information

- [Bar1982] shows that Ising problems are hard.
- [Ber1999] discusses a broad tractable subclass of Ising problems in the context of rounding algorithms.
- [Kol2004] characterizes binary-variables energy functions that can be minimized by graph cuts.
- [Cou2009] and [Sch2009] discuss particular cases of tractable Ising problems.

4.1 Native Formulations: Ising and QUBO (and MIS)

The D-Wave QPU can be viewed as a heuristic that minimizes Ising objective functions using a physically realized version of quantum annealing. In Ising formulation, given a problem of $N$ variables corresponding to physical Ising spins, $s = [s_1, ..., s_N]$, and a configuration of $h_i$ local fields and $J_{i,j}$ couplings between spins, the QPU (in its solution spin values) minimizes the objective function

$$E(s|h, J) = \sum_{i=1}^{N} h_i s_i + \sum_{i<j}^{N} J_{i,j} s_i s_j$$

$s_i \in \{-1, +1\}$

It is straightforward to translate between Ising and QUBO representations. Similarly, it is simple to translate to these representations a problem formulated as either a maximum independent set (MIS), which seeks the largest subset of vertices of graph $G = (V, E)$, or a Max Cut, which seeks a subset of vertices that maximizes the number of edges between the subset and remaining vertices.
In QUBO formulation, $N$ binary variables are represented as an upper-diagonal matrix $Q$, where diagonal terms are the linear coefficients and the nonzero off-diagonal terms are the quadratic coefficients. The objective function to minimize is

$$E(x|Q) = \sum_{i \leq j} x_i Q_{i,j} x_j = \langle x, Qx \rangle \quad x_i \in \{0, 1\}$$

Use the simple arithmetic mapping, $s = 2x - 1$, where $1$ is the vector all of whose components are 1, to translate between these native formulations:

$$\langle x, Qx \rangle = \frac{1}{4} \langle 1, Q1 \rangle + \langle \sqrt{Q1/2}, s \rangle + \langle s, (Q/4) s \rangle$$

Thus,

$$E(x|Q) = \langle 1, Q1 \rangle + E(s|Q1/2, Q/4).$$

In MIS formulation, the objective function is:

$$\text{MIS: } E(x) = -\sum_{i \in V} x_i + \sum_{(i,j) \in E} M_{i,j} x_i x_j.$$ 

Clearly this formulation is similar to Ising with local fields $h_i = -1$ and only positive coupling strengths $J_{i,j}$ (represented by penalties $M_{i,j} > 1$).

### 4.1.1 Illustrative Example: Map-Coloring Objective in Ising Formulation

For an introduction to the technique, consider the following small example.

The Simple Map Coloring Problem section of the Two Illustrative Examples chapter develops an objective function, in QUBO formulation, for a simple two-color, single-region constraint,

$$E(a_i, b_i; q_i) = -q_B - q_G + 2q_B q_G,$$

where $q_B$ is a qubit representing blue and $q_G$ green.

In solving this objective function, the D-Wave system minimizes the equivalent Ising function. You can view this Ising formulation using the inverse of the $s = 2x - 1$ Ising-to-QUBO conversion. For this objective function, the variables are

$$x = \begin{bmatrix} q_B \\ q_G \end{bmatrix} \quad s = \begin{bmatrix} s_B \\ s_G \end{bmatrix}$$

The inverse conversion in scalar form, $q_i = \frac{s_i + 1}{2}$, is a simple mathematical manipulation of $s_i = 2x_i - 1$, where the QUBO variables in scalar form are $x_1, x_2 = q_B, q_G$. Substituting in the objective function for the simple map-coloring constraint yields

$$E(h_i, j_{i,j}; s_i) = -\frac{s_B + 1}{2} - \frac{s_G + 1}{2} + 2\frac{s_B + 1}{2} \frac{s_G + 1}{2} = \frac{1}{2} (s_B s_G - 1).$$
Table 4.1 shows the two formulations’ identical energy functions, with minimum levels for the valid states of a single selected color, either blue (0, 1 in a QUBO or −1, 1 in an Ising model) or green (1, 0 or 1, −1).

Table 4.1: Ising Equivalent of QUBO

<table>
<thead>
<tr>
<th>q_B, q_G</th>
<th>s_B, s_G</th>
<th>E(a_i, b_ij; q_i) = E(h_i, j_ij; s_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>−1, −1</td>
<td>0</td>
</tr>
<tr>
<td>0, 1</td>
<td>−1, 1</td>
<td>−1</td>
</tr>
<tr>
<td>1, 0</td>
<td>1, −1</td>
<td>−1</td>
</tr>
<tr>
<td>1, 1</td>
<td>1, 1</td>
<td>0</td>
</tr>
</tbody>
</table>

For example, the state \(q_B, q_G = 0, 1\) of the second row is represented by the QUBO penalty function,

\[
E(a_i, b_ij; q_i) = -q_B - q_G + 2q_Bq_G,
\]

\[
= -0 - 1 + 2 \times 0 \times 1
\]

\[
= -1
\]

while the equivalent state \(s_B, s_G = −1, 1\) of that same row is represented by the Ising penalty function,

\[
E(h_i, j_ij; s_i) = \frac{1}{2}(s_Bs_G - 1)
\]

\[
= \frac{1}{2}(-1 \times 1 - 1) = \frac{-2}{2}
\]

\[
= -1
\]

with the same resulting penalty.

Note that the constant \(-\frac{1}{2}\) in the Ising formulation above, which does not affect the relative energies, does not map into the formulation used to configure the D-Wave system, and is discarded when the logical qubits \(S_B, S_G\) are embedded as physical qubits (e.g., \(q_0, q_4\) for blue and \(q_1, q_5\) for green in the embedding used in the Simple Map Coloring Problem section).

**Software Tools**

See the Problem Reformulation section for a list of D-Wave software tools that automate processes such as translating between the native formulations.

**Further Information**


### 4.2 Weighted MAX-2-SAT to Ising/QUBO

Satisfiability (SAT) uses the following terminology:

- **Literal** is a Boolean variable such as \(x_i\) and \(\overline{x_i}\).
- **Clause** is a disjunction of literals such as \(x_i \lor \overline{x_j}\).
- **conjunctive normal form (CNF)** conjoins clauses by the AND operator; i.e., (clause 1) \(\land\) (clause 2).
The SAT problem is to decide whether the literals in its clauses can be assigned values that satisfy all the clauses; i.e., produce a value of 1. In CNF, the SAT is satisfied only if all its clauses are satisfied.

A 2-SAT has \( m \) clauses of 2 literals each. A MAX-2-SAT is the problem of assigning values that maximize the number of satisfied clauses (an optimization problem; a second version is to decide whether some specified number of clauses can be satisfied). Weighted MAX-SAT assigns each clause a positive weight so the violation of clause \( c_i \) incurs a cost \( w_i \). The problem is to maximize the weight of satisfied clauses.

To reformulate a MAX-2-SAT problem as a QUBO, note that (1) maximizing the weight of satisfied clauses is equivalent to minimizing the weight of unsatisfied clauses, and (2) by De Morgan’s laws, \( \overline{x_i \lor x_j} = \overline{x_i} \land \overline{x_j} \). This means that the general weighted MAX-2-SAT problem can be written as a posiform (in the context of machine learning problems, a polynomial expression that conjoins AND clauses with OR operations) that minimizes the weight of unsatisfied clauses:

\[
\min_x \sum_c w_c \overline{\ell_{c,1}} \overline{\ell_{c,2}}.
\]

### 4.2.1 Illustrative Example: Three-Variable MAX-2-SAT

For an introduction to the technique, consider the following small example.

A weighted MAX-2-SAT on three variables, \( x_1, x_2, x_3 \) with weights \( w_{c1} = 3, w_{c2} = 1, w_{c4} = 4 \), maximizes the number of satisfied clauses for the following example problem:

\[
\frac{(x_1 \lor x_2; 3)}{\text{clause 1}} \land \frac{(x_3; 1)}{\text{clause 2}} \land \frac{(x_3 \lor x_2; 4)}{\text{clause 3}}.
\]

Obtain the QUBO representation using the following steps:

1. Formulate as a posiform (using \( \overline{x_i \lor x_j} = \overline{x_i} \land \overline{x_j} \) and minimizing the weight of unsatisfied clauses as explained above):

\[
\min_x \left\{ 3(\overline{x_1} \land x_2) + 1(\overline{x_3}) + 4(x_3 \land \overline{x_2}) \right\} = \min_x \{3\overline{x_1}x_2 + \overline{x_3} + 4x_3\overline{x_2}\}.
\]

2. Write negated literals \( \overline{x_i} \) as \( (1 - x_i) \):

\[
\min_x \{3(1 - x_1)x_2 + (1 - x_3) + 4x_3(1 - x_2)\}
\]

\[
= \min_x \{3x_2 - 3x_1x_2 + 1 - x_3 + 4x_3 - 4x_2x_3\}
\]

\[
= \min_x \{1 + 3x_2 + 3x_3 - 3x_1x_2 - 4x_2x_3\}
\]

In this last expression, the example weighted MAX-2-SAT has been reformulated as a QUBO in scalar notation,

\[
E_{qubo}(a_i, b_{ij}; x_i) = \sum_i a_i x_i + \sum_{i<j} b_{ij} x_i x_j.
\]
where $a_1 = 0, a_2 = 3, a_3 = 3$ are the linear coefficients and $b_{1,2} = -3, b_{2,3} = -4$ the quadratic coefficients.

QUBOs may be written without any negative coefficients by using negative literals, expressing contributions such as

$$-3x_1 x_2$$

as either

$$-3 + 3\overline{x}_2 + 3x_1 x_2$$

or

$$-3 + 3\overline{x}_1 + 3x_1 x_2,$$

clearly showing the lower bound of $-3$.

**Software Tools**

See the *Problem Reformulation* section for a list of D-Wave software tools.

**Further Information**

- *SAT and Weighted MAX-2-SAT* expands on the satisfiability problem and derives this weighted MAX-2-SAT formulation.
- *[Bon2007]* generalizes the resolution rule for satisfiability to weighted MAX-SAT, which enables the simplification of some weighted MAX-2-SAT problems.

### 4.3 Non-Quadratic (Higher-Degree) Polynomials to Ising/QUBO

To reduce problems with interactions between more than pairs of variables to a QUBO, introduce and minimize over ancillary variables.

#### 4.3.1 Illustrative Example: Polynomial Reductions

For an introduction to the technique, consider the following small examples.

**Reduction by Minimum Selection**

The technique of reduction by minimum selection exploits the identity

$$xyz = \max_w \{w(x + y + z - 2)\}$$
where \( x, y, z \) are binary variables in some pseudo-binary function being minimized and \( w \) is the ancillary binary variable introduced to substitute quadratic terms for cubic terms.

Table 4.2 shows that the cubic and quadratic formulations are identical. In this table, column \( x, y, z \) is all possible states of the \( xyz \) term and column \( xyz \) is the corresponding values of the \( xyz \) term; column \( x + y + z - 2 \) is an intermediate calculation, the right side of the equality without the ancillary variable and before maximization, shown for clarity; and column \( \max_w \{ w(x + y + z - 2) \} \) is the final result of calculating the right side of the equality.

<table>
<thead>
<tr>
<th>( x, y, z )</th>
<th>( xyz )</th>
<th>( x + y + z - 2 )</th>
<th>( \max_w { w(x + y + z - 2) } )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 0</td>
<td>0</td>
<td>-2</td>
<td>0 ( w = 0 )</td>
</tr>
<tr>
<td>0, 0, 1</td>
<td>0</td>
<td>-1</td>
<td>0 ( w = 0 )</td>
</tr>
<tr>
<td>0, 1, 0</td>
<td>0</td>
<td>-1</td>
<td>0 ( w = 0 )</td>
</tr>
<tr>
<td>0, 1, 1</td>
<td>0</td>
<td>0</td>
<td>0 ( w = 0,1 )</td>
</tr>
<tr>
<td>1, 0, 0</td>
<td>0</td>
<td>-1</td>
<td>0 ( w = 0 )</td>
</tr>
<tr>
<td>1, 0, 1</td>
<td>0</td>
<td>0</td>
<td>0 ( w = 0,1 )</td>
</tr>
<tr>
<td>1, 1, 0</td>
<td>0</td>
<td>0</td>
<td>0 ( w = 0,1 )</td>
</tr>
<tr>
<td>1, 1, 1</td>
<td>1</td>
<td>1</td>
<td>1 ( w = 1 )</td>
</tr>
</tbody>
</table>

To reformulate a non-quadratic (higher-degree) polynomial to Ising/QUBO, substitute terms in the form of \( axyz \), where \( a \) is a real number, with one of the following quadratic terms:

\[
axyz = aw(x + y + z - 2) \quad a < 0
\]

\[
axyz = a \{ w(x + y + z - 1) + (xy + yz + zx) - (x + y + z) + 1 \} \quad a > 0
\]

**Reduction by Substitution**

The technique of reduction by substitution represents Boolean constraint \( z \leftrightarrow x_1 \land x_2 \) as a quadratic penalty function,

\[
P(x_1, x_2; z) = x_1x_2 - 2(x_1 + x_2)z + 3z.
\]

Table 4.3 shows that this function penalizes (adds a positive non-zero energy cost to) any proposed solutions that do not set the value of the ancillary variable, \( z \), identical to that of the constraint it replaces. In this table, column \( x_1, x_2 \) is all possible states of \( x_1, x_2 \) variables and column \( x_1x_2 \) is the corresponding values of the \( x_1 \land x_2 \) constraint; column \( P(z = x_1x_2) \) is the penalty for solutions that set the value of the ancillary variable, \( z \), identical to that of the constraint it replaces while column \( P(z \neq x_1x_2) \) is the penalty for incorrect solutions.

\[1 \] It is easy to verify, for example, that the equality for the negative coefficient, \( a < 0 \), holds. The term \( (x + y + z - 2) \) is positive and non-zero only for \( x, y, z = 1, 1, 1 \), so multiplying by negative \( a \) yields the only negative value of \( w(x + y + z - 2) \), which is preserved for \( w = 1 \). All other states of \( x, y, z \) result in \( (x + y + z - 2) \leq 0 \), so multiplied by \( a < 0 \) yield zero or positive values, which are minimized to zero by \( w = 0 \).
Table 4.3: Substitution Reduction.

<table>
<thead>
<tr>
<th>x_1, x_2</th>
<th>x_1 x_2</th>
<th>P(z = x_1 x_2)</th>
<th>P(z ≠ x_1 x_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0, 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1, 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1, 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

To reformulate a non-quadratic polynomial to Ising/QUBO, use this penalty function to reduce a triplet interaction to pairwise,

\[ x_1 x_2 x_3 = \min_z \{ z x_3 + MP(x_1, x_2; z) \}, \]

where \( M > 1 \) is a penalty weight.

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.

**Software Tools**

See the Problem Reformulation section for a list of D-Wave software tools.

**Further Information**

- **Posiforms and Higher-Order Polynomials** discusses posiforms and reducing higher-order polynomials to QUBOs.
- **Non-Quadratic (Higher-Order) Polynomials** discusses this technique in greater detail.
- **[Ish2011]** discusses reduction by minimum selection and by substitution and introduces an extension of the reduction by minimum selection.
- **[Bor2002]** discusses posiforms as a useful representation for pseudo-Boolean functions and shows that reformulating a problem of \( N \) terms requires at most \( N \) ancillary variables.
- **[Dat2014]** and **[Tan2015]** discuss tricks to quickly reduce higher-order polynomials to QUBOs in the context of factoring.

### 4.4 CSP to MIS

Given a set of constraints \( C = \{ C_a(x_k) \}_{a=1}^{|C|} \), this reformulation seeks a solution \( x \) that satisfies all constraints; i.e., \( \bigwedge_{C_a \in C} C_a(x) \) is true.

One standard approach of several that converts a CSP into a QUBO, defines the dual problem as follows:

Introduce a variable \( y_a \) for each constraint \( C_a \). The domain of \( y_a \) is the set of feasible solutions to \( C_a \). Coupling between dual variables arises from constraints \( C_a \) and \( C_\beta \) having common variables in their scopes. For example, if the scope of \( C_a \) is \( X_1, X_2 \), the scope of \( C_\beta \) is \( X_2, X_3 \) and the feasible sets are \{ (1, 2), (3, 1) \} and \{ (1, 4), (2, 1) \} respectively, then \( y_a \in \{ (1, 2), (3, 1) \} \) 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 $y_\alpha, y_\beta$ 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 $y_\alpha$, and we form edges between the nodes $y_\alpha, y_\beta$ if there is any conflicting setting of a shared variables. An example conflict graph is shown in Figure 4.1.

![Figure 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.](image)

If there is an independent set in the conflict graph of size $|C|$ then we have solved the problem, because MIS is easily represented as a QUBO as discussed in Native Formulations: Ising and QUBO (and MIS). We select each $y_\alpha$ in the independent set which defines values for all the $X$ variables in the scope of $C_\alpha$. The edge constraint ensures no two elements that are in conflict are in the independent set.

Software Tools

See the Problem Reformulation section for a list of D-Wave software tools.

Further Information

- The Constraints to Penalties section gives a background on CSP.
- [Bac2002] discusses advantages and disadvantages of dual variables in modeling CSPs.
- [Fra1989] addresses the equivalence of binary and non-binary CSPs and discusses two methods of converting between these formulations through dual and hidden transformations.

### 4.5 Soft CSP to MIS

Many CSPs used in applications are formulated with “hard” constraints—meaning constraints that cannot be broken in a viable solution—and “soft” constraints that incur a penalty if broken.
Hard and soft constraints can easily be represented by choosing two weights \( w_H \) (large magnitude) and \( w_S \) (small magnitude) to assign to the respective constraints. The resulting problem is to find an MIS of minimum weight.

The QUBO representation of weighted MIS for a graph \( G = (V, E) \) is simply
\[
\min \left\{ \sum_v w_v x_v + M \sum_{(v,v') \in E} x_v x_{v'} \right\}
\]
where \( w_v \) is the weight of vertex \( v \).

### Software Tools

See the *Problem Reformulation* section for a list of D-Wave software tools.

### Further Information

The *Constraints to Penalties* section gives a background on CSP.

## 4.6 CSP Reformulation with Penalty Functions

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

Even if a penalty can be derived through conversion to a linear equality or inequality, it may still be useful to apply the penalty-deriving machinery discussed in the *Nonlinear Constraints* section to construct a penalty with fewer ancillary variables.

For example, the constraint \( x_1 \lor x_2 \lor x_3 \) (which is useful for solving 3-SAT) can always be expressed as the inequality \( x_1 + x_2 + x_3 \geq 1 \), 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 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.\(^2\)

### Software Tools

See the *Problem Reformulation* section for a list of D-Wave software tools.

### Further Information

- The *CSP Conversion by Penalty Functions* section gives a background on using penalty functions to convert CSPs.
- the *Elementary Boolean Operations to QUBO* section provides a table of penalty functions for several Boolean operations.
- [Dec2003] discusses constraint programming as a declarative framework for modeling problems that require solutions to satisfy a number of constraints.
- [Ven2015b] discusses the use of penalties versus rewards for constraints.

\(^2\) In fact, this representation is commonly used to reduce 3-SAT to MAX-2-SAT.
4.6.1 Linear Equality Constraints

You can express $m$ equality constraints on $n$ Boolean variables, $x$, as $Cx = c$, where $C$ is an $m \times n$ matrix and $c$ an $m \times 1$ vector. Mapping these constraints to QUBO formulation, the penalty function $Cx - c = 0$ is simply $M||Cx - c||^2$. For $k$ independently weighted constraints, the penalty function in QUBO formulation is

$$P(x) = \sum_k m_k (\langle C_k, x \rangle - c_k)^2,$$

where $m_k^> > 0$ is the weight of the equality constraint $k$, and $C_k$ is row $k$ of $C$. This function is nonnegative and equal to zero for feasible $x$.

4.6.2 Illustrative Example: Map-Coloring with Linear Equality Constraints

For an introduction to the technique, consider the following small example.

The map-coloring problem is to assign a color to each region of a map, subject to the constraint that any two regions sharing a border must have different colors. The simple problem in this section is a snippet of the Simple Map Coloring Problem section for two regions, British Columbia, BC, and Alberta, AB, and two colors, blue and green, as shown in Figure 4.2.

**Figure 4.2**: A snippet of the Canadian map-coloring problem: two regions must select two different colors. Regions are represented by vertices and logical qubits, later to be mapped to physical qubits on a Chimera graph.

Table 4.4 shows the translation to binary, described previously in the Simple Map Coloring Problem section, for this simplified problem of $C = 2$ possible colors, where $q_B$ is a qubit representing blue and $q_G$ represents green.

**Table 4.4**: Translating Two Colors to Binary.

<table>
<thead>
<tr>
<th>Color</th>
<th>Naturals</th>
<th>Unary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>1</td>
<td>$q_B, q_G = 1, 0$</td>
</tr>
<tr>
<td>Green</td>
<td>2</td>
<td>$q_B, q_G = 0, 1$</td>
</tr>
</tbody>
</table>
For two regions, the logical qubits are denoted \( q^{BC}_{(B,G)} \) for British Columbia and \( q^{AB}_{(B,G)} \) for Alberta. To formulate a CSP with penalty functions, this example uses the following linear equality constraints:

1. In a region, only one color is selected.
   \[
   q^{BC}_B + q^{BC}_G = 1 \\
   q^{AB}_B + q^{AB}_G = 1
   \]

For the simplified problem of two regions and two colors, the constraint that adjacent regions not select the same color reduces to constraint (2) that each of the two colors be selected once.

2. Each color is selected once.
   \[
   q^{BC}_B + q^{AB}_B = 1 \\
   q^{BC}_G + q^{AB}_G = 1
   \]

This example’s \( m = 4 \) constraints (1) and (2) can be expressed in CCC formulation as

\[
\begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
q^{BC}_B \\
q^{BC}_G \\
q^{AB}_B \\
q^{AB}_G
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1 \\
1
\end{bmatrix}
\]

Mapping to \( M \parallel \text{CCC} - c \parallel^2 \) with all the weights set to \( m = k = 1 \) gives the penalty function as

\[
P_\parallel(x) = \sum_k m_k^\parallel (\langle C_k, x \rangle - c_k)^2
\]

\[
= (q^{BC}_B + q^{BC}_G - 1)^2 + (q^{AB}_B + q^{AB}_G - 1)^2 + (q^{BC}_B + q^{AB}_B - 1)^2 + (q^{BC}_G + q^{AB}_G - 1)^2
\]

Using the equality \( x^2 = x \) for binary variables on the square terms, the penalty function can be rearranged to the standard QUBO formulation as

\[
P_\parallel(x) = -2q^{BC}_B - 2q^{BC}_G - 2q^{AB}_B - 2q^{AB}_G + 2q^{BC}_B q^{BC}_G + 2q^{AB}_B q^{AB}_G + 2q^{BC}_B q^{AB}_B + 2q^{BC}_G q^{AB}_G + 4
\]

Figure 4.3 shows the penalty function for all 16 possible states of \((q^{BC}_B, q^{BC}_G, q^{AB}_B, q^{AB}_G)\) from \((0,0,0,0)\) to \((1,1,1,1)\).

The function adds a positive penalty to all possible states of the four qubits with the exception of two, where it has minima of zero: \(q^{BC}_B, q^{BC}_G, q^{AB}_B, q^{AB}_G = 0, 1, 1, 0\), which selects green for BC and blue for Alberta, and \(q^{BC}_B, q^{BC}_G, q^{AB}_B, q^{AB}_G = 1, 0, 0, 1\), which selects blue for BC and green for Alberta.

### 4.6.3 Linear Inequality Constraints

For the slightly more difficult inequality constraints of the form \( D x \leq d \), you can reduce the inequalities to equalities by introducing slack variables.

For example, the inequality constraint \( \langle D_i, x \rangle - d_i \leq 0 \), where \( D_i \) is row \( i \) of \( D \), can be rewritten as the equality \( \langle D_i, x \rangle - d_i + \xi_i = 0 \) by introducing a nonnegative slack variable \( \xi_i \geq 0 \).
Figure 4.3: Penalty function $P_\leq(x)$ showing minima when $(q^{BC}_B, q^{BC}_G, q^{AB}_B, q^{AB}_G)$ is equal to $(0, 1, 1, 0)$ and $(1, 0, 0, 1)$, the two states that meet all 4 constraints.

The slack variable may need to take a value as large as $d_i - \min_x(D_{ij}, 0)$. This slack quantity may be represented with $[d_i - \sum_j \min(D_{ij}, 0)]$ bits. We write $\xi_i = \langle 2, a_i \rangle$ where $a_i$ is the binary representation of $\xi_i$, and 2 is a vector whose elements are the powers of 2.

This binary representation enforces the requirement that $\xi_i \geq 0$. Inequality $i$ may be enforced with the penalty $(\langle D_i, x \rangle - d_i + \langle 2, a_i \rangle)^2$ and all constraints can be represented by summing individual inequality penalties with independent weights $m_i \leq i$ as

$$P_\leq(x, a) = \sum_i m_i \leq i (\langle D_i, x \rangle - d_i + \langle 2, a_i \rangle)^2,$$

where $a$ is the vector collecting all binary slack variables.

### 4.6.4 Nonlinear Constraints

The CSP Reformulation with Penalty Functions section shows how penalty functions for equality and inequality constraints may be constructed, with inequality constraints requiring ancillary variables.

Given sets of feasible $F$ and infeasible $\overline{F}$ configurations, and a requirement that $\min_a P(x, a) = o$ for $x \in F$ and $\min_a P(x, a) \geq o + 1$ for $x \in \overline{F}$ for some constant $o$ representing the objective value of feasible configurations, you can formulate a quadratic penalty as

$$P(x, a) = [x \ a] \begin{bmatrix} Q^{xx} & Q^{xa} \\ 0 & Q^{aa} \end{bmatrix} [x \ a]$$

with

$$Q^{xx} = \sum_i w_i^{xx} M_i^{xx}, \quad Q^{xa} = \sum_i w_i^{xa} M_i^{xa}, \quad Q^{aa} = \sum_i w_i^{aa} M_i^{aa}.$$
For Boolean indicator variables \( \{ \alpha_k(j) \} \) for each \( 1 \leq j \leq |F| \) and for each allowed configuration \( x_k \) you then have

\[
\sum_k \left\{ \langle \alpha_k(j) w_{x,a}^k, c_{x,a}^k(j) \rangle + \langle \alpha_k(j) w_{a,a}^k, c_{a,a}^k(j) \rangle \right\} \leq \langle w_{x,a}^k, c_{x,a}^k(j) \rangle + \langle w_{a,a}^k, c_{a,a}^k(j) \rangle \quad \forall k, j.
\]

To break the coupling of \( \alpha_k(j) \) to \( w_{x,a}^k \) and \( w_{a,a}^k \) and obtain a linear problem, introduce

\[
v_{x,a}^k(j) = \alpha_k(j) w_{x,a}^k \quad \text{and} \quad v_{a,a}^k(j) = \alpha_k(j) w_{a,a}^k.
\]

This requirement is enforced with

\[
\begin{align*}
v_{x,a}^k(j) & \leq \alpha_k(j) \\
v_{a,a}^k(j) & \leq \alpha_k(j) M \\
-v_{x,a}^k(j) & \leq \alpha_k(j) M \\
v_{a,a}^k(j) & \leq \alpha_k(j) M \\
\sum_k v_{x,a}^k(j) & = w_{x,a}^k \\
\sum_k v_{a,a}^k(j) & = w_{a,a}^k.
\end{align*}
\]

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.

The Formulating an Integer CSP as a QUBO Problem section gives an example of using these techniques on a problem.

### 4.7 CSP Reformulation with Inverse Verification

One approach to solving a constraint satisfaction problem, \( C(x) \equiv \bigwedge_{i=1}^{|C|} C_i(x_i) \), exploits the fact that NP-hard decision problems can be verified in polynomial time. If the CSP corresponding to \( C(x) \) is in NP, you can write the output \( z \iff C(x) \) as a Boolean circuit whose size is polynomial in the number of Boolean input variables. The Boolean circuit \( z \iff C(x) \) can be converted to an optimization objective using Boolean operations formulated as QUBO penalty functions, with each gate contributing the corresponding penalty function⎯see Elementary Boolean Operations to QUBO.

The resulting optimization objective—which also has a polynomial number of variables and at a feasible solution \( x^*, z^* = 1 \) evaluates to 0—may be run in reverse to infer 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.\(^3\) If you obtain a solution for which the objective evaluates to 0, you have a feasible solution to the CSP.

The Factoring section shows an example of using this technique to solve factoring problems as QUBOs. It shows that this formulation yields low requirements on both connectivity and precision constraints, making it suitable for the D-Wave system.

Fault Diagnosis may also benefit from this technique.

\(^3\) Alternatively, substitute \( z = 1 \) into the objective and eliminate \( z \) from the problem.
4.8 Elementary Boolean Operations to QUBO

As described in the CSP Reformulation with Penalty Functions section, constraints can be reformulated as penalties.

Table 4.5 shows several constraints useful in building Boolean circuits.

Table 4.5: QUBO Penalty Functions for Elementary Boolean Operations

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z \iff \neg x)</td>
<td>(2xz - x - z + 1)</td>
</tr>
<tr>
<td>(z \iff x_1 \lor x_2)</td>
<td>(x_1x_2 + (x_1 + x_2)(1 - 2z) + z)</td>
</tr>
<tr>
<td>(z \iff x_1 \land x_2)</td>
<td>(x_1x_2 - 2(x_1 + x_2)z + 3z)</td>
</tr>
<tr>
<td>(z \iff x_1 \oplus x_2)</td>
<td>(2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a)</td>
</tr>
</tbody>
</table>

4.8.1 Illustrative Example: Boolean OR as a Penalty

For an introduction to the technique, consider the following small example. Table 4.6 shows how a Boolean OR constraint,

\[z \iff x_1 \lor x_2,\]

can be reformulated using a penalty function. States that violate the constraint that \(z\) be equivalent to \(x_1 \lor x_2\) are penalized (a positive, nonzero energy cost is added) through function

\[x_1x_2 + (x_1 + x_2)(1 - 2z) + z.\]

In Table 4.6, column \(x_1, x_2\) is all possible states of the gate’s inputs; column \(x_1 \lor x_2\) is the corresponding output values of a functioning gate; column \(P(z = x_1 \lor x_2)\) is the value the penalty function adds to the energy of the objective function when the gate is functioning (zero, a functioning gate must not be penalized) while column \(P(z \neq x_1 \lor x_2)\) is the value the penalty function adds when the gate is malfunctioning (nonzero, the objective function must be penalized with a higher energy).

Table 4.6: Boolean OR Constraint as a Penalty.

<table>
<thead>
<tr>
<th>(x_1, x_2)</th>
<th>(x_1 \lor x_2)</th>
<th>(P(z = x_1 \lor x_2))</th>
<th>(P(z \neq x_1 \lor x_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0,1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1,0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1,1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

For example, for the state \(x_1, x_2 = 1, 1\) of the fourth row, when \(z = x_1 \lor x_2 = 1\),

\[x_1x_2 + (x_1 + x_2)(1 - 2z) + z = 1 + (1 + 1)(1 - 2) + 1 = 1 - 2 + 1 = 0,\]

\(4\) The XOR operator, \(\oplus\), requires the introduction of a single ancillary variable \(a\).
not penalizing the valid solution, whereas for $z \neq x_1 \lor x_2 = 0$,

$$x_1x_2 + (x_1 + x_2)(1 - 2z) + z = 1 + (1 + 1)(1 - 0) + 0 = 1 + 2 + 0 = 3,$$

adding an energy cost of 3 to the solution that violates the constraint.

**Further Information**

- The *Simple Circuit Fault Diagnosis Problem* section gives similar descriptions for the other gates.
- [Jue2016] provides numerous Boolean operations in the Ising model. These can be converted to QUBO penalty functions as shown in *Native Formulations: Ising and QUBO (and MIS)*.
This chapter briefly presents methods of decomposing a large problem into a part requiring no more than the number of qubits available on the QPU and, optionally for hybrid approaches, parts suited to conventional compute platforms. If a method seems applicable to the given problem, see the Problem Decomposition section for fuller descriptions and some background on conditioning. Detailed information is available in the literature and referenced papers to guide actual implementation.

5.1 Cutset Conditioning

Cutset conditioning conditions on a subset of variables (the “cutset”) of a graph, leaving the remaining network as a tree with complexity that is exponential in the cardinality (size) of the cutset.

Chose a subset $A$ of variables $A$ to fix throughout the algorithm so that the graph of $E(s_A, s_{A^c})$, where $\arg\min_{s_A} E(s_A, s_{A^c})$ is the optimal setting of $s_A$ for a given $s_{A^c}$, breaks into separate components that can be solved independently:

$$E(s_{A^c}) = E_{A^c}(s_{A^c}) + \sum_{\alpha} E_{\alpha}(s_{A^c})$$

where $s_A = \bigcup_{\alpha} s_{A_{\alpha}}$ and $s_{A_{\alpha}} \cap s_{A_{\alpha'}} = \emptyset$ for $\alpha \neq \alpha'$.

Choose a cutset such that each of the remaining $\min_{s_{A_{\alpha}}}$ problems is small enough to be solved on the D-Wave QPU.

To simplify outer optimization, make the number of conditioned variables as small as possible.

Software Tools

See the Problem Decomposition section for a list of D-Wave software tools.

Further Information

- The Cutset Conditioning section provides a more detailed description of this method.
- The Conditioning section gives background information on conditioning.
- [Dec1987] discusses this method for improving search performance in AI applications.
5.2 Branch-and-Bound Algorithms

Branch-and-bound algorithms progressively condition more variables to either $s_i = -1$ or $s_i = 1$, for spins, defining a split at node $s_i$. Further splits define a branching binary tree with leaves defining the $2^N$ configurations where all variables are assigned values. At each node a branch is pruned if no leaf node below it can contain the global optimum.

Software Tools

See the Problem Decomposition section for a list of D-Wave software tools.

Further Information

- The Branch-and-Bound Algorithms section provides a more detailed description of this method.
- The Conditioning section gives background information on conditioning.
- [Mar2007] considers ways to explore the search tree, including dynamic variable orderings and best-first orderings.

5.2.1 Best Completion Estimate

Branch-and-bound can benefit from using the D-Wave system to terminate searches higher in the tree.

Condition sufficient variables so the remaining can be optimized by the QPU. Instead of exploring deeper, call the QPU to estimate the best completion from that node. As the upper bound is minimized through subsequent QPU completions, this may in turn allow for future pruning.

**Note:** Since the QPU solution does not come with a proof of optimality, this algorithm may not return a global minimum.

5.2.2 Lower Bounds

The D-Wave system can also provide tight lower-bound functions at any node in the search tree.

Lagrangian relaxation finds these lower bounds by first dividing a node in the graph representing variable $s_i$ in two ($s_i^{(1)}$ and $s_i^{(2)}$) with constraint $s_i^{(1)} = s_i^{(2)}$. The original objective $E(s_i, s_{\cdot i})$ becomes $E'(s_i^{(1)}, s_i^{(2)}, s_{\cdot i})$, leaving the problem unchanged. With sufficient divided variables to decompose $E'$ into smaller independent problems the equality constraints are softened and treated approximately:
The Lagrangian for the constrained problem is
\[ L(s^{(1)}_i, s^{(2)}_i, s_{ij}; \lambda_i) = E'(s^{(1)}_i, s^{(2)}_i, s_{ij}) + \lambda_i(s^{(1)}_i - s^{(2)}_i). \]

Where \( \lambda_i \) is a multiplier for the equality constraint. Maximizing the dual function with respect to \( \lambda_i \),
\[ g(\lambda_i) = \min_{s^{(1)}_i, s^{(2)}_i} L(s^{(1)}_i, s^{(2)}_i, s_{ij}; \lambda_i), \]
provides the tightest possible lower bound.

Introduce enough divided variables to generate subproblems small enough to solve on the QPU and then optimize each subproblem’s dual function using a subgradient method to provide the tightest possible lower bounds.

**Further Information**

- [Boy2007] gives a concise introduction to subgradient methods.
- [Joh2007] gives an alternative to subgradient optimization, which examines a smooth approximation to dual function.

### 5.3 Large-Neighborhood Local Search Algorithms

Local search algorithms improve upon a candidate solution, \( s^t \), available at iteration \( t \) by searching for better solutions within some local neighborhood of \( s^t \).

Quantum annealing can be very simply combined with local search to allow the local search algorithm to explore much larger neighborhoods than the standard 1-bit-flip Hamming neighborhood.

For a problem of \( N \) variables and a neighborhood around configuration \( s^t \) of all states within Hamming distance \( d \) of \( s^t \), choose one of \( \binom{N}{d} \), and determine the best setting for these \( s_A \) variables given the fixed context of the conditioned variables \( s^t \setminus A \). Select \( d \) small enough to solve on the QPU. If no improvement is found within the chosen subset, select another.

**Software Tools**

See the Problem Decomposition section for a list of D-Wave software tools.

**Further Information**

- The Large-Neighborhood Local Search Algorithms section provides a more detailed description of this method.
- The Conditioning section gives background information on conditioning.
- [Ahu2000] describes the cyclic exchange neighborhood, a generalization of the two-exchange neighborhood algorithm.
- [Glo1990] is a tutorial on the tabu search algorithm.
- [Liu2005] presents promising results for even small neighborhoods of size \( d \leq 4 \).
5.4 Belief Propagation

The belief propagation algorithm passes messages between regions and variables that represent beliefs about the minimal energy conditional on each possible value of a variable. It can be used, for example, to calculate approximate, and in some cases exact, marginal probabilities in Bayes nets.

Further Information

- [Pea2008] describes the belief propagation algorithm.
- [Cou2009] is a tutorial on the subject.
- [Bia2014] discusses belief propagation in the context of decomposing CSPs into sub-problems small enough to be embedded onto the QPU.

5.5 Pruning Reformulation/Decomposition

Ancillary Variables

Ancillary variables introduced during reformulation and decomposing limit the size of problem you can run on the D-Wave system.

To minimize the number of introduced ancillary variables during a reduction-to-pairwise reformulation, select the pair of variables most common among the terms with more than pairwise interactions and represent the product of that pair with ancillary variable $z$. Repeat for any remaining terms of 3 or more variables. Note that subsequent iterations may involve products of ancillary variables if these ancillary pairs are the most common amongst the terms.

Software Tools

See the Problem Reformulation section for code to accomplish this reduction.

Further Information

[Bot2002] on general pseudo-Boolean optimization gives a reduction-to-pairwise algorithm, which can be modified to minimize the number ancillary variables.
The D-Wave QPU minimizes the energy of an Ising spin configuration whose pairwise interactions lie on the edges of an $M, N, L$ Chimera graph. To solve a given Ising spin problem with arbitrary pairwise interaction structure, you minor embed its graph into a Chimera graph by using qubits to represent missing edges.

Similar to Lagrangian relaxation, you map a given problem’s variable $s_i$ onto a set of qubits $\{q_i^{(1)}, \ldots, q_i^{(k)}\}$ while encoding equality constraint $q_i^{(j)} = q_i^{(j')}$ as an Ising penalty $-Mq_i^{(j)}q_i^{(j')}$ of weight $M > 0$.

The Two Illustrative Examples chapter provides several introductory examples of embeddings.

There are algorithms that can embed a problem of $N$ variables in at most $N^2$ qubits.

### 6.1 General Considerations

The following considerations apply to minor embedding.

- **Global embedding** models each constraint as an Ising model, adds all constraint models, and maps the aggregate onto the Chimera graph. Advantages of this method are that it typically requires fewer qubits and shorter chains of connected qubits for logical problem variables.

- **Locally structured embedding** models each constraint locally within a subgraph, places the local subgraphs within the Chimera graph, and then connects variables belonging to multiple local subgraphs. Advantages of this method, when the scopes of constraints are small, are typically that it is more scalable to large problems, requires less precision for parameters, and enforces qubit chains with lower coupling strengths.

- When mapping a problem’s variable to qubits chains, the penalties for equality constraints should be (1) large enough so low-energy configurations do not violate these constraints and (2) the smallest weight that enforces the constraints while enabling precise problem statement (on $h$ and $J$) and efficient exploration of the search space. An effective procedure incrementally updates weights until the equality constraints are satisfied. See the Overcoming Imprecisions of Qubit Biases and Coupling Strengths section.

- Logically identical embeddings can have different higher energy spectrums, thus different performances.

- Qubits used as couplers cannot be used as problem variables, reducing the effective size of the Ising problem that can be solved. Use embeddings that waste the fewest
Software Tools

- See the Minor Embedding section for a list of D-Wave software tools.
- The Open-Source Environment section references D-Wave’s open source embedding tool on GitHub.

Further Information

- The Embedding section discusses the above more fully.
- [Bia2016] compares global and local methods of embedding in the context of CSPs and discusses a rip-up and replace method.
- [Cai2014] gives a practical heuristic for finding graph minors.
- [Boo2016] discusses clique minor generation.
- [Jue2016] discusses using FPGA-like routing to embed.
- [Ven2015b] discusses effects of embedding the Sherrington-Kirkpatrick problem.
- [Rie2014] studies embedding and parameter setting, and their effects on problem solving in the context of optimization problems.

6.2 Chain Management

The following considerations and recommendations apply to chains.

- Prefer short chains to long chains.
- Prefer uniform chain lengths to uneven chains.
- Balance chain strength and problem range. Estimate chain strength and set just slightly above the minimum threshold needed, using strategies for auto-adjusting these chains.

6.3 Embedding Complete Graphs

The largest complete (all \( V \) vertices interconnected) graph \( K_V \) that is a minor of a \( M \times N \times L \) graph has \( V = 1 + L \min(M, N) \) vertices.

For example, 65 vertices is the theoretical maximum on a D-Wave 2000Q QPU, which supports a C16 Chimera graph (a 16x16 matrix of 8-qubit unit cells for up to\(^1\) \( 2^{MNL} = 2 \times 16 \times 16 \times 4 = 2048 \) qubits).

Table 6.1 shows some example embeddings of complete graphs on a D-Wave 2000Q QPU.

\(^1\) The yield of a working graph is typically less than 100%.
Table 6.1: Example Complete Embeddings on a D-Wave 2000Q QPU.

<table>
<thead>
<tr>
<th>Complete Graph</th>
<th>Minor of Chimera Working Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_5$</td>
<td>$1 \times 1 \times 4$ (single cell)</td>
</tr>
<tr>
<td>$K_9$</td>
<td>$2 \times 2 \times 4$ (four cells)</td>
</tr>
<tr>
<td>$K_{65}$</td>
<td>$16 \times 16 \times 4$ (all cells)</td>
</tr>
</tbody>
</table>

The $K_9$ embedded graph minor for a $2 \times 2 \times 4$ graph is shown in Figure 6.1.

Figure 6.1: Embedding a $K_9$ graph into $2 \times 2 \times 4$; A and B indicate two different blocks of 4 variables. Equivalently labeled qubits are connected by edge penalties.

Specific connectivity requirements may have problem-specific embeddings that make more effective use of qubits than using Chimera as the complete graph for immediate embedding of all Ising problems defined on $V$ variables.

**Further Information**
- The *Embedding Complete Graphs* section discusses embedding complete graphs in detail and provides examples.

### 6.4 Finding Better Embeddings

Where the exact form of a problem is malleable, for example in some machine learning problems, simplify by assuming that each node in the original graph is mapped to a single node in Chimera. Fit the problem into Chimera by maximizing the total magnitude of $J_{i,j}$ mapped to Chimera edges; that is,

$$M^* = \arg \max_M \sum_{i \neq j} \sum_{(q,q')} |J_{i,j}|m_{i,q}m_{j,q'},$$
where $M$ is subject to the mapping constraints: $\sum_q m_{i,q} = 1$, for all problem variables $i$, and $\sum_i m_{i,q} \leq 1$, for all Chimera nodes $q$.

For linear time embedding, apply a greedy heuristic to approximately maximize the objective.

Further Information
- The Finding Better Embeddings section provides more details on the above.

6.5 Reusing Embeddings

For problems that vary only the biases and weights on a fixed graph, you can set a good embedding once before submitting the problem to the QPU. There is no need to recompute the embedding (a time-consuming operation) for every submission.

6.6 Pre-embedding Local Constraint Structures

The structure of some problems contains repeated elements; for example, the multiple AND gates, half adders, and full adders in the Factoring example. Such problems may benefit from being embedded with repeating block structures for the common elements, with connectivity then added as needed.

Further Information
- [Ada2015] describes embedding an RBM on the D-Wave system by mapping the visible nodes to chains of vertical qubits and hidden nodes to chains of horizontal qubits.

6.7 Virtual Graphs

The D-Wave virtual graph feature provides tools and interactive examples that simplify the process of minor-embedding by enabling you to more easily create, optimize, use, and reuse an embedding for a given working graph. When you submit an embedding and specify a chain strength using these tools, they automatically calibrate the qubits in a chain to compensate for the effects of biases that may be introduced as a result of strong couplings.

Further Information
- D-Wave provides an open-source tool, dwave_virtual_graph on GitHub
- [Dwave6] is a white paper describing measured performance improvements from using virtual graphs.
When preparing your problem for submission, you must consider the analog nature of the D-Wave system.

7.1 Overcoming Imprecisions of Qubit Biases and Coupling Strengths

Ising problems with high-precision parameters \((h_i)\) and \((J_{ij})\) present a challenge for quantum annealers due to the finite precision available on \(h\) and \(J\). A problem may have lowest energy states that are sensitive to small variations in \(h\) or \(J\) while also requiring a large range of \(h\) or \(J\) values or high penalty values to enforce constraints on chains of qubits.

These are typically quantitative optimization problems rather than problems of a purely combinatorial nature (such as finding a subgraph with certain properties), where the number and connectivity of the qubits is more important than the weights, and problems for which near-optimal solutions are unacceptable. The solution’s quality depends on slight differences, at low-energy regions of the solution space, of the problem Hamiltonian as delivered to the QPU from its specification.

This section looks at some ways to deal with these imprecisions.

Further Information

- [Pud2014] and [Pud2015] discuss quantum error correction.
- [Kin2014] discusses preprocessing more robust problem Hamiltonians on the D-Wave system.
- Technical Description of the D-Wave Quantum Processing Unit describes integrated control errors (ICE), measurement, and effects; for example, quantization of digital to analog converters.

7.1.1 Limiting Parameter Ranges through Embedding

You can improve results by minimizing the range of on-QPU \(J\) or \(h\) values through embeddings.

For example, if a problem variable \(s_i\), which has the largest parameter value \(h_i\), is represented by qubits \(q_i^1, q_i^2, \) and \(q_i^3\) having the same value in any feasible solution, \(h_i\) can be shared across the three qubits; i.e., \(h_i s_i \rightarrow (h_i/3)(q_i^1 + q_i^2 + q_i^3)\), reducing \(h_i\) by a factor of 3.
In a similar way, coupling parameters $J_{ij}$ may also be shared. In any embedding there may be multiple edges between chains of qubits representing problem variables. You can enhance precision (at the cost of using extra qubits) by sharing the edge weight across these edges.

**Software Tools**
The client libraries include code to split parameter weight across qubits and edges.

### 7.1.2 Limiting Parameter Ranges by Simplifying the Problem

In problems with interaction $h_is_i$, where $h_i > 0$ is much larger than all other problem parameters, it is likely that in low-energy states, $s_i = -1$ (2$h$ lower in energy than $s_i = +1$). Generally, you may be able to identify, in polynomial time, a subset of variables that always take the same value in the ground state. You can then eliminate such variables from the problem.

Consider preprocessing problems to determine whether certain variable values can be inferred. There is little overhead in attempting to simplify every problem before sending it to the QPU.

**Software Tools**
Code in the client libraries preprocesses problems efficiently.

### 7.2 Adjusting the Problem Scale

In general, use the full range of $h$ and $J$ values available for the QPU when submitting a problem.

To avoid explicitly dealing with these ranges, use the auto_scale solver parameter\(^1\) to automatically scale the problem to make maximum use of the available ranges.

Note that using $J < -0.8$ may reduce the benefit of quantum annealing for some problem types. While no definitive model captures this effect, it might be useful to scale the overall problem energy down with a prefactor of 0.8. If you do so, disable the auto_scale parameter.

### 7.3 Using Spin-Reversal (Gauge) Transforms

Coupling $J_{ij}$ adds a small bias to qubits $i$ and $j$ due to leakage. This can become significant for chained qubits. Additionally, qubits are biased to some small degree in one direction or another.

Applying a spin-reversal transform can improve results by reducing the impact of possible analog and systematic errors. A spin-reversal transform does not alter the Ising problem;

\(^1\) In the D-Wave system, a solver is simply a resource that runs a problem. Some solvers interface to the QPU; others leverage CPU and GPU resources.
the transform simply amounts to reinterpreting spin up as spin down, and visa-versa, for a particular spin.

- Changing too few spins leaves most errors unchanged, and therefore has little effect.
- Changing too many spins means that most couplers connect spins that are both transformed, thus $J_{ij}$ does not change sign. As a result, some systematic errors associated with the couplers are unaffected.

Specify the number of spin-reversal transforms using the `num_spin_reversal_transforms` solver parameter. Note that increasing parameter increases the total run time of the problem.

Further Information

- Spin-Reversal Transform discusses the above more fully.
- Technical Description of the D-Wave Quantum Processing Unit describes ICE in the D-Wave system.

### 7.4 Changing the Global Annealing Schedule

**Note:** Variations on the global anneal schedule are not supported on D-Wave 2X and earlier systems. For D-Wave 2000Q systems, the maximum number of points in a schedule is system-dependent.

Some types of research may benefit from the introduction of a pause or a quench at some point in the anneal schedule. A pause dwells for some time at a particular anneal fraction; a quench abruptly terminates the anneal within a few hundred nanoseconds of the point specified. This degree of control over the global annealing schedule allows you to study the quantum annealing algorithm in more detail.

#### 7.4.1 Pause and Quench

For example, a pause can be a useful diagnostic tool for instances with a small perturbative anticrossing. While pauses early or late in the anneal have no effect, a pause near the expected perturbative anticrossing produces a large increase in the ground-state success rate.

If a quench is fast compared to problem dynamics, then the distribution of states returned by the quench can differ significantly from that returned by the standard annealing schedule. The probability of obtaining ground state samples depends on when in the anneal the quench occurs, with later quenches more likely to obtain samples from the ground state.

Supply the scheduling points using the `anneal_schedule` solver parameter.

Further Information

- [Dic2013] discusses the anticrossing example.
- Technical Description of the D-Wave Quantum Processing Unit describes varying the anneal schedule.
7.4.2 Reverse Anneal

Reverse annealing enables the use of quantum annealing as a hybrid component in local search algorithms to refine classical states. It also gives users additional control over and insight into the quantum annealing process in the D-Wave system. Examples of using this feature include Quantum Boltzmann sampling, tunneling rate measurements, and relaxation rate measurements.

Further Information

[Dwave5] is a white paper on reverse annealing.

7.5 Setting Anneal Offsets

Note: Anneal offsets are not supported on D-Wave 2X and earlier systems. Before using this feature, query the solver properties using SAPI calls to determine whether it is supported and, if so, to obtain the available tuning ranges per qubit.

Anneal offsets may improve results for problems in which the qubits have irregular dynamics for some easily determined reason; for example, if a qubit’s final value does not affect the energy of the classical state, you can advance it (with a positive offset) to reduce quantum bias in the system;

Anneal offsets can also be useful in embedded problems with varying chain length: longer chains may freeze out earlier than shorter ones, which means that at an intermediate point in the anneal, some variables act as fixed constants while others remain undecided. If, however, you advance the anneal of the qubits in the shorter chains, they freeze out earlier than they otherwise would. The correct offset will synchronize the annealing trajectory of the shorter chains with that of the longer ones.

If you decide that offsetting anneal paths might improve results for a problem, your next task is to determine the optimal value for the qubits you want to offset. As a general rule, if a qubit is expected to be subject to a strong effective field relative to other qubits, delay its anneal with a negative offset. The ideal offset magnitudes are likely to be the subject of trial and error, but expect that the appropriate offsets for two different qubits in the same problem to be within 0.2 normalized offset units of each other.

Supply the array of offsets for the qubits in the system using the `anneal_offsets` solver parameter with a length equal to the `num_qubits` property.

Further Information

- *Technical Description of the D-Wave Quantum Processing Unit* describes anneal offsets.
- [Kin2016] discusses the use of anneal offsets.
- The D-Wave website provides an interactive feature example demonstrating anneal offsets for problems of different sizes.
- *Using Anneal Offset* shows how to run a problem with anneal offset.
7.6 Controlling the Energy Gap

There are strategies for increasing the gap between ground and excited states during the anneal. For example, different choices of constraints when reformulating a CSP as a QUBO affect the gap. Consider also the differences between maximizing the gap versus creating a uniform gap.

Further Information

- [Bia2014] discusses constructing a penalty function for a given constraint with the largest possible gap, subject to bounds on the supported $h$s and $J$s.
- [Pud2014] and [Pud2015] discuss error suppression techniques using auxiliary qubits and the energy gap.

7.7 Reducing Neighbor Interactions

The dynamic range of $h$ and $J$ values may be limited by ICE. Instead of finding low-energy states to an optimization problem defined by $h$ and $J$, the QPU solves a slightly altered problem that can be modeled as

$$E^{\delta}_{\text{ising}}(\mathbf{s}) = \sum_{i=1}^{N} (h_i + \delta h_i) s_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} (J_{i,j} + \delta J_{i,j}) s_i s_j,$$

where the ICE errors $\delta h_i$ and $\delta J_{i,j}$ depend on $h_i$ and on the values of all incident couplers $J_{i,j}$ and neighbors $h_j$, as well as their incident couplers $J_{j,k}$ and next neighbors $h_k$. For example, if a given problem is specified by $(h_1 = 1, h_2 = 1, J_{1,2} = -1)$, the QPU might actually solve the problem $(h_1 = 1.01, h_2 = 0.99, J_{1,2} = -1.01)$.

Changing a single parameter in the problem might change all three error terms, altering the problem in different ways.

Further Information

[HAR2010] discusses how applied $h$ bias leaks from spin $i$ to its neighboring spins.

7.8 Avoiding I/O Errors

With a readout fidelity of 99%, we recommend at least two read-anneal cycles per problem to expose possible outliers. The most cost-effective use of QPU time (that is, to best amortize the fixed cost of programming the QPU against total cost), is found by taking enough reads to at least equal the programming time.

To guard against occasional programming errors, we recommend splitting a given problem into two separate QPU requests to trigger a reprogramming cycle and reduce the probability of error to less than 1%.
7.9 Post-Processing

When submitting a problem to the QPU, you have an option to apply low treewidth graph algorithms to improve solutions. You can choose:

- No postprocessing (the default).
- Optimization postprocessing to obtain a set of samples with lowest energy on a graph $G$.
- Sampling postprocessing to obtain a set of samples that correspond to a target Boltzmann distribution with inverse temperature $\beta$ defined on the logical graph $G$.

Postprocessing does not affect throughput—it works in tandem with the system and adds little overhead time (except for the last batch). However, if you have fewer than about 50 reads, postprocessing will likely not batch the outputs, so you do get overhead.

Further Information

- Postprocessing Methods on D-Wave Systems describes postprocessing.
8.1 Working with the D-Wave System

Fig. 8.1 shows a simplified diagram of the computation workflow on a D-Wave system. Each problem consists of a single input together with parameters specifying, for example, the number of reads and whether postprocessing (PP) should be invoked. A problem is sent across a network to the SAPI server and joins a queue. Queued problems are assigned to workers, which can run in parallel. A worker prepares the problem for the QPU and optionally for postprocessing, sends the problem to the QPU queue, receives results, and bundles the results with additional information about the problem (such as runtime information).

The total time for a problem to pass through the system is the service time. Note that service time includes wait time in two queues, which can vary according to the number of problems and workers active at any moment. The QPU handles problems one at a time; the time for one problem to move through the QPU is the QPU access time.

Note the following considerations and recommendations on system overhead and network latency:

- Timing anomalies (spikes) are present in any system.
- Submit jobs at time of least contention; for example, small jobs versus large jobs.

Further Information

- The Measuring Computation Time on D-Wave Systems guide describes the computation process, in the context of system timing, on D-Wave quantum computers.
8.2 Problem Reformulation

SAPI client libraries provide utilities for problem reformulation; for example,

- `ising_to_qubo()` converts an Ising problem to a QUBO equivalent.
- `qubo_to_ising()` converts a QUBO problem to an Ising equivalent.
- `reduce_degree()` reduces the polynomial degree of an array of terms to two by introducing ancillary variables.
- `make_quadratic()` reduces the polynomial degree of a vector of numbers to two by introducing ancillary variables.

**Further Information**

- See the C, MATLAB, and Python developer guides, which describe the APIs used for programming a D-Wave system.

8.3 Problem Decomposition

SAPI client libraries provide utilities for problem decomposition; for example,

- `fix_variables()` fixes variables for solving Ising problems.
- `QSage` (accessed through the `solve_qsage()` utility) is a hybrid extension of the tabu search algorithm that uses quantum annealing to heuristically minimize objective functions that might not be mathematically expressible.

**Further Information**

- See the C, MATLAB, and Python developer guides, which describe the APIs used for programming a D-Wave system.

8.4 Minor Embedding

SAPI client libraries provide utilities for minor embedding; for example,

- `find_embedding()` heuristic to find an embedding of a QUBO/Ising problem in a graph.
- `embed_problem()` embeds an Ising problem into a graph.
- `unembed_answer()` unembeds answer to the original problem from solver.
- `get_chimera_adjacency()` returns the pairs of interconnected qubits on a Chimera graph.
- `get_hardware_adjacency()` returns the pairs of interconnected qubits on a solver.
- `linear_index_to_chimera()` and `chimera_to_linear_index()` convert between Chimera indexing schemes.

**Further Information**

- See the C, MATLAB, and Python developer guides, which describe the APIs used for
programming a D-Wave system.

8.5 Submitting Problems to the QPU

SAPI client libraries provide utilities for managing connections to solvers and solving problems; for example,

- `RemoteConnection()` and `local_connection()` connect to remote or local solvers.
- `solve_ising()` and `async_solve_ising()` solve an Ising problem synchronously or asynchronously.
- `solve_qubo` and `async_solve_qubo()` solve a QUBO problem synchronously or asynchronously.

**Further Information**

- See the C, MATLAB, and Python developer guides, which describe the APIs used for programming a D-Wave system.

8.6 Sampling Service

D-Wave simplifies the training of probabilistic models by providing access to high performance classical and quantum resources through the D-Wave Sampling Service—a set of Python libraries that make it easy to sample and perform the required Monte Carlo calculations. The service provides access to statistics from undirected graphical models based on samples from Boltzmann and quantum Boltzmann distributions.

8.7 Open-Source Environment

The D-Wave GitHub site is located here: https://github.com/dwavesystems.

D-Wave GitHub provides an open-source development environment for utilities that many users can find helpful in solving the types of problems relevant to the D-Wave system; for example,

- `minorminer()`, `chimera-embedding()`, and `dwave_virtual_graph()` tools for minor embedding.
- `dimod()` provides a shared API for Binary Quadratic Program (BQP) samplers.
- `dwave_micro_client()` is a minimal implementation of the REST interface used to communicate with D-Wave Solver API (SAPI) servers.
- `dwave_neal()` is a C++ simulated annealing sampler for general Ising model graphs.
- `dwave_sapi_dimod()` is a dimod wrapper for D-Wave’s SAPI Client Library.
- `dw_sa_cha()` simulated annealing solvers.
- `Qbsolv()` is a decomposing solver that finds a minimum value of large QUBO problems.
• D-Wave NetworkX extension to NetworkX, a Python language package for exploration and analysis of networks and network algorithms.

For the latest software in this environment, see the documentation on the site.
9.1 Formulating an Integer CSP as a QUBO Problem

As an example application of reformulating a given problem as a QUBO with the techniques of Reformulating a Problem, this example shows how any integer program may be converted to QUBO form. An example of an integer CSP is the map coloring problem described in Simple Map Coloring Problem.

Consider the integer program:

$$\min_y \langle c, y \rangle \quad \text{subject to: } Ay = a, \ By \leq b, \ y_i \in \{0, D_i - 1\}$$

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 $y_i = \langle 2, x_i \rangle$ where $2 = [2^d, \ldots, 2, 1]$ is the vector of powers of two and $x_i = [x_i, d_i, \ldots, x_i, 1, x_i, 0]$ 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 $\langle 2, x_i \rangle \leq D_i - 1$.

Thus, the original integer program may easily be converted to:

$$\min_x \langle c, x \rangle \quad \text{subject to: } Ax = a, \ Bx \leq b, \ x_i \in \{0, 1\}$$

where the new $B$ and $b$ have the additional inequality constraints appended, and where $c$ has been suitably modified. In this form, it remains only to represent the equality and inequality constraints within the unconstrained QUBO. These are easily translated to quadratic penalties.

Thus, the binary program can be written as the QUBO objective:

$$\arg\min_{x, \xi} \left\{ \langle c, x \rangle + \sum_k m_k \left( \langle A_k, x \rangle - a_k \right)^2 + \sum_i m_i^\le \left( \langle B_i, x \rangle - b_i + \langle 2, \xi_i \rangle \right)^2 \right\}$$

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 $m^=$ and $m^\le$. 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 $m = [m^=, m^\le]$ is some setting of the penalty weights and $x(m), \xi(m)$ are the optimal
variable settings for that choice of \( m \), then we update the penalty weights in proportion to the constraint violations:

\[
m_k := m_k + \alpha \left| \langle A_k, x(m) \rangle - a_k \right| + \alpha \left| \langle B_k, x(m) \rangle - b_k + \xi(m) \right|,
\]

where \( \alpha > 0 \) determines the rate at which penalty weights are accumulated.

**Further information**

- [Bac2002] compares performance of two transformations from non-binary to binary constraints used in CSPs: the dual transformation and the hidden (variable) transformation.

### 9.2 Factoring

The factoring problem is to decompose a number into a product of other numbers (factors) that give the original when multiplied together. [Bur2002] shows that factoring can be used as a generator of difficult optimization problems.

This section shows a complete example of solving a problem with QUBOs, using techniques that make best use of the D-Wave system. As 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 example factors \( p \), an integer of \( \ell \) bits, as a product of a pair of integers \( a \) and \( b \), all represented in binary\(^1\). Factors \( a \) and \( b \) are assumed to be \( \ell_a \) and \( \ell_b \) bits respectively so that \( \ell = \ell_a + \ell_b \) or \( \ell = \ell_a + \ell_b - 1 \).

One approach is to cast this task as an optimization problem for which a reasonable objective is to minimize the difference:

\[
\arg\min_{a,b} \left( p - \langle a, 2 \rangle \langle 2, b \rangle \right)^2.
\]

This reformulation involves quartic interactions in the optimization variables so ancillary variables will need to be introduced to reduce the objective to quadratic as described in *Non-Quadratic (Higher-Degree) Polynomials to Ising/QUBO*. More problematically, there are a large number of interactions between pairs of variables and the magnitudes of these interactions varies from 1 to \( 2^{\ell_a} + 2^{\ell_b} - 4 \). Thus, both connectivity and precision requirements are severe in this formulation of the problem as described in *Overcoming Imprecisions of Qubit Biases and Coupling Strengths*.

To ameliorate these difficulties, we adapt the “inverse verification” idea to this factoring context. Representing factoring as a constraint satisfaction problem, as discussed in *CSP Reformulation with Inverse Verification*, greatly reduces precision requirements, and Boolean circuits (see *Elementary Boolean Operations to QUBO*) enable us to tuneably define the connectivity between optimization variables.

---

\(^1\) That is, \( p = \langle 2, p \rangle \), \( a = \langle 2, a \rangle \), and \( b = \langle 2, b \rangle \) where \( 2 \) is a vector of the powers of two of appropriate length.
### 9.2.1 Multiplication Circuits as QUBOs

For factorization, the verification circuit is formed from the circuit which multiplies the inputs \( a \) and \( b \). We then fix the output of the circuit to the known bit string \( p \), and minimize the energy of the penalty formulation of the circuit to find \( a \) and \( b \). The multiplication circuit is built from AND gates, half adders and full adders. The AND gate which realizes \( t = x_1 \land x_2 \) is given by the penalty \( P(\land)(x_1, x_2; t) = x_1 x_2 - 2(x_1 + x_2)t + 3t \). 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 shown in Figure 9.1.

![Figure 9.1: Multiplication of two 4-bit integers.](image)

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 \(a_i b_j\) is the logical AND of the two bits. We introduce the ancillary variables \(t_{i,j} \equiv a_i \land b_j, s_l^b\) 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 Figure 9.1. As an example, the following QUBO objective

\[
\begin{align*}
X & \quad a_3 & a_2 & a_1 & a_0 \\
& b_3 & b_2 & b_1 & b_0 \\
\quad a_3 b_3 & a_3 b_2 & a_3 b_1 & a_3 b_0 \\
& a_2 b_3 & a_2 b_2 & a_2 b_1 & a_2 b_0 \\
& \quad a_1 b_3 & a_1 b_2 & a_1 b_1 & a_1 b_0 \\
& \quad \quad a_0 b_3 & a_0 b_2 & a_0 b_1 & a_0 b_0 \\
\end{align*}
\]

\[= \quad p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 \]
realizes the multiplication of two 4-bit integers \( a \) and \( b \) into their product \( p \).

- \( p_0: P_{\lambda}(a_0, b_0; p_0) \)
- \( p_1: P_{\lambda}(a_1, b_0; t_{1,0}) + P_{\lambda}(a_0, b_1; t_{0,1}) + P_{2A}(t_{1,0}, t_{0,1}; p_1, c_1^1) \)
- \( p_2: P_{\lambda}(a_0, b_2; t_{0,2}) + P_{\lambda}(a_1, b_1; t_{1,1}) + P_{\lambda}(a_2, b_0; t_{2,0}) + P_{2A}(t_{2,0}, t_{1,1}; s_2^1, c_2^1) + P_{3A}(t_{0,2}, s_2^1, c_1^1; p_2, c_2^2) \)
- \( p_3: P_{\lambda}(a_0, b_3; t_{0,3}) + P_{\lambda}(a_1, b_2; t_{1,2}) + P_{\lambda}(a_2, b_1; t_{2,1}) + P_{\lambda}(a_3, b_0; t_{3,0}) + P_{2A}(t_{2,1}, t_{3,0}; s_3^1, c_3^1) + P_{3A}(t_{1,2}, s_3^1, c_2^1; s_2^2, c_2^3; p_3, c_3^3) \)
- \( p_4: P_{\lambda}(a_1, b_3; t_{1,3}) + P_{\lambda}(a_2, b_2; t_{2,2}) + P_{\lambda}(a_3, b_1; t_{3,1}) + P_{2A}(t_{2,2}, t_{3,1}; s_3^1, c_4^1) + P_{3A}(t_{1,3}, s_3^1, c_2^1; s_2^2, c_2^3; p_4, c_4^3) \)
- \( p_5: P_{\lambda}(a_2, b_3; t_{2,3}) + P_{\lambda}(a_3, b_2; t_{3,2}) + P_{3A}(t_{2,3}, t_{3,2}; c_4^1; s_3^1; c_5^2) + P_{3A}(s_3^2, c_4^2; c_5^3; p_5, c_5^3) \)
- \( p_6: P_{\lambda}(a_3, b_3; t_{3,3}) + P_{3A}(t_{3,3}; c_4^2, c_5^3; p_6, p_7) \)
- \( p_7: \) 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 **Figure 9.2**.

**Figure 9.2:** Graphical representation of the multiplication circuit for two 4-bit integers. The structure reflects Figure 9.1. See text for the explicit QUBO form of the circuit.

Yellow ellipses represent ∧ 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 \( a \) and \( b \) variables as well as the intermediate sum and carry variables. As is visible from **Figure 9.2** the connectivity between variables is much sparser than the naive \( (p - (a, 2) \cdot (2, b))^2 \) objective. Connectivity is defined by the sparse connections between small completely connected subgraphs of size 3, 4, and 5.
9.2.2 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 \( \tilde{P}_{2A}(x_1^1, x_2^1, x_2^2; s, c) \) which realizes the constraint \( s + 2c = x_1^1 x_2^2 + x_2 \) 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 CSP Reformulation with Penalty Functions, we rapidly find that the following penalties realize these constraints:

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

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

Note that no additional ancillary variables are needed to represent these non-linear constraints. Thus, we can eliminate all the \( t_{ij} \) variables representing the \( \land \) 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 Figure 9.3.

![Figure 9.3: The constraint network representing the multiplication of two 8-bit numbers.](image)

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

Further, circuit improvements are possible. The resulting QUBO can be a difficult optimization problem to solve, with a great many local minima.
This chapter provides background information and more details on the techniques presented in previous chapters for the purpose of helping users unfamiliar with those to assess whether one or more might pertain to a given problem. Consult with the literature and with D-Wave for additional information when implementing.

10.1 Problem Reformulation

10.1.1 SAT and Weighted MAX-2-SAT

[Coo1971] and [Lev1973] identified satisfiability as the first NP-complete problem. The conjunctive normal form of SAT is a decision problem that 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. A literal is a variable or a negation of a variable. For example, the clause $x_1 \lor \overline{x}_2$ is satisfied if either $x_1 = true = 1$ or $x_2 = false = 0$.

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 called MAX-SAT (MAXimize the number of satisfied clauses) 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.

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. $\{(x_1 \lor \overline{x}_2; 3), (x_3; 1), (\overline{x}_3 \lor x_2; 4)\}$ represents a problem of three variables consisting of 3 clauses having weights 3, 1, and 4 respectively.

If all clauses of a weighted MAX-SAT 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 prob-

---

1 An overline on a Boolean variable indicates its negation, i.e. $\overline{x} = 1 - x$. 

---
How might we represent clauses as terms in a QUBO? A weighted clause \((x_1 \lor \bar{x}_2; 3)\) is unsatisfied if \(x_1 \land x_2\) is true, and this conjunction is equivalent to the product of Boolean variables \(x_1 x_2\). If the clause is unsatisfied then a cost of 3 is incurred, thus the penalty representing the clause is \(3x_1 x_2\). The weighted MAX-2-SAT problem \(\{(x_1 \lor x_2; 3), (x_3; 1), (\bar{x}_3 \lor x_2; 4)\}\) is then easily represented as \(3x_1 x_2 + x_3 + 4x_3 \bar{x}_2\). 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:

\[
\min_x \sum_c w_c \ell_{c,1} \ell_{c,2}
\]

where \(\ell_{c,1}\) and \(\ell_{c,2}\) are the two literals of clause \(c\), and \(w_c\) is its weight.

### 10.1.2 Posiforms and Higher-Order Polynomials

Posiforms are a useful representation for pseudo-Boolean functions\(^2\) (functions which map bit-strings to real values). General posiforms may have terms consisting of products of more than 2 literals. For a problem of \(N\) terms the required number of ancillary variables in polynomial is \(N\). 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:

\[
\min_x \left\{ \sum_{v \in V} x_v + M \sum_{(v,v') \in E} x_v x_{v'} \right\}
\]

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:

\[
\min_x \left\{ \sum_{v \in V} \bar{x}_v + M \sum_{(v,v') \in E} x_v x_{v'} \right\}
\]

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 \(x_{ij}\) 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:

\[
\sum_{(c_1, c_2, c_3) \in S} x_{c_1,c_2} x_{c_1,c_3} x_{c_2,c_3}
\]

\(^2\) See \([Bor2002]\).
where $S$ is the set of all subsets of size 3 of $\{1, 2, \ldots, N\}$. 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 the general weighted MAX-2-SAT problem written as the posiform in SAT and Weighted MAX-2-SAT is:

$$\min_x \sum_c w_c \ell_c \overline{t}_{c,1} \overline{t}_{c,2} \overline{t}_{c,3}.$$  

### Non-Quadratic (Higher-Order) Polynomials

For many problems, the natural model includes interactions between more than pairs of variables. How can such problems be reduced to the pairwise QUBO representation?

One approach is to introduce and minimize over ancillary variables. The constraint $z \iff x_1 \land x_2$, 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:

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

This penalty function enforces the constraint because it attains the minimal value of 0 at the 4 $(x_1, x_2, z)$ combinations satisfying $z \iff x_1 \land x_2$, while all other combination of values have $P > 0$. We use this result to reduce a triplet interaction to pairwise interaction as follows:

$$x_1 x_2 x_3 = \min_z \{zx_3 + MP(x_1, x_2; z)\}$$  

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.

As extra ancillary variables limit the problem size addressable by the D-Wave system, it is desirable to minimize the number of introduced variables. The reduction-to-pairwise algorithm of [Bor2002] is easily modified to minimize the number ancillary variables. Select the pair of variables that is most common amongst all terms with more than pairwise interactions, and replace it with ancillary variable $z$ in all those terms. Repeat until no more terms of 3 or more variables remain. Note that subsequent iterations may involve products of ancillary variables if these ancillary pairs are the most common amongst the terms.

### 10.1.3 Constraints to Penalties

Most applications involve associated constraints on variables which restrict bitstrings to certain feasible configurations. The QPU cannot natively impose constraints; instead you introduce energy penalties that penalize infeasible configurations. Penalty functions are

---

3 The same penalty function written in Ising terms of -1/1 variables is $P(s_1, s_2; s_z) = 3 + s_1 s_2 - 2(s_1 + s_2) s_z - s_1 - s_2 + 2s_z$.

4 The penalty weight may be taken to be $M = 1$.  

---
also a source of higher order interactions. This section considers approaches to solving optimization problems with constraints.

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 \( C_\alpha(\mathbf{x}_\alpha) \) defined over a subset of variables \( \mathbf{x}_\alpha \) defines the set of feasible and infeasible combinations of \( \mathbf{x}_\alpha \). The constraint \( C_\alpha \) may be be viewed as a predicate which evaluates to true on feasible configurations and to false on infeasible configurations. For example, 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 variables. We indicate the set of constraints as \( \mathcal{C} = \{ C_\alpha(\mathbf{x}_\alpha) \}_{\alpha=1}^{|\mathcal{C}|} \) and we seek a solution \( \mathbf{x} \) that satisfies all constraints, i.e. \( \bigwedge_{C_\alpha \in \mathcal{C}} C_\alpha(\mathbf{x}) \) is true.

### 10.1.4 CSP Conversion by Penalty Functions

The CSP \( \rightarrow \text{MIS} \rightarrow \text{QUBO} \) 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.

This section follows the same approach as was introduced to reduce higher-order problems to quadratic in *Posiforms and Higher-Order Polynomials*. For each constraint \( C_\alpha(\mathbf{x}_\alpha) \) we introduce a quadratic penalty function \( P_\alpha(\mathbf{x}_\alpha, \mathbf{a}_\alpha) \). The penalty function is defined so that:

\[
\begin{align*}
\min_{\mathbf{a}_\alpha} P_\alpha(\mathbf{x}_\alpha, \mathbf{a}_\alpha) &= o \quad \text{for all feasible } \mathbf{x}_\alpha \\
\min_{\mathbf{a}_\alpha} P_\alpha(\mathbf{x}_\alpha, \mathbf{a}_\alpha) &\geq o + 1 \quad \text{for all infeasible } \mathbf{x}_\alpha.
\end{align*}
\]

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

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

gives a feasible solution \( \mathbf{x}^*, \mathbf{a}^* \) satisfying all constraints and having objective value \( P(\mathbf{x}^*, \mathbf{a}^*) = o \) if the original CSP has a feasible solution \( \mathbf{x}^* \). If the original CSP is infeasible then \( P(\mathbf{x}^*, \mathbf{a}^*) \) will be at least \( o + 1 \).

The utility of this approach depends upon being able to conveniently represent common constraints \( C_\alpha \) as QUBO penalties. There are techniques to automatically construct QUBO penalty function from specifications of \( C_\alpha \).

### 10.1.5 Nonlinear Constraints

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 configura-
tions. Let $F$ represent a set of feasible configurations, and $\overline{F}$ represent a set of infeasible configurations. We require $\min_{\alpha} P(x, a) = 0$ for $x \in F$ and $\min_{\alpha} P(x, a) \geq o + 1$ for $x \in \overline{F}$ for some constant $o$. Since $P$ must be quadratic, we write the penalty as:

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

with

$$Q^{x,x} = \sum_i w_i^{x,x} M_i^{x,x}, \quad Q^{x,a} = \sum_i w_i^{x,a} M_i^{x,a}, \quad Q^{a,a} = \sum_i w_i^{a,a} M_i^{a,a}.$$  

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

$$\min_{w,a} \|w\|_1$$

subject to:

$$\langle w^{x,x}, c^{x,x}(j) \rangle + \min_{k} \left\{ \langle w^{x,a}, c^{x,a}_k(j) \rangle + \langle w^{a,a}, c^{a,a}_k \rangle \right\} = o \quad \forall j \in F$$

$$\langle w^{x,x}, c^{x,x}(\overline{j}) \rangle + \min_{k} \left\{ \langle w^{x,a}, c^{x,a}_k(\overline{j}) \rangle + \langle w^{a,a}, c^{a,a}_k \rangle \right\} \geq o + 1 \quad \forall \overline{j} \in \overline{F}, \forall k.$$  

where $k$ ranges over the allowed configurations $a_k$ of $a$, and the $c$ vectors have components indexed by $j$ given by:

$$[c^{x,x}(j)]_i = x_i^T M_i^{x,x} x_i, \quad [c^{x,a}_k(j)]_i = x_i^T M_i^{x,a} a_k, \quad [c^{a,a}_k]_i = a_k^T M_i^{a,a} a_k.$$  

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:

$$\min_{w,a} \|w\|_1$$

subject to:

$$\langle w^{x,x}, c^{x,x}(j) \rangle + \min_{k} \left\{ \langle w^{x,a}, c^{x,a}_k(j) \rangle + \langle w^{a,a}, c^{a,a}_k \rangle \right\} = o \quad \forall j \in F$$

$$\langle w^{x,x}, c^{x,x}(\overline{j}) \rangle + \langle w^{x,a}, c^{x,a}_k(\overline{j}) \rangle + \langle w^{a,a}, c^{a,a}_k \rangle \geq o + 1 \quad \forall \overline{j} \in \overline{F}, \forall k.$$  

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 $1 \leq j \leq |F|$ and for each allowed configuration $a_k$. Define the vector $a(j)$ 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:

$$\sum_k \left\{ \langle \alpha_k(j) w^{x,a}, c^{x,a}_k(j) \rangle + \langle \alpha_k(j) w^{a,a}, c^{a,a}_k \rangle \right\} \leq \langle w^{x,a}, c^{x,a}_k(j) \rangle + \langle w^{a,a}, c^{a,a}_k \rangle \quad \forall k, j.$$  

Unfortunately, this constraint couples $\alpha_k(j)$ to $w^{x,a}$ and $w^{a,a}$.

To break this coupling and obtain a linear problem we introduce $v_k^{x,a}(j) = \alpha_k(j) w^{x,a}$ and $v_k^{a,a}(j) = \alpha_k(j) w^{a,a}$. This

\[\text{CPLEX and other commercial grade optimization software can solve problems with quadratic constraints so it may be used to directly address this formulation.}\]
requirement is enforced with:

\[
\begin{align*}
    v_k^{x,a}(j) & \leq a_k(j) \\
    v_k^{a,a}(j) & \leq a_k(j)M \\
    -v_k^{x,a}(j) & \leq a_k(j)M \\
    -v_k^{a,a}(j) & \leq a_k(j)M \\
    \sum_k v_k^{x,a}(j) & = w^{x,a} \\
    \sum_k v_k^{a,a}(j) & = w^{a,a}.
\end{align*}
\]

The first two of the constraints in each column requires that \(v_k^{x,a}(j) = 0\) and \(v_k^{a,a}(j) = 0\) if \(a_k(j) = 0\). The penalty weight \(M\) should be chosen to be larger than the largest absolute value of the optimal values of \(w^{x,a}\) and \(w^{a,a}\) 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.

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 \(x_1 \lor x_2 \lor x_3\) (which is useful if we’re solving 3-SAT) can always be expressed as the inequality \(x_1 + x_2 + x_3 \geq 1\), 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

\[
-3 + (\overline{x}_1 + \overline{x}_2 + \overline{x}_3)(1 + a) + a + x_1x_2 + x_2x_3 + x_3x_1
\]

represents the same constraint using a single ancillary variable\(^6\).

## 10.2 Problem Decomposition

To solve a problem with more variables than the available number of qubits, break the problem into subproblems, solve the subproblems, and then reconstruct an answer to the original problem from the subproblem solutions. Divide-and-conquer and dynamic programming algorithms have a rich history in computer science for problems of this type and can be adapted to map problems onto the D-Wave QPU.

### 10.2.1 Conditioning

The approaches here rely on conditioning (or assigning) a subset of variables to particular values and solving the problem that remains. If \(A\) indicates a set of variables that are unconditioned, and \(\backslash A\) indicates the variables conditioned to certain values, then an Ising objective \(E(s)\) can be minimized in two stages:

\[
\min_{s} E(s) = \min_{s_{A}, s_{\backslash A}} E(s_{A}, s_{\backslash A}) = \min_{s_{\backslash A}} E(s_{\backslash A})
\]

\(^6\) In fact, it is this representation which is commonly used to reduce 3-SAT to MAX-2-SAT.
where

\[ \mathcal{E}(s_{\setminus A}) = \min_{s_{\setminus A}} E(s_A, s_{\setminus A}) = E(s_{\setminus A}^*(s_{\setminus A}), s_{\setminus A}). \]

Here, \( s_{\setminus A}^*(s_{\setminus A}) = \arg\min_{s_{\setminus A}} E(s_A, s_{\setminus A}) \) is the optimal setting of \( s_A \) for a given \( s_{\setminus A} \). Conditioning makes the \( \min s_A \) problem smaller and simpler than the original problem. The graph of this conditioned problem over the vertices \( s_A \) of \( E(s_A, s_{\setminus A}) \) is formed from the graph of the original problem by removing all conditioned variables and all edges attached to a conditioned variable. Of course, you still must identify the lowest energy setting of the conditioned variables.

The outer loop optimization determining the conditioned variable settings is the \( \min s_{\setminus A} \mathcal{E}(s_{\setminus A}) \) problem. The graph for the \( \mathcal{E}(s_{\setminus A}) \) problem has connections between variables in \( s_{\setminus A} \) which have been induced through the elimination (minimization) of the \( s_A \) variables; that is, through the dependence on \( s_{\setminus A}^*(s_{\setminus A}) \). In summary, conditioning casts the \( \min E(s) \) problem in terms of the smaller problem \( \min s_{\setminus A} \mathcal{E}(s_{\setminus A}) \) where \( \mathcal{E}(s_{\setminus A}) \) can be evaluated at any \( s_{\setminus A} \) using the D-Wave system. The outer optimization over \( s_{\setminus A} \) is carried out using conventional software algorithms.

In certain situations, techniques that infer values for the unconditioned variables based on side constraints or bounding information can further reduce the smaller \( \min s_{\setminus A} \) subproblem. 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 use no inference. In contrast, complete search methods like Branch-and-Bound Algorithms rely heavily on inference to prune their search spaces.

### 10.2.2 Cutset Conditioning

Cutset conditioning is a simple decomposition method that chooses a subset of variables \( A \) (which remain fixed throughout the course of the algorithm) so that the resulting subproblem \( \min_{s_A} E(s_A, s_{\setminus A}) \) defined in Conditioning decomposes into a set of disjoint subproblems.\(^7\) In this case, the graph of \( E(s_A, s_{\setminus A}) \) breaks into separate components indexed by \( \alpha \) which can be solved independently. Formally:

\[
\mathcal{E}(s_{\setminus A}) = \min_{s_A} E(s_A, s_{\setminus A})
= \min_{s_A} \left\{ E_{\setminus A}(s_{\setminus A}) + \sum_{\alpha} E_a(s_{\setminus A}, s_{A_\alpha}) \right\}
= E_{\setminus A}(s_{\setminus A}) + \sum_{\alpha} \min_{s_{A_\alpha}} E_a(s_{\setminus A}, s_{A_\alpha})
= E_{\setminus A}(s_{\setminus A}) + \sum_{\alpha} \mathcal{E}_a(s_{\setminus A})
\]

where \( s_A = \bigcup_{\alpha} s_{A_\alpha} \text{ and } s_{A_\alpha} \cap s_{A_{\alpha'}} = \emptyset \text{ for } \alpha \neq \alpha' \). Here, \( \alpha \) labels the disconnected components of \( A \) once all variables have been fixed to \( s_{\setminus A} \). With a proper choice of the conditioning variables, the graph is cut into disjoint partitions that are optimized with respect to \( s_{A_\alpha} \) independently. The set of conditioned variables that cuts the original problem into pieces is called the cutset. Choose a cutset such that each of the remaining \( \min s_{A_\alpha} \) problems is

\(^7\) While it is always possible to find a subset \( A \), the subset might be large for highly connected problems.
small enough to be solved on the D-Wave system. A small example is illustrated in Figure 10.1.

Figure 10.1: 8-variable graph, before (left) and after (right) conditioning. If the graph is conditioned on variable $s_1$, it decomposes into 3 smaller subproblems. The remaining variable sets $s_A = \{s_2, s_3\} \cup \{s_4, s_5, s_6\} \cup \{s_7, s_8\}$ define three decoupled subproblems.

The outer optimization problem that determines optimal values for the cut-set variables $s_A$ 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 find the global minimum of $E(s_A)$. Note, however, that the inner optimizations over $s_A$ come with no proof of optimality.

To simplify outer optimization, make the number of conditioned variables as small as possible. Determining the smallest cutset is NP-hard in general, so employ fast heuristics.

An alternative criterion to determine the set $s_A$ of conditioned variables is a tree or a collection of trees, because tree-like graphs can be optimized in polynomial time. Also determine the set $s_A$ so that the resulting graph—after the conditioned variables and their edges are removed—is either a tree or collection of disconnected trees (see [Dec1987]). The cutsets needed to form trees are typically larger than those needed to form Chimera-structured subproblems, so there is a significant advantage in using this approach to solve subproblems on the D-Wave system. An effective way to find a small set of vertex separators is to apply a graph partition heuristic like METIS,\textsuperscript{8} and then build a cutset of vertex separators by selecting a vertex cover from the edges that span partitions.

10.2.3 Branch-and-Bound Algorithms


Branch-and-bound algorithms progressively condition more and more variables to particular values. For simplicity, assume that spins $s_i$ are conditioned in index order $i = 1, 2, 3 \ldots$. At some point in the search, variable $s_i$ must be conditioned to either $s_i = -1$ or $s_i = 1$. This possibility defines a split at node $s_i$. At this node, variables $s = 1 \ldots i - 1$ have been conditioned to values and the others 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 each node in the traversal of this tree, a branch-and-bound algorithm decides whether to prune the tree at this node, effectively skipping exploration of the subtrees below it. A

\textsuperscript{8} METIS is a set of programs for partitioning graphs and finite-element meshes, and producing fill reducing orderings for sparse matrices.
branch can be pruned if it can be shown that it is not possible for any leaf node below it to contain the global optimum. This pruning is accomplished by maintaining upper and lower bounds on the global minimum of \( E(s) \) as the tree traversal is carried out. At any point in the search, the upper bound is the smallest value \( E(s) \) that has been seen at the visited leaves. The lower bound at a node in the search is calculated as a lower bound on \( \min_{s_A} E(s_A, s_{\setminus A}) \) where \( s_A \) is the set of as-yet-unconditioned variables. If the lower bound at a given node exceeds the current upper bound, then there is no need to search below this node, since no leaf of any of its subtrees can be optimal. The efficacy of this pruning is determined both by the tightness of the lower bound approximation and by the ordering in which variables are conditioned.

Branch-and-bound methods can benefit from the quantum annealing solver by terminating searches higher in the tree. After sufficient variables are conditioned so that the remaining unconditioned variables can be optimized by the D-Wave system, there is no need to explore deeper—simply call the D-Wave system to estimate the best completion from that node. As the upper bound is minimized through subsequent D-Wave system completions, this may in turn allow for future pruning. Again, since the D-Wave system does not come with a proof of optimality, this algorithm may not return a global minimum.

Another way to incorporate the D-Wave system is to use quantum annealing to provide tight lower-bound functions at any node in the search tree. Lagrangian Relaxation for Finding Lower bounds discussed below is one approach.

During the branch-and-bound process, you can also take advantage of cutset decompositions. As the search moves between nodes in the search tree, the graph for the subproblem that must be solved changes. In particular, the vertices in the graph that correspond to the fixed variables at the search tree node are removed along with all edges connected to these vertices. The graph may disconnect at some nodes in the search tree, in which case you 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 accessed. See [Mar2007], which considers other ways to explore the search tree, including dynamic variable orderings and best-first orderings. All of these improvements may also be applied in this setting. See also [Ros2016] on branch-and-bound heuristics in the context of the D-Wave Chimera architecture.

### Lagrangian Relaxation for Finding Lower bounds

Lagrangian relaxation relies on a different simplification of the graph from that shown in Figure 10.1 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 graph representing a variable \( s_i \). Divide 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 \( E(s_i, s_{\setminus i}) \) is modified to \( E'(s_i^{(1)}, s_i^{(2)}, s_{\setminus i}) \). If you ignore the equality constraints among divided variables, then with sufficiently many divided variables, the modified objective \( E' \) decomposes 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, \( h_i = h_i^{(1)} + h_i^{(2)} \) is required. The advantage of dividing variable \( s_i \) comes when the equality constraint is softened and treated approximately.

Introducing a multiplier \( \lambda_i \) for the equality constraint, the Lagrangian for the constrained
problem is
\[ L(s^{(1)}_i, s^{(2)}_i, s_{j}; \lambda_i) = E'(s^{(1)}_i, s^{(2)}_i, s_{j}) + \lambda_i (s^{(1)}_i - s^{(2)}_i). \]
Minimizing this Lagrangian for any fixed value of \( \lambda_i \) provides a lower bound to \( \min_s E(s) \).

The dual function is defined as
\[ g(\lambda_i) = \min_{s^{(1)}_i, s^{(2)}_i, s_{j}} L(s^{(1)}_i, s^{(2)}_i, s_{j}; \lambda_i), \]
and maximizing the dual with respect to \( \lambda_i \) provides the tightest possible lower bound.

As an example, consider again the small 8-variable problem, shown on the left side in Figure 10.2. The Lagrangian relaxed version of the problem obtained by dividing variable \( s_1 \) is shown on the right. 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 subproblem is still too large, it can be decomposed further either through another variable division or through conditioning.

One promising approach is to introduce as many divided variables as needed to generated subproblems small enough to be solved with the current number of qubits. Each subproblem is solved to give a dual function, which is then optimized using a subgradient\(^9\) method to provide the tightest possible lower bounds.

Because a particular anneal may not produce the global minimum, the lower bounds returned by Lagrangian relaxation may be too high. If so, branch-and-bound may mistakenly prune branches containing the global minimum. However, viewing this Lagrangian-relaxed branch-and-bound algorithm as a heuristic, expect good results if the quantum annealing process is effective at locating low-energy states.

10.2.4 Large-Neighborhood Local Search Algorithms

Local search algorithms improve upon a candidate solution, \( s^t \), available at iteration \( t \) by searching for better solutions within some local neighborhood of \( s^t \). Most commonly, the neighborhood for bitstrings is the Hamming neighborhood consisting of all bitstrings that

Quantum annealing can be very simply combined with local search to allow the local search algorithm to explore much larger neighborhoods than the 1-bit-flip Hamming neighborhood. As the size of the neighborhood increases, the quality of the local minima improves. Consider a problem of \( N \) variables, and define the neighborhood around configuration \( \mathbf{s} \) as all states within Hamming distance \( d \) (with \( d < N \)) of \( \mathbf{s} \). Searching within this neighborhood for a configuration \( \mathbf{s}^{d+1} \), having energy lower than \( E(\mathbf{s}^d) \), is a smaller optimization problem than the original; however, it still cannot be directly carried out by quantum annealing hardware. Instead, choose one of the \( \binom{N}{d} \) subsets of \( d \) variables. Call the selected subset \( A \) and note that \( |A| = d \). Having selected \( A \), determine the best setting for these \( s_A \) variables given the fixed context of the conditioned variables \( \mathbf{s}_{\setminus A} \). Select \( d \) such that the smaller \( d \)-variable problem can be embedded within the D-Wave QPU. If no improvement is found within the chosen subset, another subset of size \( d \) may be selected.

This algorithm stagnates at a configuration that is a local optimum with respect to all \( d \) variable spin flips. However, if \( d \) and \( N \) are both large, many size-\( d \) subsets around each configuration \( \mathbf{s}^d \) exist, so trapping in \( d \)-optimal local minima rarely occurs and progress may be slow as mainly unpromising neighborhoods are explored. It is possible to infer variable subsets that need not be considered, and heuristics may be defined that generate promising subsets\(^{10}\).

### 10.3 Embedding

The D-Wave QPU minimizes the energy of an Ising spin configuration whose pairwise interactions lie on the edges of a Chimera graph \( M, N, L \). To solve an Ising spin problem with arbitrary pairwise interaction structure, the corresponding graph must be minor embedded into a Chimera working graph.

An Ising optimization problem is defined on an arbitrary graph containing \( N \) nodes (variables) and \( M \) edges. The working graph in the D-Wave system corresponds to a graph with fixed connectivity—at most 6 edges per node—on which you can mimic arbitrary connectivity by using qubits to represent edges as well as variables.

In Lagrangian relaxation, an optimization variable \( s_i \) is split into two variables, \( s_i^{(1)} \) and \( s_i^{(2)} \), and the constraint \( s_i^{(1)} = s_i^{(2)} \) added. A similar technique can be used to map an optimization variable \( s_i \) onto a set of one or more qubits \( \{q_i^{(1)}, \ldots, q_i^{(k)}\} \). Because all qubits (assumed to be \( \pm 1 \) valued) represent the same problem variable, 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:

\[
-Mq_i^{(j)}q_i^{(j')}
\]

\(^{10}\) [Liu2005] presents promising results for even small neighborhoods of size \( d \leq 4 \). With the D-Wave QPU, much larger neighborhoods can be considered.
where $M > 0$ is the weight of the penalty. If $M$ is sufficiently large, the lowest energy state in the full problem always has $q^{(j)}_i = q^{(j')}_i$ because that feasible assignment is $2M$ lower in energy than the infeasible assignment $q^{(j)}_i \neq q^{(j')}_i$.

In this way, strings of qubits are related to each other to create chains that can connect arbitrary vertices. To create these chain-like connectors, weigh the penalties large enough so that low-energy configurations do not violate the equality constraints. Balancing this, use the smallest possible penalty weight that enforces 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.

For example, if qubits $q_4$ and $q_{32}$ represent problem variables $s_1$ and $s_2$ respectively and an $(s_1, s_2)$ edge in the problem graph exists, then this coupling can be realized by representing problem variable $s_1$ with the string of qubits $\{q_4, q_{12}, q_8\}$, as shown in Figure 10.3. 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 mapping a single optimization variable to a connected set of qubits in the working graph mimics higher degrees of connectivity than are present in the graph. For example, variable $s_1$ represented by qubits $\{q_4, q_{12}, q_8\}$ can be connected to up to 12 other qubits and thus up to potentially 12 other optimization variables (if each qubit represents a distinct optimization variable).
10.3.1 Formalizing Minor Embeddings

This section considers the general problem of solving an arbitrary problem on the working graph in light of the variable-to-qubit mapping observation above.

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 \( v_2 \), combining them into one. If the sets of neighboring vertices of \( v_1 \) and \( v_2 \) in the original graph are \( N_1 \) and \( N_2 \) then the neighbors of the fused node are \( (N_1 \cup N_2) \setminus \{v_1, v_2\} \). Given this notation, if \( G \) is a general graph representing an Ising problem, it can be minor embedded into Chimera if it is isomorphic to a graph minor of the Chimera working graph.

To more formally specify the embedding problem, let \( G = (V, E) \) be a graph for a problem. Fundamentally, there are two requirements on embedding:

- A given vertex \( v \in V \) must be mapped to a connected set of qubits \( q \in \text{Chimera} \).
- All edges \((v_1, v_2) \in E \) must be mapped to at least one edge in the working graph.

To satisfy the first constraint, posit an unknown table, EmbedDist\((v, q, d)\), that maps vertex \( v \) to qubit \( q \) and is distance \( d \) from a reference qubit also mapped from \( v \). Additionally, define Embed\((v, q)\) \( \leftrightarrow \exists d \) EmbedDist\((v, q, d)\). The constraints\(^{11}\) on these tables are then:

1. \( \forall v, q, d \) EmbedDist\((v, q, d) \rightarrow (d = 0) \lor (\exists v_1, q_1 \text{Chimera}(q_1, q_1) \land \text{EmbedDist}(v_1, q_1, d - 1)) \).
2. \( \forall v_1, v_2 E(v_1, v_2) \rightarrow \exists q_1, q_2 \text{Chimera}(q_1, q_2) \land E(v_1, q_1) \land \text{Embed}(v_2, q_2) \).

The predicate Chimera\((q_1, q_2)\) is true if and only if there is an 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 graph between problem variables \( v_1 \) and \( v_2 \). In addition to these two constraints, there are the following 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 \( q \): \( \forall v, q \text{Embed}(v, q) \rightarrow \neg \exists v_1 (v_1 \neq v) \land \text{Embed}(v_1, q) \). The second bookkeeping constraint can also be specified as \( \forall q \text{COUNT}(v, 1, \text{Embed}(v, q)) \leq 1 \).\(^{12}\)
5. There is 1 root for each \( v \): \( \forall v \text{COUNT}(q, 1, \text{EmbedDist}(v, q, 0)) = 1 \).

This specification does not minimize the total number of qubits used. However, when quantifying over \( d \) (the size of the connected graphs representing problem variables), specify a maximal value \( D \) so that \( 0 \leq d \leq D \). In this way, the size of the embeddings may be somewhat controlled. For graphs specified through \( E(v_1, v_2) \), however, the problem may be infeasible if \( D \) is too small.

Note: These techniques help solve arbitrary Ising problems by transforming the problem to an alternative form that preserves the low-energy states. In particular, qubits are introduced and constraints added that force sets of qubits to take identical values. These

\(^{11}\) This constraint specification is executable, and solvers are available that can solve for the unknown tables EmbedDist\((v, q, d)\) and Embed\((v, q)\) from the constraint specification.

\(^{12}\) The aggregate operator COUNT\((v, 1, \phi(v))\) counts the values in the domain of \( v \) for which predicate \( \phi(v) \) is true.
transformations may affect the efficacy of quantum annealing.

10.3.2 Embedding Complete Graphs

Because a complete graph on $V$ vertices (all vertices connected to all other vertices) can be used to represent any Ising problem defined on $V$ variables, consider what is the largest complete graph $K_V$ that is a minor of a $M \times N \times L$ graph. Knowing this complete minor allows for immediate embedding of all problems of $V$ or fewer variables.

The largest complete minor has $V = 1 + L \min(M, N)$ vertices.\(^{13}\)

A graph minor for a small $2 \times 2 \times 4$ graph is shown in Figure 10.4. A similar result for a $4 \times 4 \times L$ graph is in Figure 10.5. If Figure 10.5 were expanded, the top left corner comprising the A and B blocks alone would appear as in Figure 10.4, for $L = 4$.

![Figure 10.4: Embedding of $K_9$ into $2 \times 2 \times 4$; A and B indicate two different blocks of 4 variables. Equivalently labeled qubits are connected by edge penalties.](image)

Connectivity is addressed at the cost of larger lattices. Note also that specific connectivity requirements may have problem-specific embeddings that make more effective use of qubits than using Chimera as the complete graph on $V$ vertices.

\(^{13}\) That this is the largest complete minor follows immediately from the tree-width of Chimera being $L \min(M, N)$.\(^{13}\)
Figure 10.5: Condensed representation of the complete graph of size $K_{4L+1}$ on $4 \times 4 \times L$. 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.
10.3.3 Finding Better Embeddings

In some cases, the exact form of a problem is malleable. One important example arises in machine learning problems. The true objective in predictive modeling is to minimize the error on future observations so that they are accurately predicted. However, without 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. Minimizing training set error runs the risk of over-fitting and additional terms favoring “simpler” models over complex ones are typically added to the objective. Thus, there is latitude in the precise objective. In such cases, the embedding method can be simplified.

Let $G = (V, E)$ represent the Chimera graph. 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 $J_{i,j}$ must be assigned to an edge. The simplifying assumption requires that each node in the original graph be mapped to a single node in Chimera.

The mapping $\varphi$ is encoded with a set of binary variables $M_{i,q}$, which indicate that problem variable $i$ is mapped to Chimera node $q$. A valid mapping requires $\sum_q m_{i,q} = 1$ for all problem variables $i$, and $\sum_i m_{i,q} \leq 1$ for all Chimera nodes $q$. Because the original problem can be altered, fit the problem into Chimera by maximizing the total magnitude of $J_{i,j}$ mapped to Chimera edges; that is:

$$M^* = \arg\max_M \sum_{i \neq j} \sum_{(q,q') \in E} |J_{i,j}| m_{i,q} m_{j,q'}$$

where $M$ 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, apply a greedy heuristic to approximately maximize the objective.

As a starting point, let $i_1 = \arg\max_i \sum_{j < i} |J_{i,j}| + \sum_{j > i} |J_{i,j}|$. So $i_1$ is the row/column index of $J$ with the highest sum of magnitudes ($J$ is assumed to be upper-triangular). Map $i_1$ to one of the Chimera vertices of highest degree. For the remaining variables, assume $\varphi$ is defined 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 $|J_{i_{k+1},j}|$ and $|J_{i, i_{k+1}}|$ over all $j \in \{1, \ldots, k\}$ for which $(q_j, q_{k+1})$ is a Chimera edge. This fast greedy heuristic performs well.

10.4 Solving a Problem on the QPU

10.4.1 Spin-Reversal Transform

Applying a spin-reversal transform can improve results by reducing the impact of analog errors that may exist on the QPU. A spin-reversal transform does not alter the Ising problem, in the sense that there is a direct one-to-one mapping between all solutions to the original problem to the altered problem that preserves their energies—solutions of the original problem and of the transformed problem have identical energies. Rather, the transform
reverses the meanings of a collection of individual spins $s_p$ as follows:

$$s_p \rightarrow s'_p = -s_p$$

$$h_p \rightarrow h'_p = -h_p$$

$$J_{i,j} \rightarrow J'_{i,j} = -J_{i,j}$$

for either $i = p$ or $j = p$. 

(10.1)

This mapping ensures that the classical Ising spin energy is unchanged; the transform simply amounts to reinterpreting spin up as spin down, and visa-versa, for a particular spin. Any number of spins may be transformed in this way, but the Ising problem remains the same and therefore we expect the same answers. A spin-reversal transform is defined according to a subset of spins for which this transform is applied.

Some sources of ICE depend on the magnitude of the $(h, J)$ problem parameters. Random spin transformations can be used to avoid systematic errors arising from multiple requests of the same problem, by “breaking up” these dependencies.

From a single problem defined by a set of parameters $\{h_i, J_{i,j}\}$, a sequence of spin-reversal transforms $T', T''\ldots$, each applied to a random subset of spins, defines a sequence of equivalent problems $(h', J')$, $(h'', J'')\ldots$. All have the same energy spectrum, but their implementation on the QPU may invoke different ICE effects (some better and some worse than the original).

When studying the behavior of the QPU in solving a problem, include in the analysis a number of spin-reversal transforms of the problem. The transform samples randomly over sets of possible systematic errors. The correlations between these error sets, at least as they relate to certain error sources, can be reduced with the following considerations:

- Changing too few spins leaves most errors unchanged, and therefore has little effect.
- Changing too many spins means that most couplers connect spins that are both transformed, thus $J_{i,j}$ does not change sign. As a result, some systematic errors associated with the couplers are unaffected.

To sample over a wide range of systematic errors, consider the number of couplers and spins involved in a particular transform.

**Note:** Spin-reversal transforms do not necessarily remove errors due to ICE; rather, they randomize the effects of certain errors that depend on $(h, J)$. This creates a spectrum of error magnitudes from which the best results can be selected. Some types of systematic error—ICE 1, as defined in Technical Description of the D-Wave Quantum Processing Unit for example—are not affected by spin-reversal transforms.

Specify the number of spin-reversal transforms using the `num_spin_reversal_transforms` parameter when submitting a problem.

### 10.4.2 Using Anneal Offset

To run a problem with anneal offsets:

1. Permitted parameters vary by QPU, so first retrieve the solver properties for your system.
If anneal offsets are supported, the returned values include a property that shows the positive-negative range of permitted offset values, in normalized offset units, for each qubit in the QPU. (Both values are 0 for any inoperable qubits.) Negative values force the qubit to anneal later than the standard trajectory; positive values, earlier.

In the following truncated list, for instance, we see that qubit 0 can be annealed a maximum of 0.2255351369888578 normalized offset units later than the standard trajectory, or a maximum of 0.050118919330857284 normalized offset units earlier.

\[
[-0.2255351369888578, 0.050118919330857284],
[-0.2258618355939089, 0.12045964565008474], ...
\]

2. Embed the problem on the graph. For best results, embed the problem multiple times.

3. Prepare an array of offset values based on the information retrieved from the system, the hardware adjacencies of the embedded problem, and any known dynamics of the problem.

4. Supply the array of offsets for the qubits in the system using the `anneal_offsets` parameter with a length equal to the `num_qubits` property. The appropriate format depends on the client; see the appropriate developer guide for your client for more information.

5. Submit the problem.

6. Compare results of the same problem run with and without offset annealing paths, and the same problem run with different offset values. Adjust the offset values accordingly.
CHAPTER 11

BIBLIOGRAPHY
BIBLIOGRAPHY


[Dwave1] “Structured Prediction Reference Example Description”

[Dwave2] “RBM Reference Example Description”


