UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Effectively Encoding SAT and Other Intractable Problems into Ising Models for Quantum Computing

Stefano Varotti

Advisor

Prof. Roberto Sebastiani

DISI, Università degli Studi di Trento

Co-Advisor

Dr. Aidan Roy

D-Wave Systems Inc.

January 2019

Abstract

Quantum computing theory posits that a computer exploiting quantum mechanics can be strictly more powerful than classical models. Several quantum computing devices are under development, but current technology is limited by noise sensitivity. Quantum Annealing is an alternative approach that uses a noisy quantum system to solve a particular optimization problem. Problems such as SAT and MaxSAT need to be encoded to make use of quantum annealers. Encoding SAT and MaxSAT problems while respecting the constraints and limitations of current hardware is a difficult task. This thesis presents an approach to encoding SAT and MaxSAT problems that is able to encode bigger and more interesting problems for quantum annealing. A software implementation and preliminary evaluation of the method are described.

Keywords
AQC, Quantum Annealing, SAT, MaxSAT, SMT

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Disclaimer: The research work described in this PhD Dissertation is joint work with Dr. Zhengbing Bian, Dr.Fabian Chudak, Dr.William Macready, Dr.Aidan Roy from D-Wave System Inc. and with my advisor Prof.Roberto Sebastiani from Università di Trento. Most of the scientific content presented here can be found in the papers:

- Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and MaxSAT with a quantum annealer: Foundations and a preliminary report. In *The 11th International Symposium on Frontiers of Combining Systems, FroCoS'17*, volume 10483 of *LNCS*. Springer, 2017.
- Zhengbing Bian, Fabián A. Chudak, William G. Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and MaxSAT with a quantum annealer: Foundations, encodings, and preliminary results. 2018. `https://arxiv.org/abs/1811.02524` . Under submission for journal publication.

The development of quantum computing theory has been one of the most intense area of research in complexity theoretical computer science in recent years. Many ways have been devised to exploit quantum weird-

ness to obtain computational speedups, but the development of quantum computing hardware has been stuck by the high susceptibility to noise. While noise tolerant quantum circuits are slowly improving over time, the currently available hardware shows a behavior that is typical of quantum systems but provides limited computational flexibility. Quantum annealers accept some level of noise and decoherence to allow for large quantum systems.

1.1 The Problem

Quantum annealers are able to solve a specific subset of a single optimization problem. Whereas this problem is NP-hard, and thus it is theoretically possible to convert any NP problem instance into such problem, in practice the process of encoding is difficult due to several limitations.

The thesis will focus on SAT problems, as alternative tools able to tackle worst-case instances would be critically useful. Whereas the hardware is able to solve several NP-hard problems it is not particularly suited to solve SAT problems. To demonstrate the capabilities of quantum annealing, SAT problems that are considerably hard for state-of-the-art SAT solvers are preferred. Encoding such a problem into a quantum annealing problem would provide a convincing evaluation for the approach.

1.2 The Solution

In this thesis, a method is proposed to encode SAT and MaxSAT problems into quantum annealing problems. First, a method to generate optimal encodings for Boolean functions is outlined. This method uses SMT solvers and requires knowing the models and counter-models of the Boolean function, and thus it is not applicable to solve large SAT problems. SMT

solving is instead used to build a library of efficiently encoded gates.

The second part of the solution consists in a multi-step process where the input Boolean formula is broken into multiple components from the pre-encoded library and these elements are re-composed into a final quantum annealing problem. This part uses a heuristic approach to shape the input formula into an encoding that respects the hardware constraints.

This approach provides an effective and efficient method for SAT problem encoding. It is thought for SAT solving of generic Boolean circuits, while the state of the art is focused either on optimization problem or in specific constraint satisfaction problems.

1.3 Structure of the Thesis

The first half of the thesis, containing Chapter 2 to 4, will provide the context and background for the thesis. The second chapter will outline the motivations behind this area of research and the goals of this thesis, why quantum computing can be useful and what are the current limits. The third chapter will provide a comprehensive background on quantum computing, quantum annealing and other aspects of quantum computation, and then it will introduce various logic problems such as SAT, MaxSAT, SMT and OMT. The fourth chapter will survey related work on the same problem and highlight differences and limitations.

The second half of the thesis, from Chapter 5 to 9, will show the novel contributions. The fifth chapter will lay the theoretical foundations for the rest of the work, stating concepts such as penalty functions, variable placement, and embedding. The sixth chapter will explain how to use SMT and OMT solvers to find the optimal encoding of a Boolean function and possibly a placement. The seventh chapter will provide a framework for encoding large SAT problems using functions pre-encoded with the

techniques explained in the previous chapter. Each of the steps will be explained in details. The eighth chapter will describe an implementation of the work, describing details and usage of several Python libraries. The basic file formats, encoding tasks and command line interfaces will be described. Finally, The ninth chapter will provide an experimental evaluation of the techniques on SAT and MaxSAT problems, with details on the choice of the benchmark problem.

Part I

Motivations, Background, State-of-the-art

Chapter 2

Motivations and Goals

Quantum computing research has been thriving over the last decades since its inception. This chapter is dedicated to explaining the reason for this interest, what is its potential advantage over currently available computers, and what are the obstacles that need to be overcome. First it will introduce the context of quantum computation and computational complexity. Then it will outline the computational advantages of quantum computing, and its practical limits. Finally, the chapter will focus on the particular branch that the thesis will contribute to and on which are the main obstacles to be overcome.

To understand the interest in quantum computing we need to consider its context in the field of complexity theory. Complexity theory is concerned with the asymptotic resource requirements necessary to solve a problem. One of the most important results in complexity theory is NP-completeness and NP-hardness. These concepts are fundamental and required for understanding quantum computing. In the last 20-30 years there has been a development of the theory of quantum computing complexity. This theory has important implications in computer science and physics and opens new possibilities in computing.

Whereas the research on quantum computer technology still has not

reached the goal of manufacturing reliable large-scale devices, limited alternative models have proven to be feasible. In particular, quantum annealing has shown an advantage over the classical version, simulated annealing. We will see how quantum annealing differs from gate-model quantum computers and what are the limits of this approach.

2.1 Intractable problems and NP-Hardness

Two of the most important concepts in complexity theory are nondeterministic polynomial complexity and NP-hardness. The **Nondeterministic Polynomial (NP)** class of problems consists in all the problems that have an algorithm that, given a solution, are able to check it in polynomial time. If no better algorithm is available, in the worst case every possible solution has to be checked. A **NP-hard** problem is a problem such that an instance of any problem in the NP class can be converted into an instance of the NP-hard problem. Thus, an efficient solver for any single NP-Hard problem would be capable to solve all NP problems as well. For this reason NP and NP-hard problems are considered to be intractable, i.e. they do not have an efficient algorithm. The proof of intractability of NP problems (the so-called P vs. NP problem) is still an open problem, but intractability is strongly believed due to various weaker theorems [46]. All NP-complete and NP-hard problems known so far have no efficient algorithm, and any algorithm found would solve efficiently all the other problems. When we encounter NP-hard algorithms in the real world, we rely on heuristics and approximations, and we are not guaranteed to reach the solution of our problem in a reasonable time.

Intractability is a problem because many real-world useful problems are NP-Hard, from optimization, planning and artificial intelligence in general, system analysis and many others. Almost all cryptography techniques rely

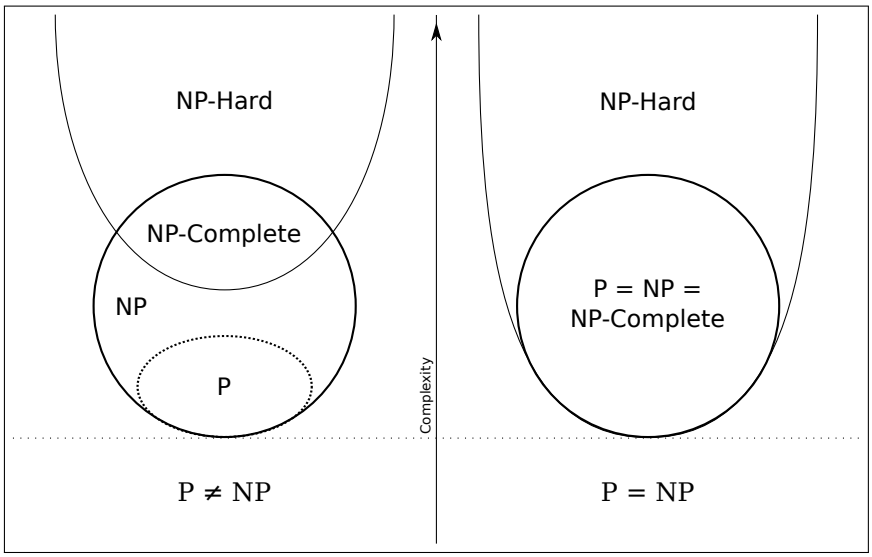


Figure 2.1: Venn diagrams for complexity classes, if $P \neq NP$ or if $P = NP$. ©Behnam Esfahbod, under CC-BY-SA license.

on the intractability of various NP algorithms. As an example that will be useful later, **Public Key Cryptography** relies on the hardness of the factorization problem. Factorization is in NP and no polynomial time algorithm is known, but it is not believed to be NP-hard. The abstract nature of the NP-hardness definition implies that problem complexity and intractability is independent of the hardware used, as long as it follows the deterministic turing machine model. This was the case for all known computer architectures until the development of quantum computing theory. The non-determinism of quantum mechanics happens to allow a more powerful model of computation.

2.2 Quantum efficient problems

The most surprising characteristic of the quantum mechanics theory is the intrinsic non-determinism that it suggests. The laws of physics are generally formulated in order to gain more knowledge about the world, but quantum mechanics posits that there are some details of reality that

are intrinsically unknowable. It is an inherently probabilistic theory, unlike statistical mechanics. In the classical statistical mechanics view of the world, probability distribution is caused by the uncertainty of the observer. In quantum mechanics we have superposition of states instead. Different world histories interfere with each other. This leads to all kinds of counter-intuitive phenomena: entanglement, tunneling. A notable example is the Einstein-Podoslky-Rosen paradox, an experiment that disproves any hidden variable theory [86].

Quantum computing is a model of computation that relies on the weirdness of quantum mechanics. We expect a computer to strictly follow a specific sequence of instructions and acts on it internal state to reach a solution. This is the deterministic model of computation. Non-deterministic automata instructions instead are partially defined, and succeed when there exists a sequence of legal instructions that solve the problem.

As a quantum system is a superposition of classical states, a quantum computer is in a superposition of states. While the evolution of a quantum system is completely determined by its Hamiltonian function, Bell's theorem provides an example of a quantum system which behavior cannot be described by a deterministic local state (a local hidden variable). The behavior of quantum computers lie in between deterministic and non-deterministic machines, more powerful than deterministic, but bound by quantum-mechanic laws to less than the vast possibilities of non-deterministic automata. The implications of this will be explored further in the next chapter. The class of problems that are efficiently solvable by quantum computers is called **bounded-error quantum polynomial time (BQP)**. Various problems have been shown to have efficient quantum algorithms. The performance of a quantum algorithm relative to the best classical alternative is called quantum speedup. It is not necessarily the case that the quantum algorithm is asymptotically better, but quantum

Problem	Classical complexity	Quantum complexity	Quantum speedup
Factorization	$O(e^{(n \log n)^{1/3}})$	$O(n^2 \log n)$	exponential
Unstructured search	$O(2^n)$	$2^{\sqrt{n}}$	quadratic
Deutsch–Jozsa algorithm	$O(2^n)$	1	exponential

Table 2.1: Complexity comparison between various algorithms, quantum speedups [56].

computing subsumes classical computing so it can only improve.

The most famous problem that is sped up by quantum computers is factorization, which hardness underlies all public key cryptography schemes used nowadays on the Internet. In 1994, Peter Shor showed that there is a quantum algorithm that solves efficiently factorization[87]. Thus the ability to manufacture a scalable quantum computer would compromise most current internet security standard. As research makes quantum computing more and more feasible, there is large interest for quantum-proof encryption, and work for standardization is underway[11, 26]. On the upside, another important problem in BQP is simulation of quantum systems[40]. Quantum supremacy obviously implies that quantum systems are not efficiently simulable by classical computers. We would like to predict and analyze the behavior of quantum systems, and classical computer can do so with limited accuracy. An efficient quantum simulation would be extremely useful in science, especially material science.

Still, a quantum computer is not as powerful as a generic non-deterministic Turing machine. **Grover's algorithm** [51] provides a brute-force search that improves worst case complexity from $O(2^n)$ to $O(2^{\sqrt{n}})$, and subsequent work has proved that this is the best possible quantum algorithm for unstructured search[10]. In this case then the best result we can get is only a quadratic speedup, which is still worse than a non-deterministic Turing machine that is exponentially faster.

The optimality of Grover's algorithm implies that exponential speedup

can be obtained only when the problem has an underlying structure that can be exploited. For example, factorization can be sped up because it is a special case of the so-called hidden-subgroup problem. The hidden-subgroup problem consists in finding an algebraic group hidden in a bigger algebraic structure, and quantum computers can solve it efficiently in the case of finite Abelian groups. Table 2.1 shows a sample of algorithms and their best-known complexities on classical and quantum computers.

2.3 Noise and decoherence in quantum computing

Whereas quantum computing theory has grown considerably and presented many useful novel algorithms, the construction of an actual quantum computer has proved to be a challenge. As we will see later, quantum computing assumes a noiseless computing hardware, and by itself has no provision for noise tolerance. Thus, quantum computers are really sensitive to noise, as any interaction with environment causes **decoherence**. A system that fully loses coherence becomes equivalent to a non-quantum system in a random state. Indeed a topic of great interest in quantum computing and quantum communication is error resilience. Some solutions are quantum error correction or topological quantum computing.

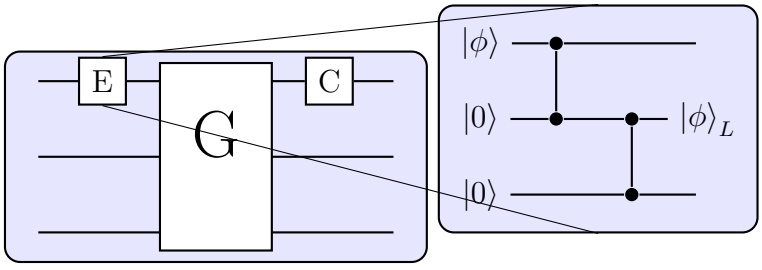


Figure 2.2: An example of quantum circuit, where the input is encoded. Each logical qubit is represented as group of three physical qubits (on the right), and is error corrected after the operation.

Quantum error correction subsumes standard error correction schemes

in order to be able to cope with corruption of quantum states. A system of multiple qubits is used to resiliently store a single logical qubit. An illustration of a circuit with a simple error correction mechanism is available in Figure 2.2. Before the computation happens, a single logical qubit is encoded into multiple qubits and after computation any error or change is corrected. If the error level in the underlying system is below a certain threshold, one can build a error correction system with arbitrarily low error rate[47]. It is estimated that thousands of qubits are necessary for error-correcting a single logical qubit[27]. Whereas the technology is bound to improve, error-free quantum computation needs to attain significant manufacturing improvements before being possibly realizable.

Currently available quantum computing hardware is still too noisy to perform significant error-corrected quantum computation. An alternative approach is instead to focus on exploiting quantum systems that are noisy and decohere quickly: a way to do that is using a process called **Quantum Annealing**. Quantum annealing is the quantum analogue of simulated annealing, a standard algorithm in optimization. Simulated annealing mimics the annealing process in physics in order to optimize a desired function. It is often effective because it provides a simple compromise between exploitation and exploration, but of course it is likely to get stuck on sub-optimal solutions on hard problems. Quantum annealing is essentially a quantum system that undertakes an annealing process. it is less likely to get stuck in higher-energy states due to the **tunneling effect** even if the system loses coherence before the end of the process.

2.4 Adiabatic Quantum Computing and Quantum Annealing

Quantum annealing is closely related with the theory of adiabatic quantum computing. **Adiabatic quantum computing** is an alternative model of quantum computing that exploits the quantum adiabatic theorem. This theorem ensures that a quantum system that evolves sufficiently slowly in time will stay in the lowest energy state, called **ground state**. An adiabatic quantum computer works by slowly transitioning a cold quantum system from a standard initial state into a desired final state. The user programs the computer by engineering the system in such a way that its final ground state encodes the solution to the specified problem. The time necessary to perform this transition is determined using the quantum adiabatic theorem, and depends on the lowest energy gap between the ground state and the second lowest energy state during the transition.

The minimum transition time represents the time required by an adiabatic quantum computer to solve a problem. Because of this its estimation and asymptotic growth has been the focus of researchers. Initially it was thought to be more powerful than traditional quantum computing, but a careful analysis proved worse performance when the annealing speed is assumed to be constant, and equivalent with an adaptive annealing speed [3]. Furthermore, a traditional quantum computer can be simulated with an adiabatic one and vice versa, so they truly are two different interpretations of the same computing model. Adiabatic quantum computing though has an advantage considering that it takes noise in consideration: the minimum energy gap separates the noiseless ground state from noise-excited states, and thus ensures that there is no interaction with noise below a certain level. Thus, compared to the gate model the adiabatic model includes a form of noise resilience in its model. On the other hand, there is no known

error correction threshold theorem for AQC.

Adiabatic quantum computing assumes that the system never leaves the ground state. This happens only when the system is at absolute zero and perfectly isolated, or at least when it is isolated up to noise lower than the minimum gap. In real systems though the temperature can never reach absolute zero, and noise cannot be eliminated completely. Thus with the current technology we cannot reliably ensure the adiabatic condition. In fact, current hardware still faces significant problems in managing noise and large scale systems tend to lose coherence very fast.

We can still exploit the fact that when the adiabatic condition is removed the system still tends toward the lowest energy state in a process called annealing. A quantum annealer still shows the presence of quantum effects but loses coherence during the run. Quantum annealing accepts a loss of coherence that is too big to perform adiabatic quantum computing, but still manages to exploit partial coherence to perform tunneling. While simulated annealing perform a stochastic search of the lowest energy state, a quantum annealer tries to exploit coherence to explore a larger state space at the same time [4].

Quantum annealers, compared to the quantum gates model, have also a benefit in their affinity to the simulated annealing algorithm. While quantum algorithm like Shor's algorithm have no counterpart in classical computing, simulated annealing is a widely used and established algorithm in stochastic optimization. Thus simulated annealing can provide a clear reference for evaluating performance and results of quantum annealers.

2.5 Quantum Annealers and D-Wave

D-Wave Systems has built several quantum annealers. These machines consist in Josephson junctions with tunable interactions. This kind of

circuit requires to be cooled to close to 0 degrees Kelvin, and are heavily shielded against EM radiation.

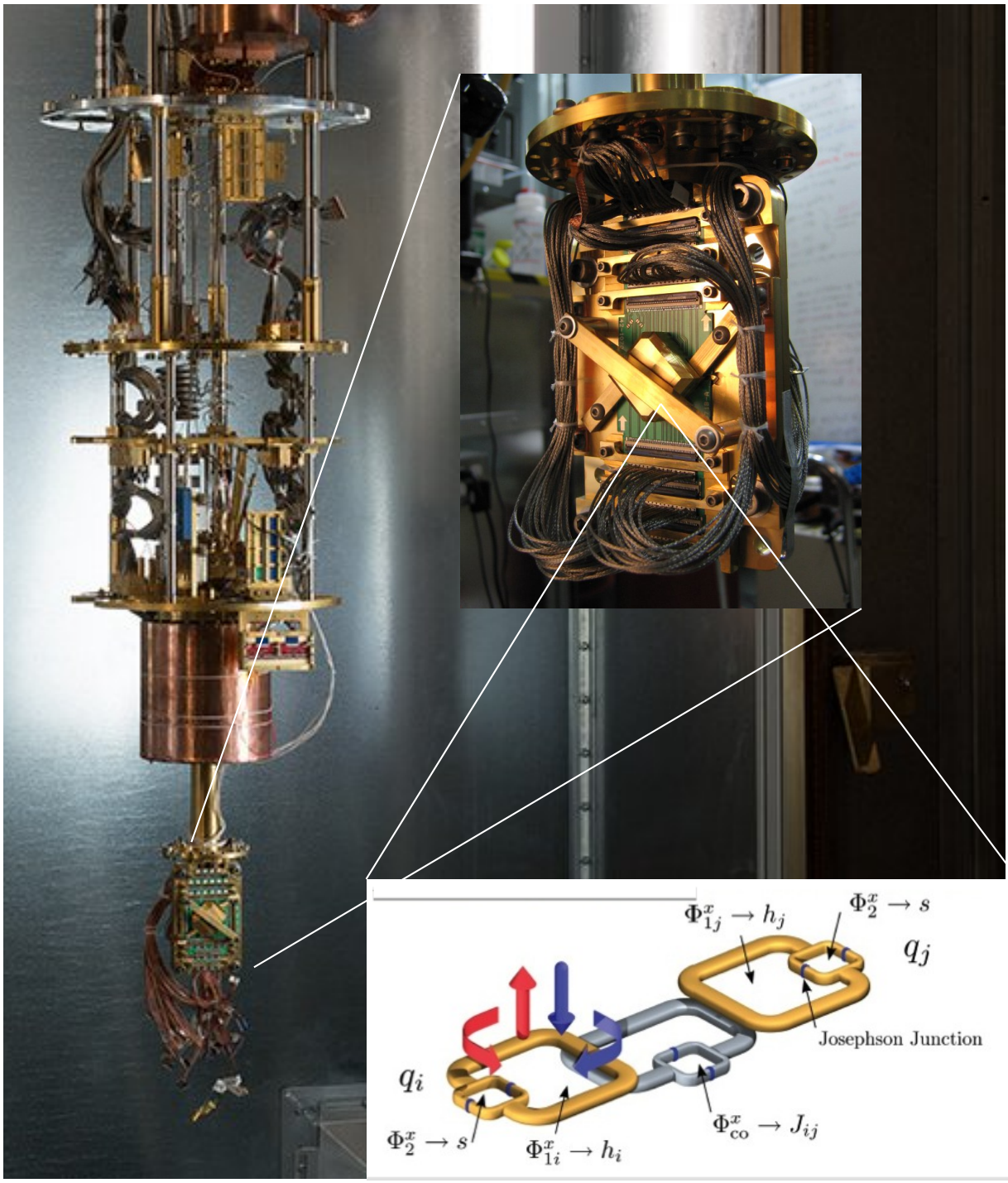


Figure 2.3: D-Wave's Quantum annealer. A picture of the refrigerator/shielding with a detail on the chip and on the schema of a Josephson junction. Courtesy of D-Wave Systems Inc.

The energy model of a D-Wave quantum annealer can be represented by a second degree real polynomial in where qubit states can be either -1 or 1. Such type of models is widely studied in Physics, and it is known as **Ising model**. The formula for this model is shown in equation 2.1. In this model, z_i are variables in $\{-1,\ 1\}$ and represent the state of a qubit. The parameters θ that influence the system's behavior are divided in three types: θ_0 is called **offset**, the θ_i are called **biases** and the θ_{ij} are called **couplings**.

$$P(z) \stackrel{\text{def}}{=} \theta_0 + \sum_i \theta_i z_i + \sum_{i,j} \theta_{ij} z_i z_j \tag{2.1}$$

The problem of finding the ground state given of an Ising model is a particular case of the so called **quadratic unconstrained binary optimization (QUBO)** problem.[1]It is easy to show that QUBO is a NP-hard problem by reducing CIRCUIT-SAT (satisfiability of a Boolean circuit), a NP-complete problem, into QUBO. We can translate each gate of a circuit into a simple sub-problem and then compose them in a consistent way (we will see later how). Table 2.2 shows simple translations for basic logic gates.

Gate	Formula	QUBO polynomial
AND	$x_3 = x_1 \wedge x_2$	$3 - x_3 - x_2 + 2x_1 + x_3x_2 - 2x_3x_1 - 2x_2x_1$
XOR	$x_3 = x_1 \oplus x_2$	$5 + x_3 + a_2 - a_3 +$ $+x_1a_1 - x_1a_2 - x_1a_3 - x_2a_1 - x_2a_2 - x_2a_3 + x_3a_2 - x_3a_3$
NOT	$x_2 = \neg x_1$	$1 + x_1x_2$

Table 2.2: QUBO encoding of the basic logic gates. The QUBO polynomial is at its minimum value when the relation between variables is true. Notice how implementing the XOR gate requires adding two ancillary variables.

[1]Usually QUBO problems are stated on variables in $\{0, 1\}$ rather than $\{-1, 1\}$, which is an equivalent formulation

D-Wave's machine has been demonstrated to have better performance than simulated annealing for certain random QUBO problems [59], and has been used for solving traffic optimization [73] and quantum simulation of material properties [57]. Using the naive conversion from SAT to QUBO instead yields QUBO problems that are not suited to the annealer architecture, thus so far naively-encoded circuits are too large or are very simple for a traditional SAT solver.

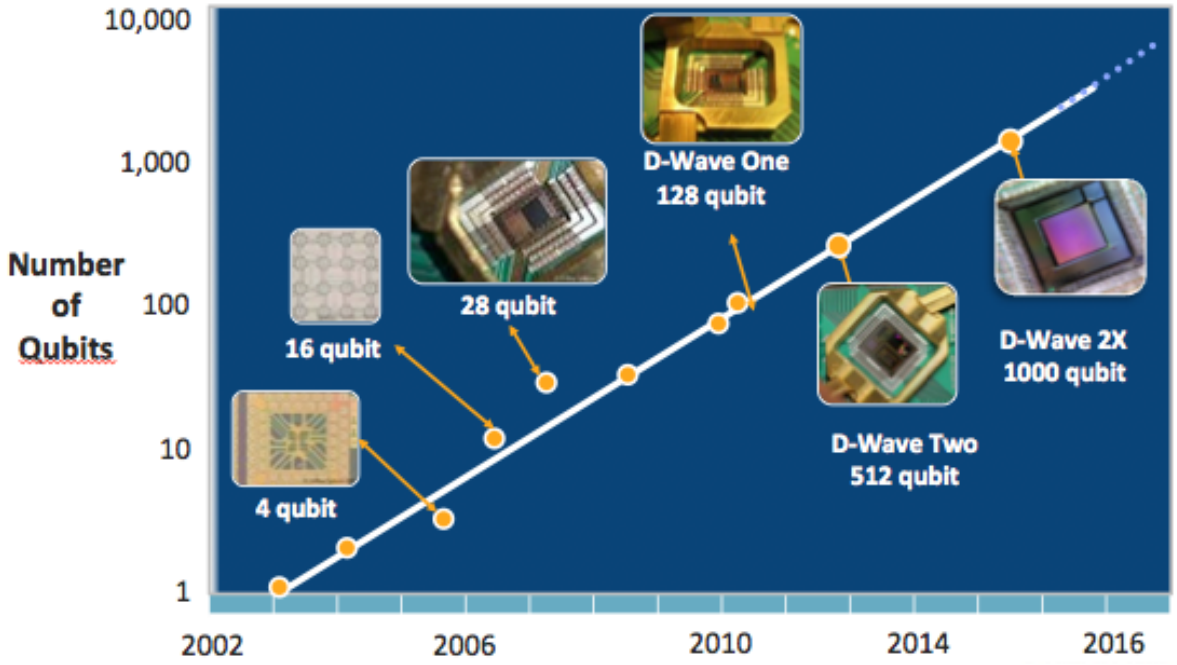


Figure 2.4: D-Wave annealers growth over the years. Courtesy of D-Wave Systems Inc.

The number of available qubits in D-Wave's quantum annealers is growing in a Moore-like fashion. Figure 2.4 shows the history of D-Wave machine in the last years and illustrates the phenomenon. This suggest an explosive growth of computing power considering that the number of qubits grows exponentially. In the latest years D-Wave put effort into improving the architecture rather than just increase the number of qubits. The result is a newer, better-connected architecture called Pegasus. We will see in the following chapters the implications of having this improved architecture.

2.6 Issues in Encoding for Quantum Annealers

Quantum annealers are still hard to build and operate at large scale. While naive conversion from circuit to QUBO is straightforward, the result is cannot be passed to the quantum annealer directly. This is because so far:

- The number of available qubits is limited.
- The number of couplings is limited.
- Noise and control precision limit the chances of success.

The number of available qubits is limited. Even with 2000+ qubits, the size of problems that can be solved is still small. This is because the majority of qubits in quantum annealing has to be used to encode the problem, unlike circuit models where the number of qubits represent the input width and the circuit size are two different metrics for complexity. Consider a boolean circuit that we want to check for satisfiability. In Grover search we need enough gate to set up a superposition and run the circuit reversibly, and while this is costly we can reuse qubits for intermediate values. When using quantum annealing we need to encode the full circuit into a Ising model. Thus a deep circuit using many levels with relatively few bits at a time (for example, a cryptography primitive with a limited internal state and many computation rounds) will require less qubits (ignoring the space requirements of gates).

The number of couplings is limited. For a complete graph, the number of possible couplings grows to the square of the number of qubits. While difficult problems are not necessarily dense, a few qubits tend to have high degree. In practice, each qubit can be coupled to a fixed low number of neighbors, and the number of available couplings scales with the square

root of the number of qubits. The problem encoding process then has to accommodate for the couplings that are available. In general, there is a measure that correlates directly with problem complexity[39], called **tree-width**. It is a measure of similarity to trees and will play an important role in the encoding process. In Chimera and Pegasus, tree-width grows linearly with the number of qubits.

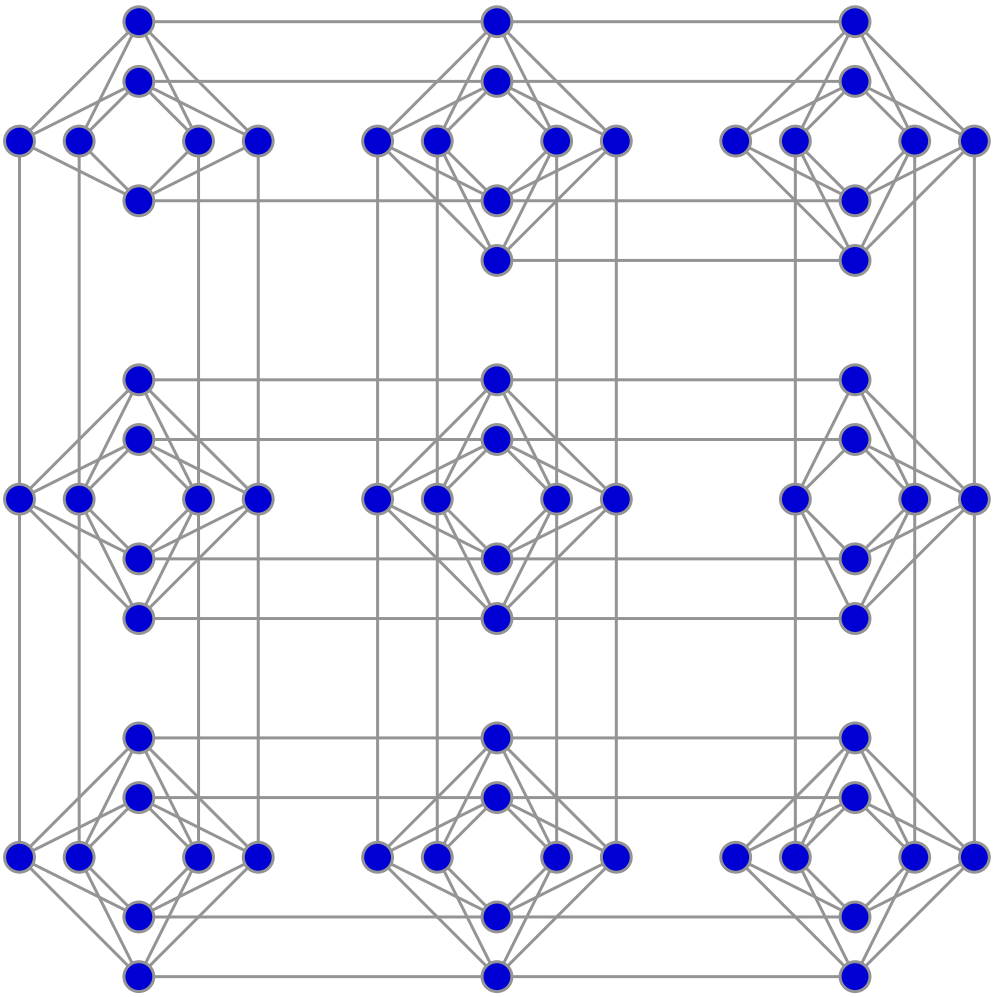


Figure 2.5: A small Chimera annealer with missing qubits. A 3-by-3 grid of Chimera tiles, where the top-left and middle-right tiles have a damaged/unusable qubit.

Noise and control precision limit the chances of success. Whereas the D-wave machine is close to 0 K and heavily shielded, noise can still cause performance degradation. In annealing the ground state of the annealer encodes the best solution of our problem while higher energy states are less useful. Thus we wish to have energy gaps between solution and non-solutions that are as large as possible. Furthermore, obviously there are range bounds and precision limits on the biases and couplings.

2.7 Goals

The concept of intrinsic greater power of quantum computers is generally called **quantum supremacy**. Proving quantum supremacy with a real device is a major goal of quantum computing research. To test for it we need problems that are impossible for standard computers and easy for quantum ones. CSP and SAT problems are very general problems, and tend to become very hard even for small sizes. SAT problems have the potential to be small enough to fit in the hardware but are still very hard for traditional computers.

D-Wave's quantum annealers have proven their strength for certain random QUBO problems that match their architecture. Exploiting the annealer for generic problems is less straightforward due to the need of encoding. Quantum annealing hardware is expected to grow in a Moore-like fashion, so it will eventually reduce the overhead, but we want to exploit in the best way the hardware we have now or in the near future, and possibly to solve interesting problems with it. My research goal is to pick a NP-hard problem and convert into an Ising problem in an effective and efficient way.

Effective means that the encoding process is capable to produce encodings that fit in the available hardware, even when the input problem is

considerably big/difficult.

Efficient means that the encoding process is reasonably fast to perform, at least it has to be faster than actually trying to solve the problem with a standard computer.

First we will lay down a theoretical framework to approach the encoding problem. We will frame it as a logic problem. This framework will become the basis for the use of SMT solving techniques. We will then outline the encoding strategy and describe in details each step. Then I will discuss how to decompose the encoding process and how to perform each step in software. After the process to encode a problem into a single quantum annealing problem, we need to consider how to exploit quantum annealing. In practice we need to decompose our problem into sub-problems that are solvable by the hardware and reasonably hard for traditional computers. In later chapters we will see how it can be done. Finally we will perform some preliminary evaluation to test the consistency of the approach.

Chapter 3

Background

3.1 Quantum computing

We will start with a quick survey on quantum computing. Whereas the encoding process can consider the actual computation as a black box process, the theory behind quantum computation is useful to justify the context of the thesis and it can provide useful tools to interpret the results. As the survey will be lacking in depth, the reader is invited to refer to Quantum Computation and Quantum Information[74] for further details.

3.1.1 Quantum mechanics

The theory of quantum mechanics is based on the theory of complex linear algebra. In this survey a basic knowledge of linear algebra will be necessary, but most of the concepts will be explained as they appear. For the purpose of the thesis the survey will assume only the finite dimensional spaces, as they represent quantum system with discrete states. Generalizing results from finite-dimensional to infinite-dimensional spaces is not trivial, but most of the assertions in this survey are equally valid in the infinite-dimensional case.

The theory of quantum mechanics can be stated with a series of four

basic postulates. These postulates will serve as a framework to write and reason about quantum phenomena. The first postulate defines the state of a quantum system:

Postulate 1. *Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system's state space.*

The basis of the Hilbert space will represent the set of possible outcomes, i.e. the post-measurement classical states that we can observe. For example, a system with two possible states (i.e. a qubit), the state of the system will be represented as a two dimensional vector with two basis vector that we will call, using the ket-notation, $|0\rangle$ and $|1\rangle$. We call a **pure state** a unit vector (or, equivalently, a ray) in the Hilbert space that represents a possible state of the system. We consider only unit vectors as states as possible outcomes depend only on the relative amplitudes and phases between vector components.

A pure state represents a perfectly known system; We can model uncertainty over the quantum state with the so-called **mixed states**. A mixed state is a distribution over superposition of states. It can be represented simply as a density operator, a convex combination of multiple projective operators. Thus, given possible states $|\Psi_i\rangle$ with probabilities p_i, the mixed state ρ of the system is:

$$\rho = \sum_i p_i \left|\Psi_i\right\rangle \left\langle\Psi_i\right| \tag{3.1}$$

Two observations can be done on this formula. First, the density matrix for a pure state is simply its projection operator $|\Psi\rangle \langle\Psi|$. Then, in the same way that $\sum_i p_i = 1$, we have that the trace of the density operator $tr(\rho)$ is 1.

The second postulate describes how a quantum system evolves over time:

Postulate 2. *The time evolution of the state of a closed quantum system is described by the Schrödinger equation,*

$$i\frac{d\left|\Psi\right\rangle}{dt} = H\left|\Psi\right\rangle \tag{3.2}$$

H is a fixed Hermitian operator known as the Hamiltonian of the closed system. Then, the evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\Psi_1\rangle$ of the system at time t_1 is related to the state $|\Psi_2\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t1 and t2.

$$\left|\Psi_2\right\rangle = U\left|\Psi_1\right\rangle \tag{3.3}$$

Recall that an Hermitian operator is an operator H that is its own dual ($H = H^\dagger$) and always has a spectral decomposition ($H = \sum_i e_i \left|E_i\right\rangle \left\langle E_i\right|,\ e_i \in \mathbb{R}$), while a unitary operator is a operator U such that $U^\dagger U = I$. The spectral decomposition of the Hermitian matrix in Schrödinger equation has an important interpretation. The eigenstates $|E_i\rangle$ are called **stationary states**, and the eigenstate with lowest eigenvalue e_0 is called **ground state**. A direct consequence of the spectral decomposition is that quantum states evolve using unitary matrixes. If the quantum state is a unit vector, a quantum computer instruction is a unitary operator.

We can easily generalize system evolution on a mixed state ρ given the unitary operator U:

$$\rho' = U\rho U^\dagger \tag{3.4}$$

Notice how the postulate refers to an isolated system and a Hamiltonian fixed in time. We generally want to interact with a system and vary its

Hamiltonian. For many such systems we can write a time-varying Hamiltonian, where the effect of the environment on the system is represented with a changing Hamiltonian $H(t)$:

$$i\frac{d\left|\Psi\right\rangle}{dt} = H(t)\left|\Psi\right\rangle \tag{3.5}$$

The third postulate describes how the quantum world interacts with the classical world, i.e. how measurements are made.

Postulate 3. *Quantum measurements are described by a collection* $\{M_m\}$ *of measurement operators such that:*

$$\sum_m M_m^\dagger M_m = I \tag{3.6}$$

These are operators acting on the state space of the system being measured. The index m *refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is* $\left|\Psi\right\rangle$ *immediately before the measurement then the probability that result* m *occurs and the state of the system after the measurement are given by:*

$$p(m) = \left\langle\Psi\right| M_m^\dagger M_m \left|\Psi\right\rangle \tag{3.7}$$

$$\left|\Psi_m\right\rangle = \frac{M_m\left|\Psi\right\rangle}{\sqrt{p(m)}} \tag{3.8}$$

The postulate is the most general specification of measurement. In the simplest case, called **projective measurement**, the M_m operators can be mapped into projective operators for a particular basis of the Hilbert space:

$$M_m^\dagger M_m = P_m = \left|\phi_m\right\rangle\left\langle\phi_m\right| \tag{3.9}$$

In the case that the measurement outcomes m are real-valued, we can represent the measurement with a single matrix M, called **observable**.

This form allows us to represent concisely the expected value of the measurement $\mathbb{E}_\Psi[M]$.:

$$M = \sum_m m \left|\phi_m\right\rangle \left\langle\phi_m\right| \tag{3.10}$$

$$\mathbb{E}_\Psi[M] = \langle\Psi|M\Psi\rangle \tag{3.11}$$

We can generalize the measurement of a mixed state ρ and get its probability and post-measurement state:

$$p(m) = tr(M_m^\dagger M_m \rho) \tag{3.12}$$

$$\rho_m = \frac{M_m^\dagger \rho M_m}{p(m)} \tag{3.13}$$

Notice how we have non-determinism in measurement. Even in a fully-known pure state the outcome of a measurement can be random. Furthermore, measurement is a destructive operation, as the original state is modified, and only projective measurements are repeatable, i.e. applying the same measurement twice yields the same result.

The last postulate describes the composition of quantum states:

Postulate 4. *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is prepared in the state* $|\Psi_i\rangle$*, then the joint state of the total system is* $|\Psi_1\rangle \otimes |\Psi_2\rangle \otimes ... \oplus |\Psi_n\rangle$.

For example the composition of two one-qubit systems labeled a and b, each using the base $|0\rangle$, $|1\rangle$, is a Hilbert space with basis:

$$|0_a\rangle \otimes |0_b\rangle = |00\rangle$$
$$|0_a\rangle \otimes |1_b\rangle = |01\rangle$$
$$|1_a\rangle \otimes |0_b\rangle = |10\rangle$$
$$|1_a\rangle \otimes |1_b\rangle = |11\rangle$$

It follows then that the number dimensions of a discrete Hilbert space doubles for each added qubit. Thus, the representation of a quantum state in a classical computer needs an amount of space that is exponential to the number of qubits. This is one of the reasons for distinction from quantum and classical computing, and what makes many-body matter simulations hard to simulate and predict.

Another important consequence is that a two-qubit system can be put in entangled state $|\Psi\rangle = |00\rangle + |11\rangle$. This state is peculiar because the system cannot be expressed anymore as the composition of two single-qubit systems: this is an **entangled state**.

3.1.2 Qubits, quantum gates, adiabatic computing

The basic unit of quantum information is a **qubit**, a discrete quantum system with two possible states usually called $|0\rangle$ and $|1\rangle$. While a bit is an element that can be in one of two states, a qubit is in a superposition of these two states. A qubit state can be represented as a point on the **Bloch sphere**. A pure state is a point on the surface while a mixed state as a point inside its volume. Notice that this representation fails to generalize to multiple qubits.

Within the Bloch representation, unitary transformations are represented by 3d rotations and reflections of the sphere. Thus, while the only single-bit functions are identity and negation, we are able to perform several several quantum operations on a single qubit. The simplest example

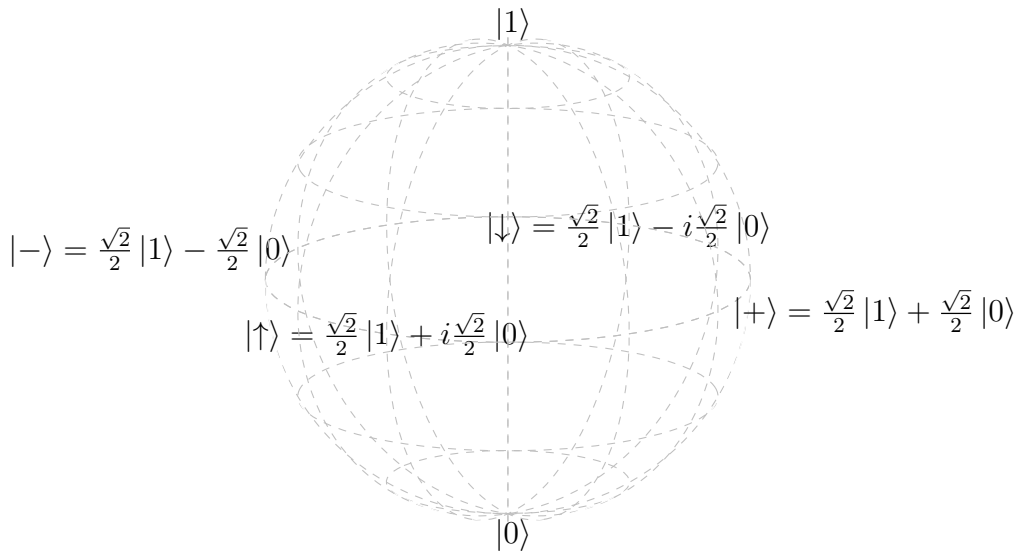


Figure 3.1: Representation of the state of a single qubit as a Bloch sphere. Any point on the surface of the sphere represents a pure state, and any point inside the sphere represents a mixed state.

is negation (Equation 3.14), other important operations are the Hadamard gate (Equation 3.15) and the $\pi/8$ half-phase gate (Equation 3.16).

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.14}$$

$$H = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.15}$$

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{3.16}$$

The most important multiple qubit operation is the controlled-not gate (CNOT) where the second bit is negated if the first is 1. The CNOT, Hadamard and $\pi/8$ half-phase gate are a universal set of gates [74]. This means that we can approximate any quantum transformation on an arbitrary number of qubits using only these gates. Figure 3.2 shows the canonical representations for these common gates. These gates are funda-

mental for quantum computing theory, and by virtue of their universality they can represent any computation done by quantum systems, but as we will focus on quantum annealing, they will be of limited use.

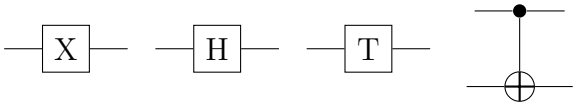


Figure 3.2: Standard representation for the basic quantum gates (from left to right: negation, Hadamard, half-phase and controlled not).

3.2 Adiabatic Quantum Computing and quantum annealing

3.2.1 Adiabatic Quantum Computing

The adiabatic quantum computing approach relies on the adiabatic quantum theorem. It is a direct consequence of the time-dependent Shrödinger equation. The adiabatic theorem states that when the Hamiltonian changes slowly enough over time, a quantum system that starts in the initial ground state ends in the final ground state.

Theorem 1. *Given a quantum system at ground state* $\left|\Psi_0^i\right\rangle$ *and the time-varying Hamiltonian* $H(t) = H_i(T-t)+H_f t$*, given large enough transition time* T *the system will stay in the ground state up to the final state* $\left|\Psi_0^f\right\rangle$.

As stated above, the theorem does not specify any explicit bound on T. In general, if g_{min} is the minimum energy gap between the ground state $|\Psi_0(t)\rangle$ and the other stationary states the required time will be proportional to the inverse of the square of g_{min}, but there are more rigorous bounds on the transition time. AQC can efficiently simulate quantum gates and vice versa: A quantum circuit can be encoded as the ground state of a Hamiltonian, and the AQC Hamiltonian can be integrated to a unitary

operator that can be approximated by quantum gates. The reader that is interested in a thorough exposition is suggested to consult [3].

The quantum adiabatic theorem holds for a system that is already at the ground state. If we switch point of view to thermodynamics, this state is at the absolute zero temperature. In theory, in an adiabatic quantum system the quantized energy gaps theoretically ensures that a higher temperature requires a discrete amount of energy. In thermodynamics when the temperature is not zero we have a process called annealing. Both in classical and quantum annealing the probability of a certain state is governed by Boltzmann statistics, stated in Theorem 2.

Theorem 2. *A thermodynamic system at equilibrium can be found in the state x with the following probability:*

$$p(x) = \frac{1}{Z} e^{\frac{E(x)}{kT}} \tag{3.17}$$

where T is the temperature of the system, and Z is the partition function: $Z = \sum_x e^{\frac{E(x)}{kT}}$

The Boltzmann distribution is used widely to calculate properties of thermodynamics systems. The most relevant observation for the thesis is that when the temperature is high every state is equi-probable while when the temperature is small low energy states are more probable. Simulated annealing consists essentially in simulating a system moving toward equilibrium while lowering the temperature. A low energy state consists in a solution with a low cost according to the problem constraints.

Boltzmann statistics are valid for classical Ising model and quantum systems at high temperature and low concentrations, and so are limited in explaining quantum annealer, but are conceptually important when considering open quantum systems that tend toward an equilibrium state [5].

3.2.2 Ising models

So far the Hamiltonian function that represent the energy landscape has been not yet defined. The quantum annealers that will be used in this thesis will have an Ising energy model. The Ising model is one of the most important models in statistical physics. It has been used to study various phenomena in matter, like magnetization.

In the Ising model the particles/elements can be only in two states, -1 or 1. The Hamiltonian is defined as a second degree real polynomial on binary variables (Equation 3.18). In this polynomial θ_i represent biases of a single element, while θ_{ij} represent the effect of interaction between pairs of elements.

$$H(\underline{\mathbf{z}}) = \sum_i \theta_i z_i + \sum_{ij} \theta_{ij} z_i z_j \tag{3.18}$$

The problem of finding the minimum of a polynomial over binary variables is called **quadratic unbounded binary optimization (QUBO)**. A QUBO problem ask, given a quadratic polynomial with binary variables what is its minimum value assignment. In the QUBO literature it is usually assumed that variables takes values in $\{0, 1\}$ while Ising model variables take values in $\{-1, 1\}$, but the two representations are equivalent and conversion is trivial. If the state of the system is expressed as a binary vector $\underline{\mathbf{z}} \in \{0, 1\}^n$, the QUBO problem derived by an Ising model can be expressed also as a quadratic form (Equation 3.19). In this case the parameters are represented with a single matrix Θ.

$$H(\underline{\mathbf{z}}) = \underline{\mathbf{z}}^T \Theta \underline{\mathbf{z}} \tag{3.19}$$

3.2.3 Induced graphs and their properties

Ising models are generally classified by the topology of the interaction. Given a second degree polynomial we can define an **induced graph** where vertices are variables/qubits, edges are non-zero second degree terms or available couplings. The properties of the induced graph are important in terms of complexity of the problem and useful for encoding. Given a graph G, a graph minor is a graph where an edge or a vertex is removed, or two vertices are merged. When G is an induced graph, setting the value of a variable or forcing equivalence between two variables is equivalent to removing or merging vertices.

The shape of the induced graph of a QUBO problem affects its hardness. We have seen that QUBO is a NP-hard problem in the general case, but it is not trivial that an Ising model with a certain topology is NP-Hard as well, and indeed a planar graph with no biases are tractable [7]. Later we will see various method to reduce general QUBO problems into problems that have the induced graph of the quantum annealer hardware.

In later chapters we will make use of symmetries in induced graphs. A permutation $\sigma : V \to V$ is an **automorphism** of a graph G if relabeling its vertices with σ produces the same graph: $\sigma(G) = G$. Automorphisms form a group (where the identity function is the identity and function composition is the operator), called $Aut(G)$. When the group is associated with an action $\phi(\sigma, x) : (Aut(G), X) -> X$ (see [43]) the group orbit of x is the set of elements of X that can be reached from x: $G \times x = \{\forall g \in Aut(G) : \phi(g, x)\}$. Group orbits form a partition of X, and thus form an equivalence relation on it.

Another important graph property is **tree-width**. The tree-width encodes the smallest node clustering that yields a tree. Given a graph $G = (V_G, E_G)$ a tree decomposition is a tree T composed of nodes $X_1, ..., X_n$

such that:

- $X_1, ..., X_n$ are subsets of V_G, and $\bigcup X_i = V_G$.
- All the X_i that contains a vertex v form a connected subtree of T.
- For every edge $(v, w) \in E_G$ there exists a X_i that contains both v and w.

The width of the decomposition is the maximum size of the X_i minus 1, and the tree-width of a graph is the minimum width for all possible decompositions of a graph. Figure 3.3 illustrates a simple tree decomposition of width 2. Tree-width is an important property in computer science: the complexity of a dynamic programming depends on the tree-width of the problem dependencies, and CIRCUIT-SAT complexity is linear for a circuit of fixed tree-width. Tree-width is NP-Hard to compute but is easy to approximate using heuristics methods.

3.2.4 D-Wave machine

The quantum annealer that is assumed to be used during this thesis is produced by D-Wave Systems. The basic unit of their machine is the Superconducting QUantum Interference Device (SQUID). The SQUID consists in a electrical circuit containing Josephson junctions, a circuit component that exhibits known quantum effects when exposed to a transversal magnetic field. At low enough temperatures, the current in the circuit flows in a superposition of clockwise and counterclockwise direction. These two states form a qubit. Figure 3.4 shows a simple schema of the device. The user sets the desired Ising Hamiltonian by controlling the external magnetic field. To perform an anneal run, the Hamiltonian is slowly turned in a adiabatic fashion from a standard initial value to the desired desired energy landscape.

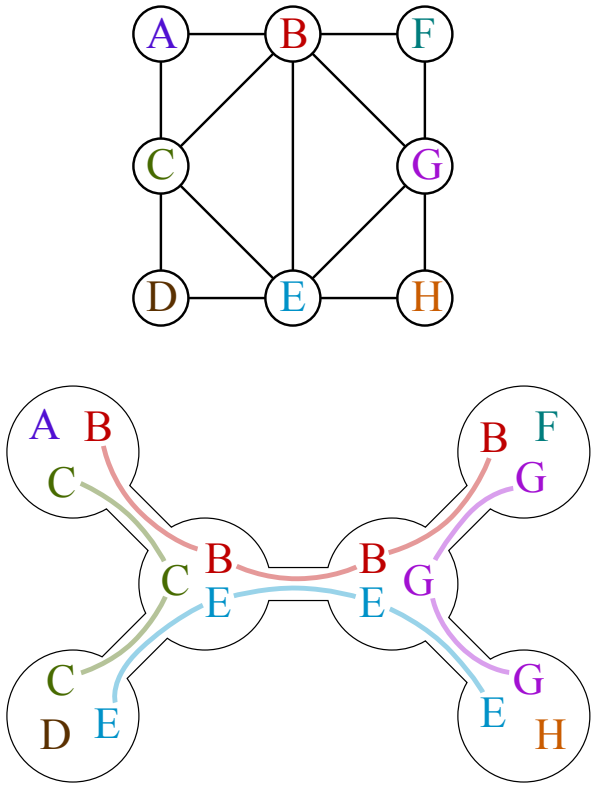


Figure 3.3: Tree decomposition of a graph with tree-width 2. A dynamic programming task that depends only on local interactions/edges can traverse the tree decomposition to choose sub-graphs for partial computations. The tree-width is a measure of how many nodes are shared between sub-graphs.

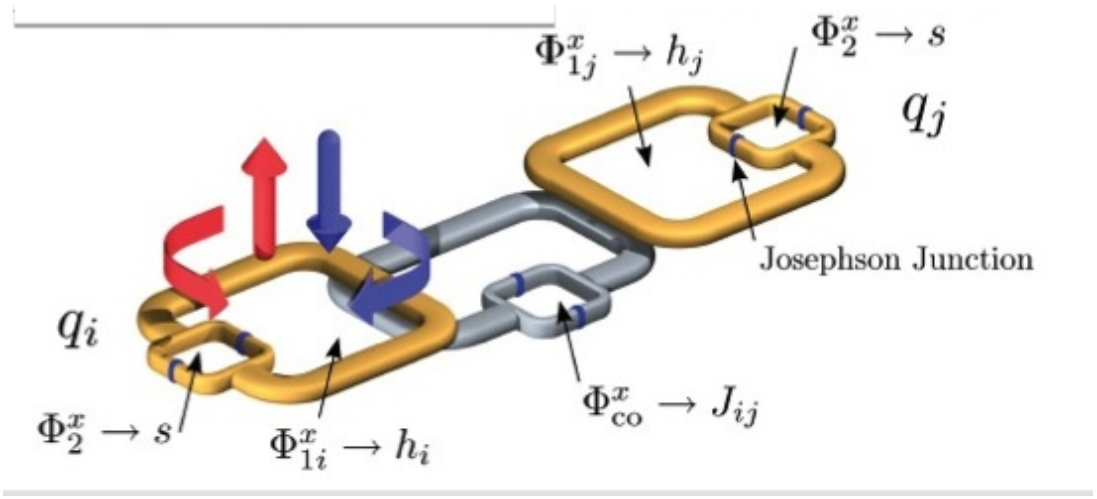


Figure 3.4: Illustration of a SQUID qubit. The user sets the various biases h_i and couplings J_{ij} by tuning the various magnetic fields Φ.. Courtesy of D-Wave Systems Inc.

The latest annealer is called D-Wave 2000Q, announced in 2017. Since October 2018 is available to the public through a cloud service.[1] The D-Wave 2000Q machine has 2048 qubits and 5600 couplers in a regular pattern called **Chimera topology**. This topology consists in groups of 8 tightly connected qubits in a bipartite graph called cell, and a grid of cells where 4 qubits have vertical parallel connections and 4 qubits have horizontal parallel connections. Figure 3.5 shows the induced graph of a D-Wave 2000Q Chimera chip.

The Chimera topology is a bipartite graph, thus there are no cliques. This topology contains as a minor complete graphs and complete bipartite graphs, we will see then how to encode complete graphs into chimera topology. Thanks to the cell separation encoded problems often have a clear cut distinction between functional units, where complex relationship are expressed using the dense connection within the chip and couplings between cells are used to transfer information.

While all the published work so far has been on the Chimera topol-

[1] available at `https://cloud.dwavesys.com/leap/`

ogy, in 2018 D-Wave divulged details about a newer topology, the so called **Pegasus topology**. This topology is more complex and more densely connected, paving the way for more efficient encodings. Figure 3.6 illustrates a small example of a Pegasus 4, containing the equivalent of a square of 4 by 4 tiles. In this newer topology there is no more a clear decomposition into tiles, rather than that clusters of qubit are connected in a more interleaved fashion. Furthermore, the Pegasus architecture adds a new kind of connection between two horizontal or vertical qubits. This allows the new architecture to have 3 and 4 cliques, that were absent from Chimera.

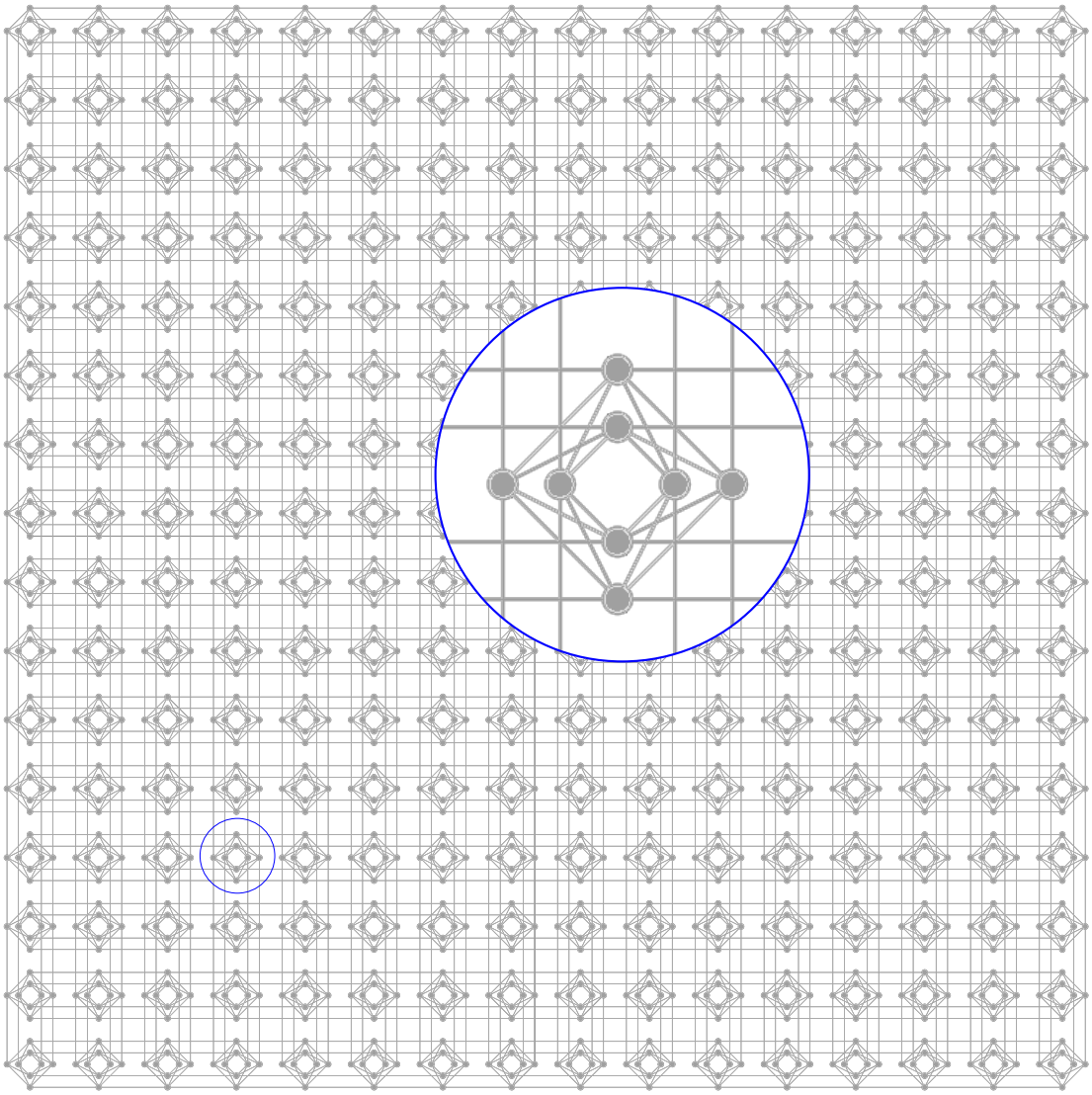


Figure 3.5: 16×16 Chimera topology with a detail on a single tile, circled in blue.

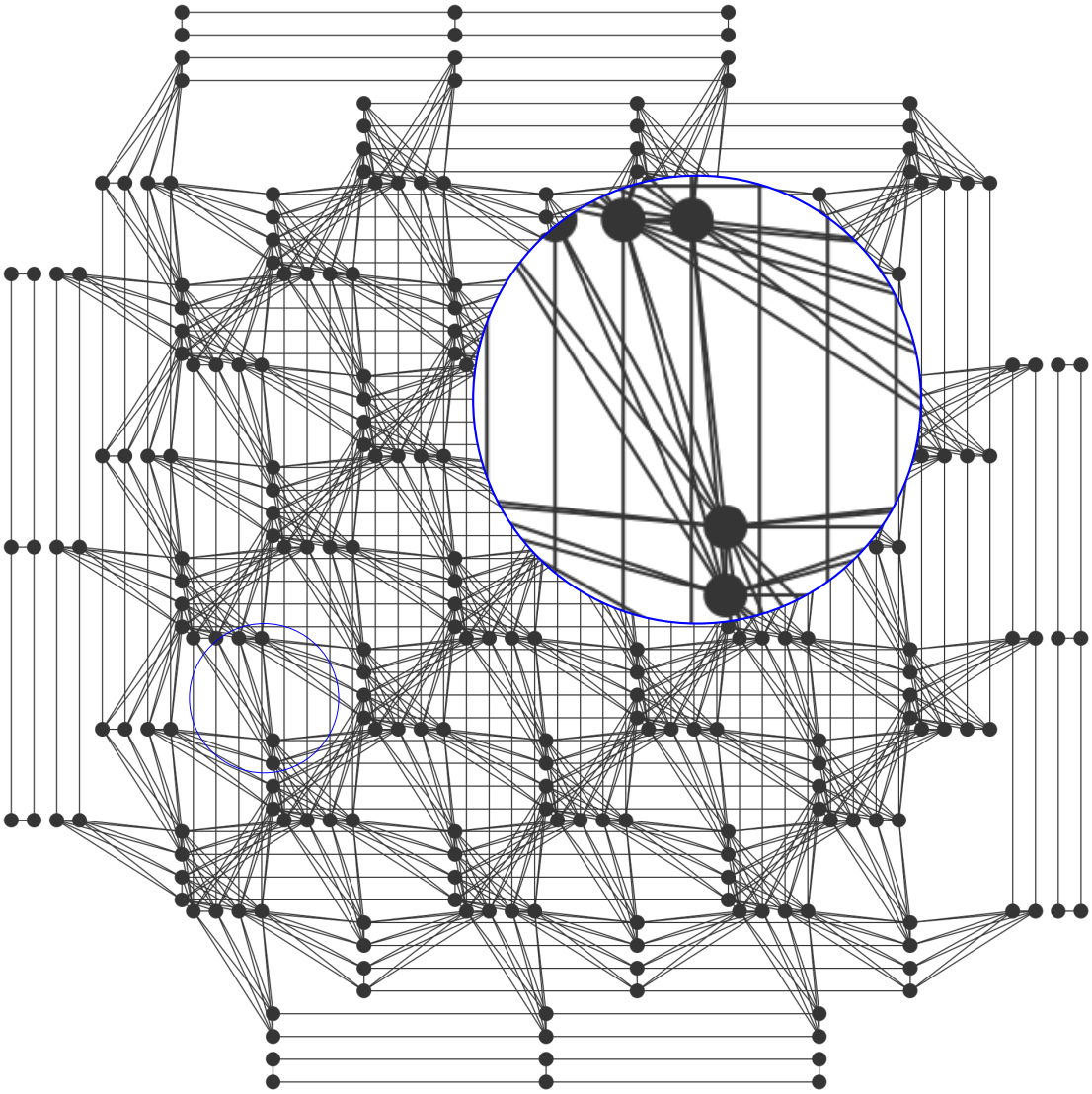


Figure 3.6: Small pegasus topology, with focus on a single 4-clique. circled in blue.

3.3 SAT, MaxSAT, SMT and OMT

3.3.1 Basics

In the following we recall the main concepts of the basic syntax, semantics and properties of Boolean and first-order logic and theories. We refer the reader to [19, 65, 62, 9, 84] for more details.

Boolean logic deals with formulas over Boolean variables, variables that can assume the value true ($\top$) or false ($\bot$). Given some finite set of Boolean variables, or Boolean atoms, $\underline{\mathbf{x}}$ the language of Boolean logic ($\mathcal{B}$) is the set of formulas containing the atoms in $\underline{\mathbf{x}}$ and closed under the standard propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ (respectively called: NOT, AND, OR, IMPLY, IFF, XOR).

The AND ($\wedge$) operation is also called **conjunction**, OR ($\vee$) is also called **disjunction** and NOT ($\neg$) is also called **negation**. All other connectives can be defined in terms of disjunction, conjunction and negation. The meaning of these connectives, i.e. the value of the formula given the value of its variables, can be defined using truth tables. A **literal** is an atom, x (positive literal) or its negation $\neg x$ (negative literal). We implicitly remove double negations: e.g., if l is the negative literal $\neg x_i$, then by $\neg l$ we mean x_i rather than $\neg\neg x_i$.

A formula is in **negative normal form (NNF)** if only AND and OR are used, and negation appears only in negative literals. Every formula can be converted into NNF using deMorgan's theorems. A **clause** is a disjunction of literals. A formula is in **conjunctive normal form (CNF)** if it is written as a conjunction of clauses. Conversely, a **cube** is a disjunction of literals and a formula is in **disjunctive normal form (DNF)** if it is written as a disjunction of cubes.

An assignment $\underline{\mathbf{x}}$ **satisfies** $F(\underline{\mathbf{x}})$ iff it makes it evaluate to true. If so, $\underline{\mathbf{x}}$ is called a **model** for $F(\underline{\mathbf{x}})$. A formula $F(\underline{\mathbf{x}})$ is **satisfiable** iff at least

one truth assignment satisfies it, **unsatisfiable** otherwise. $F(\underline{\mathbf{x}})$ is **valid** iff all truth assignments satisfy it. $F_1(\underline{\mathbf{x}}), F_2(\underline{\mathbf{x}})$ are **equivalent** iff they are satisfied by exactly the same truth assignments.

A formula $F(\underline{\mathbf{x}})$ which is not a conjunction can always be decomposed into a conjunction of smaller formulas $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ by means of **Tseitin's transformation** [94], as in Equation 3.20, where the F_is are simple subformulas which decompose the original formula $F(\underline{\mathbf{x}})$, and the y_is are fresh Boolean variables each labeling the corresponding F_i.

$$\begin{aligned} F(\underline{\mathbf{x}}) &= F_m(F_{m-1}(...(F_1(\underline{\mathbf{x}})))) \\ F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}}) &\stackrel{\text{def}}{=} \textstyle\bigwedge_{i=1}^{m-1}(y_i \leftrightarrow F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i)) \wedge F_m(\underline{\mathbf{x}}^m, \underline{\mathbf{y}}^m) \end{aligned} \tag{3.20}$$

Tseitin's transformation guarantees that $F(\underline{\mathbf{x}})$ is satisfiable if and only if $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ is satisfiable, and that if $\underline{\mathbf{x}}, \underline{\mathbf{y}}$ is a model for $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$, then $\underline{\mathbf{x}}$ is a model for $F(\underline{\mathbf{x}})$. For this reason it is used recursively for efficient CNF conversion of formulas [94].

A **quantified Boolean formula (QBF)** is an extension over the aforementioned Boolean formulas. It is defined inductively as follows: a Boolean formula is a QBF; if $F(\underline{\mathbf{x}})$ is a QBF, then $\forall x_i F(\underline{\mathbf{x}})$ and $\exists x_i F(\underline{\mathbf{x}})$ are QBFs. QBFs can be converted to Boolean formula through **Shannon's expansion**: $\forall x_i F(\underline{\mathbf{x}})$ is equivalent to $(F(\underline{\mathbf{x}})_{x_i=\top} \wedge F(\underline{\mathbf{x}})_{x_i=\bot})$ and $\exists x_i F(\underline{\mathbf{x}})$ is equivalent to $(F(\underline{\mathbf{x}})_{x_i=\top} \vee F(\underline{\mathbf{x}})_{x_i=\bot})$.

3.3.2 And-Inverter Graphs

Any Boolean function can be represented as an **And-Inverter graph**. An AIG, as the name suggests, is composed by 2-input AND gates and negations. More precisely, an AIG for $F(\underline{\mathbf{x}})$ is a **directed acyclic graph (DAG)** D on vertex set $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{g}} = (x_1, \ldots, x_n, g_1, \ldots, g_m)$ with the following properties:

1. Each x_i has no incoming edges and each g_k has 2 incoming edges, and there is a unique g_o with no outgoing arcs (the **primary output**).

2. Each edge $z \to g$ is labelled with a sign $+$ or $-$ indicating whether or not z should be negated as an input to g; define a literal $l_i(z) = z$ for an edge with sign $+$ and $l_i(z) = \neg z$ for an edge with sign $-$.

3. For each node g_k with edges incoming from z_1 and z_2, there is an AND function $A_k(g_k, z_1, z_2) = g_k \leftrightarrow l_k(z_1) \wedge l_k(z_2)$, such that

$$F(\underline{\mathbf{x}}) \leftrightarrow \bigwedge_{k=1}^{m} A_k(\underline{\mathbf{z}}) \wedge (g_o = \top). \tag{3.21}$$

If is $F(\underline{\mathbf{x}})$ is in CNF form, we can trivially construct an AIG by rewriting each OR clause as an AND function using De Morgan's Law, and then rewriting each AND function with more than 2 inputs as a sequence of 2-input AND functions.

Example 1. *The function*

$$F(\underline{\mathbf{x}}) = x_1 \wedge x_2 \wedge \neg x_3$$

is represented by both of the And-Inverter Graphs in Figure 3.7.

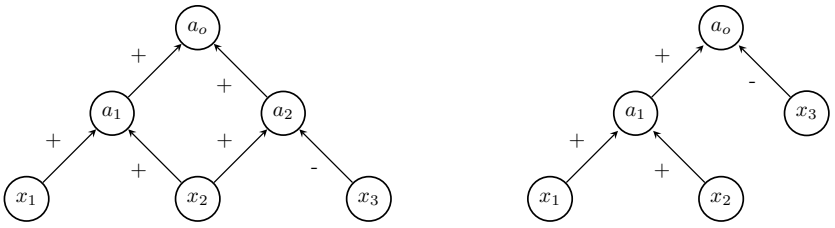


Figure 3.7: Two And-Inverter Graphs representing the function $F(\underline{\mathbf{x}}) = x_1 \wedge x_2 \wedge \neg x_3$.

Let G be an AIG with a node z. A **cut** C of z is a subset of vertices of G such that every directed path from an input x_i to z must pass through C. The sub-graph of G composed by all vertices that are crossed by any

path from C to z is effectively a AIG representation of z as a function of C, since the Boolean value of z is determined completely by C. The equivalent function is the **Boolean function of z represented by C**. Cut C is **k-feasible** if $|C| \leq k$ and **non-trivial** if $C \neq \{z\}$. For fixed k, there is a simple linear-time algorithm to enumerate all k-feasible cuts in an AIG. Starting from the inputs $\underline{\mathbf{x}}$ to the primary output, we can traverse the graph to list the k-feasible cuts of node a_i by combining k-feasible cuts of a_i's two inputs.

3.3.3 SAT

Propositional Satisfiability (SAT) is the problem of establishing whether an input Boolean formula is satisfiable or not. SAT is a NP-complete problem [37]. Not all SAT problem instances are hard: some restricted versions, such as 2-SAT and HORN-SAT, are tractable.

Whereas it is implausible to find an algorithm that solves SAT problems beyond a certain size in the worst case, a large amount of effort and ingenuity has been put into speeding up resolution in the average case. Every year a competition between the state-of-the-art solvers is held [1]. In this competition newer techniques and approaches are held in comparison. Efficient SAT solvers are publicly available, most notably those based on **Conflict-driven clause-learning (CDCL)** [65] and on **stochastic local search** [64]. Most solvers require the input formula to be in CNF, implementing a CNF pre-conversion based on Tseitin's transformation (Equation 3.20) when this is not the case. See [19] for a survey of SAT-related problems and techniques.

The most effective SAT solvers are based on CDCL. CDCL solvers are able to prove the unsatisfiability of a formula, thus they are **complete** solvers. CDCL solvers try partial assignments until a solution is found or when a clause becomes unsatisfiable. In the latter case, a conflict analysis is

performed after which a new clause is learned. This learned clause reflects the latest decision done by the solver that is responsible for the conflict. With this clause the solvers avoids future visit to the same unsuccessful solution sub-space.

A different approach is to perform a random walk in the solution space. This is the stochastic local search approach. This approach is very successful on large random SAT problems but is not complete and thus it cannot prove the unsatisfiability of a formula. SLS solvers start from a random assignment and try to minimize the number of unsatisfied clauses. Various techniques are employed to maximize state exploration and to avoid loops. SLS techniques are closely related to simulated annealing approaches, though the latter are much more general.

3.3.4 MaxSAT

MaxSAT is an extension of SAT, where we ask what model satisfies the maximum amount of clauses of a CNF formula F (that is typically unsatisfiable, though a satisfiying model is a valid solution for a MaxSAT problem instance). It is generally more useful to consider extensions of MaxSAT, such as weighted MaxSAT and partial weighted MaxSAT. **Weighted MaxSAT** $\{\langle F_k, c_k\rangle\}_k$ is an version of MaxSat such that each clauese F_k of F is given a positive penalty $c_k \in \mathbb{R}^+$ if F_k is not satisfied, and an assignment minimizing the sum of the penalties is sought. **Partial Weighted MaxSAT** is a further extension of Weighted MaxSAT such that some clauses, called **hard constraints**, must be satisfied, so they have penalty $+\infty$.

MaxSAT solvers rely heavily on SAT solution techniques. Efficient MaxSAT solvers are publicly available (see, e.g., [62, 92]). MaxSAT solver can use CDCL SAT techniques by using **core-guided** search: when the SAT solver identifies a UNSAT core (i.e subset of clauses), the MaxSAT solver bounds the upper cost of the search. SLS solvers can be extended

to Weighted MaxSAT by keeping account of the clause penalty during the optimization search. SLS-based MaxSAT solver are penalized for Partial weighted MaxSAT, as optimal model search cannot be trivially confined to models that satisfy the hard constraints.

3.3.5 SMT and OMT

Satisfiability Modulo Theories (SMT) is another extension of SAT and a limited version of a first-order logic reasoning. It consists in checking the satisfiability of first order formulas in a background theory $\mathcal{T}$ or in a combinations of particular theories. SMT solving is focused on specific theories of interest, that generally have a specific decision algorithm.

For example, given $\underline{\mathbf{x}}$ as in the previous section and some finite set of rational-valued variables $\underline{\mathbf{v}}$, the language of the theory of **Linear Rational Arithmetic** ($\mathcal{LRA}$) extends that of Boolean logics with $\mathcal{LRA}$-atoms in the form $(\sum_i c_i v_i \bowtie c)$, c_i being rational values, $v_i \in \underline{\mathbf{v}}$ and $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$, forming linear algebraic expressions on the real numbers.

In the theory of **linear rational-integer arithmetic with uninterpreted functions symbols** ($\mathcal{LRIA}\cup\mathcal{UF}$) the $\mathcal{LRA}$ language is extended by adding integer-valued variables to $\underline{\mathbf{v}}$ ($\mathcal{LRIA}$) and **uninterpreted function symbols**. A n-ary function symbol $f()$ is said to be **uninterpreted** if its interpretations have no constraint, except that of being a function (congruence): if $t_1 = s_1$, ..., $t_n = s_n$ then $f(t_1, ..., t_n) = f(s_1, ..., s_n)$. For example, $(x_i \rightarrow (3v_1 + f(2v_2) \leq f(v_3)))$ is a $\mathcal{LRIA} \cup \mathcal{UF}$ formula. Notice that the notions of literal, assignment, clause and CNF, satisfiability, equivalence and validity, Tseitin's transformation, quantified formulas and so on extend trivially to $\mathcal{LRA}$ and $\mathcal{LRIA} \cup \mathcal{UF}$. **Satisfiability Modulo** $\mathcal{LRIA} \cup \mathcal{UF}$ **(SMT($\mathcal{LRIA} \cup \mathcal{UF}$))** [9] is the problem of deciding the satisfiability of arbitrary formulas on $\mathcal{LRIA} \cup \mathcal{UF}$. It is one of the most

important combination of theories and is extensively studied. Efficient SMT($\mathcal{LRIA} \cup \mathcal{UF}$) tools are available, including MathSAT5 [36].

SMT solvers rely on decision algorithms, one for each theory, and **theory combination** techniques to ensure the consistency of the model. Most modern SMT solver use lazy CDCL solving. During the CDCL search on the Boolean skeleton of the SMT formula, assertions on each theory are checked for consistency, and the theory solvers contribute to the conflict analysis.

Optimization Modulo $\mathcal{LRIA} \cup \mathcal{UF}$ (OMT ($\mathcal{LRIA} \cup \mathcal{UF}$)) [84] is yet another extension of SMT($\mathcal{LRIA} \cup \mathcal{UF}$). In OMT the goal is to search solutions which optimize some $\mathcal{LRIA}$ objective(s). While in MaxSAT each clause has attached a particular penalty cost, in OMT the cost is represented as a real-valued variable c. The focus on finding the minimal model makes the problem more complex but enables further optimizations. It is possible furthermore to have multiple cost variables c_i. In this case, generally a **Pareto-efficient** solution is wanted. A solution is Pareto-efficient if no single c_i can be improved without worsening other c_j costs. Efficient OMT($\mathcal{LRA}$) solvers, like OptiMathSAT [85], are available on the Internet.

Chapter 4

Related Work

In this chapter we will provide a brief overview of the literature pertaining quantum annealing and D-Wave's machines. Most of the literature is tangential to the SATtoIsing problem but provides a useful comparison. First the chapter will outline how different problems have been encoded into Ising problems, then describe different techniques to specifically make use of D-Wave hardware and finally will show some result reports on quantum annealing experiments.

4.1 Combinatorial Problems and CSP Encoding

There have been various previous efforts to map constraint satisfaction problems to Ising models [95, 80, 42, 77, 79, 75, 98, 17, 55]. Most of those mappings have been for specific constraints types, but some were more systematic.

The core theoretical framework used in this thesis appeared in [13], applied generic discrete optimization problems. The paper introduces the concepts that will be outlined in the next chapters. In particular it introduces the definition of penalty functions, the use of SMT solvers and the variable elimination reformulation, and problem embedding. It is focused on encoding specific constraints, in particular it provides a specific exam-

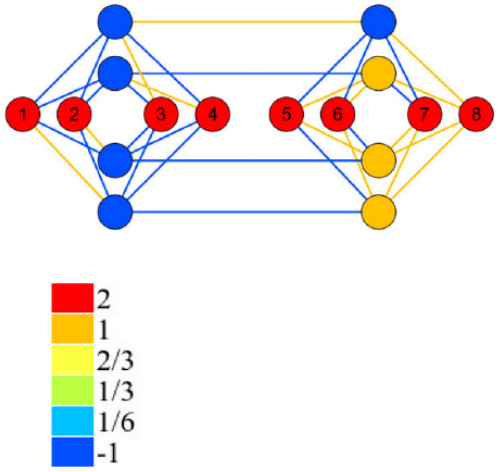


Figure 4.1: Encoding of an 1-in-8 CSP constraint found with a SMT solver, from [13].

ple, a parity check problem. Figure 4.1 shows the constraint used in the example, where exactly one out of 8 variables is $\top$.

A later paper by Bian et al. [14] applies the same approach to **fault analysis** of Boolean circuits. This problem consists in the following: given a Boolean circuit and a input-output pair, find the minimum number of gates that are faulty. Again, the paper uses an approach very similar to the one outlined in this thesis. Boolean circuits are encoded similarly but the goal of fault analysis is different. Thus the paper introduces an alternative definition of penalty function that is more suited to the task. Rather than just finding a satisfiable solution, there is an interest in fair sampling of possible solutions.

A paper by Lucas [63] (see also [34]) provides encoding of various NP-hard problems into Ising, among these is an encoding of the 3SAT problem. Given a graph G, the **maximal independent set (MIS)** problem consists in finding the greatest set X of vertices such that no edge (i, j) in G contains both ends in X: $i \notin X \vee j \notin X$. The paper provides the following Ising model encoding for the MIS problem, where $x_i = 1$ iff $x_i \in X$.

$$H = \sum_{i \in V_G} -x_i + \sum_{(i,j) \in E_G} 2x_i x_j$$

Furthermore, it reports a trivial encoding of 3SAT to MIS: for each 3-clause, add a 3-clique to G, where each node represents a literal. Then, for each node representing literal l add an edge to each node representing $\neg l$. If a solution exists where at least one literal per clause is in the MIS X, set that literal to $\top$; Otherwise, no satisfying assignment exists. This encoding of 3SAT suffers from low effectiveness: three qubits are used for each clause, plus a large amount of edges for each variable are added. The result of the encoding then is usually large and hard to embed in the hardware.

A paper by Chancellor et al. [29] provides an encoding for the Max-k-SAT and low-density parity check problems. Two encodings are provided for two specific classes of constraints, disjunctions and parity checks. Using these two constraints the paper proposes an encoding for the **Low Density parity problem**, used in efficient turbo codes. While heavily tuned and effective for the problem at hand, the two constraints are not extremely suited for generic SAT and maxSAT problems in general. The two encoding can be seen in Figure 4.2.

Pakin in [76] provides a macro language to work with constraint satisfaction problems. It implements a format to express Ising models and libraries of penalty functions, and a software tool to handle them. The software relies on D-Wave software libraries to perform the final embedding step.

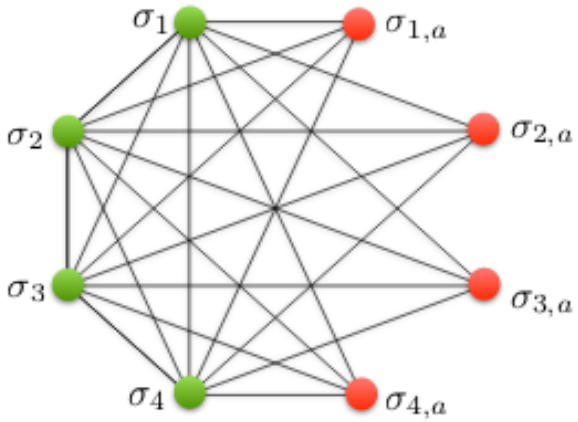


Figure 4.2: Induced graph for the encodings for the Max-4-SAT clause $((x_1 \vee x_2 \vee x_3 \vee x_4))$ and parity check $((x_1 \oplus x_2 \oplus x_3 \oplus x_4))$ from [29].

4.2 Placement and Routing

There are have been several approaches to map large Boolean functions or more generally large discrete optimization problems to fit D-Wave hardware.

Most of these efforts have used global embedding (described in the next chapters) [25], or otherwise, as in Trummer et al. [93], Chancellor et al. [29], Zaribafiyan et al. [97], and Andriyash et al. [6], used a ad-hoc placement approach optimized for the specific constraints at hand.

Su et al. [90] instead used a different place-and-route approach, based on simulated annealing of gate positions. The paper goal is to provide an encoding for Boolean satisfiability problems. It uses a simple QUBO encoding, with a short list of encoded two-input gates, and uses simulated annealing for placement and routing. Table 4.1 shows final results for embedding various functions on a 100×100 Chimera hardware graph, with a decisive under-usage of resources.

Name	WireLength	CellUsage	TotalUsage	RunTime (s)
C6288	68366	11.88%	66.14%	704.31
C5315	42438	8.89%	41.82%	376.51
pair	37268	7.89%	36.80%	344.62
dalu	41730	6.82%	40.11%	309.39
frg2	28422	5.66%	27.75%	202.16
C3540	31944	5.16%	30.67%	222.75
i7	13032	4.33%	13.73%	119.66
sgen6-960-5-1	37200	4.44%	34.13%	2391.32
edges-072-3-7923777-13	21366	3.75%	19.49%	1077.08
x3	18098	4.29%	18.06%	106.61
edges-070-3-1250111-33	20036	3.64%	18.36%	1613.07
apex6	15678	3.93%	15.72%	98.5
sgen6-840-5-1	28074	3.87%	26.11%	1672.47
i9	17166	3.68%	16.92%	92.56
i6	10744	3.42%	11.26%	64.43
rot	11894	3.31%	11.99%	77.79
too_large	22766	3.63%	21.84%	99.86
alu4	26076	3.59%	24.65%	186.44
edges-025-4-10062999-1-00	12106	2.48%	11.37%	835.36
i2	4778	2.09%	5.05%	36.23

Table 4.1: Table of encoding results from [90]. The columns contain, respectively: Name of the encoded problem, total number of qubits used for wires/chains, percentage of hardware cells/qubits used, and the run-time of the algoritm.

4.3 Performance Benchmarks

Regarding D-Wave hardware performance, there have been several publications benchmarking the performance compared to software solvers.

McGeoch et al. [66] and Santra et al. [83] looked at (weighted) Max2SAT problems, comparing the state-of-the art with quantum annealers. It is straightforward to convert Ising problem to Max2SAT problems; Each term of a QUBO problem can be interpreted as a clause where the θ parameter is the clause weight. Figure 4.3 plots the relative performance between the quantum annealer and various software algorithms.Another paper, by King et al. [58], found similar results for a class of constraint satisfaction problems.

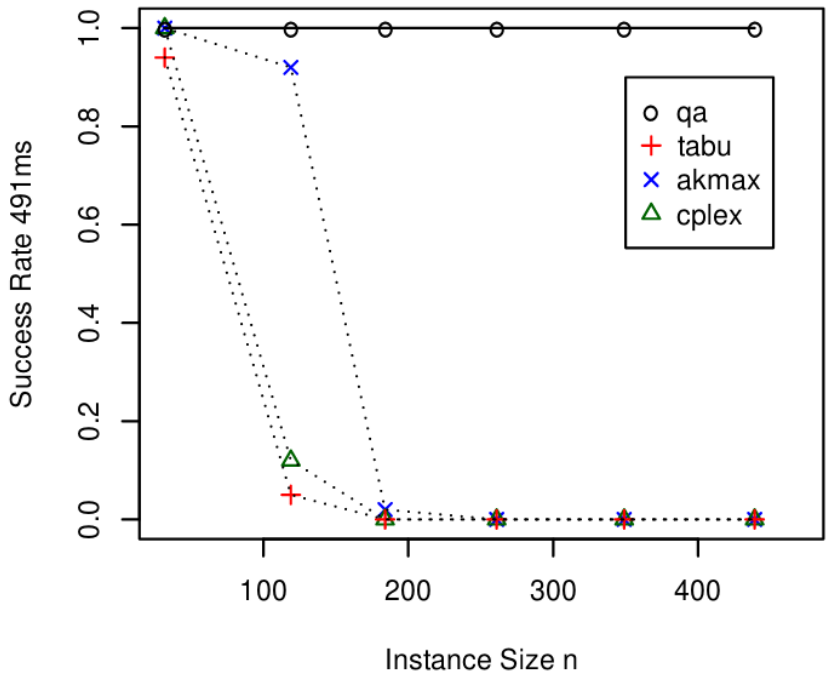


Figure 4.3: Plotted success rates with a $491ms$ second threshold for various solvers on various Max2SAT problems, from [66]. . The graph compares the performance of various solvers (tabu,akmax,cplex) for the timescale of a quantum annealing process on a D-Wave machine (qa) and various software solvers. Larger problems require increasing amount of computation, while the quantum annealer finds optimal solutions for all problems in a single run (with 1000 samples returned per run).

Douglass et al. [41] and Pudenz et al. [78] looked at ALLSAT problems. The goal in these papers is to sample multiple diverse solutions of a Boolean formula, in particular for the construction of SAT filters. SAT filters are, similarly to Bloom filters, used to perform probabilistic membership queries on large sets. Each element of a large set is mapped to a set of clauses for a SAT problem. One or more solutions of this SAT problem will function as filter. To query the filter for an element, we check if the previous solutions satisfy the mapped clauses. If not, the filter query returns a negative result. Thus, building a SAT filter requires finding a large number of solutions of a particular SAT problem. Figure 4.4 shows the relative performance of

D-Wave annealer vs. various SAT solvers in SAT filter construction.

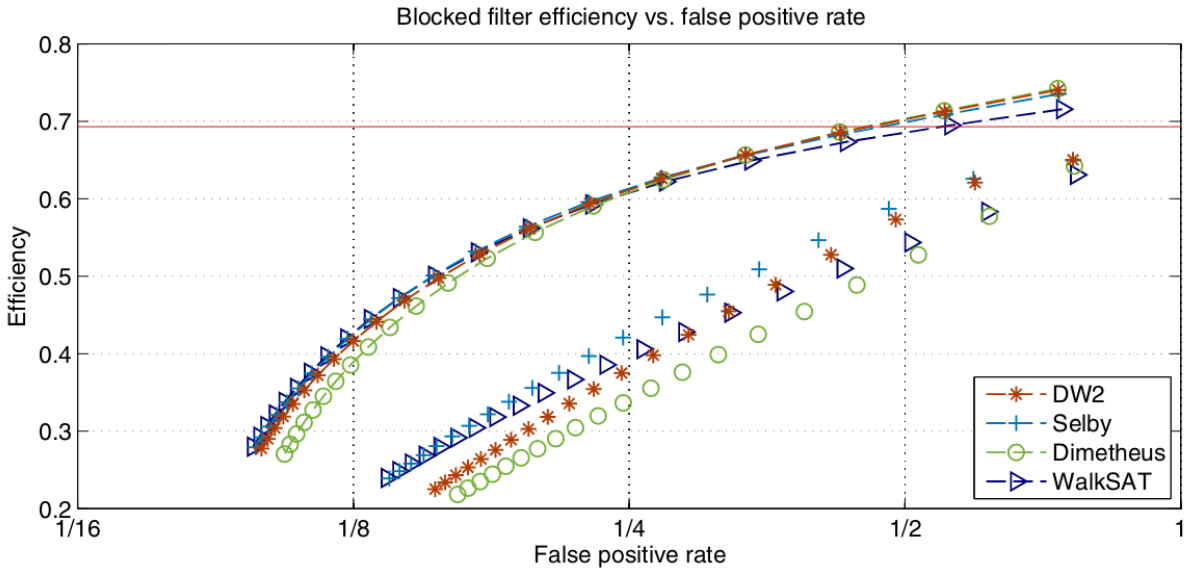


Figure 4.4: Comparison of SAT filter performance of quantum annealing vs. various ALLSAT solvers from [41]. For comparison, the theoretical efficiency value for a Bloom Filter (0.69) is indicated (red line). For each solver, the graph shows performance for both off-line (upper, dashed lines) and on-line (lower, no lines) filters.

For a more theoretical approach Farhi et al. [45] and Hen and Young [52] studied the performance of adiabatic quantum computing on several SAT and other intractable problems, using simulations to determine experimentally what are the run-times for reaching a solution. Both paper found that when the adiabatic quantum computer interpolates linearly between initial and final state the required time increased exponentially, even if with a lower coefficient than classical annealers.

Part II

Contributions

Chapter 5

Theoretical foundations

In this chapter we will lay out the theoretical foundations of the encoding problem. We will follow the structure of the paper "Solving SAT and MaxSAT with a Quantum Annealer: Foundations, Encodings, and Preliminary Results" [18].

5.1 Problem statement

First, let's focus on the problem. Let $F(\underline{\mathbf{x}})$ be a Boolean function on a set of n **Boolean variables** $\underline{\mathbf{x}} \stackrel{\text{def}}{=} \{x_1, ..., x_n\}$. Ising models are defined on binary variables, so we represent Boolean value $\perp$ with -1 and $\top$ with $+1$, we can then assume that $x_i \in \{-1, 1\}$. Suppose that we have a quantum annealer with n qubits defined on a hardware graph $G = (V, E)$ (usually a sub-graph of a Chimera or a Pegasus graph of Figures 3.5 and 3.6 if not otherwise specified). As stated before we assume that the state of each qubit z_i corresponds to the value of variable x_i, $i = 1, \ldots, n = |V|$. One way to use the quantum annealer to determine whether $F(\underline{\mathbf{x}})$ is satisfiable is to find an energy function as in Equation 3.18 whose ground states $\underline{\mathbf{z}}$ correspond to the satisfying assignments $\underline{\mathbf{x}}$ of $F(\underline{\mathbf{x}})$.

Example 2. *Suppose that F is defined as follows:*

$$F(\underline{\mathbf{x}}) \stackrel{def}{=} x_1 \oplus x_2$$

Since $F(\underline{\mathbf{x}}) = \top$ if and only if $x_1 + x_2 = 0$, the Ising model in a graph will contain 2 qubits z_1, z_2 joined by an edge $(1, 2) \in E$ such that $\theta_{12} = 1$ and will have two ground states $(+1, -1)$ and $(-1, +1)$, which correspond to the satisfying assignments of F, and two excited states $(+1, +1)$ and $(-1, -1)$, corresponding to the non-satisfying ones.

In reality the number of functions $F(\underline{\mathbf{x}})$ that can be solved with this approach is very limited. This is because the energy $H(\underline{\mathbf{z}})$ in (3.18) is restricted to second-degree polynomials and because the graph G is typically sparse. To deal with this problem we can use a larger quantum annealer with a number h of additional qubits representing **ancillary Boolean variables** (or **ancillas** for short) $\underline{\mathbf{a}} \stackrel{\text{def}}{=} \{a_1, ..., a_h\}$, so that $|V| = n + h$. A **variable placement** is a mapping of the $n+h$ input and ancillary variables into vertices of the hardware graph G. Since G is not a complete graph, the energy function will have different properties with different variable placements. We call **Ising encoding** the vector of values provided to the annealer for the θ parameters in (3.18) together with a variable placement. The **gap** $g_{min} \geq 0$ of an Ising encoding is the minimum energy difference $(\min \Delta H(\underline{\mathbf{z}}))$ between a ground state (i.e., satisfying assignments) and the other excited states (i.e., non-satisfying assignments). As we have seen, the stability of an adiabatic quantum system transition depends on the minimum gap and in practice larger gaps lead to higher success rates during the annealing process [13]. Thus, we define the **encoding problem** for $F(\underline{\mathbf{x}})$ as the problem of finding an Ising encoding with the maximum gap.

Recall that the encoding problem is typically over-constrained. The Ising model of Equation 3.18 has to discriminate between m satisfying assignments and k non-satisfying assignments, with $m + k = 2^n$, whereas

the number of degrees of freedom is given by the number of the θ_i and θ_{ij} parameters, which grows linearly in Chimera and Pegasus architectures. Thus the number of ancillas that are needed in order to have a solution (h) can grow exponentially with the number of $\underline{\mathbf{x}}$ variables of the Boolean formula (n).

In the rest of this chapter, we will assume that a Boolean formula $F(\underline{\mathbf{x}})$ is provided and that a sufficient number h of qubits is used for ancillary variables $\underline{\mathbf{a}}$.

5.2 Penalty Functions

In the initial phases we will assume that a variable placement chosen by the user is given along with the Boolean formula, placing $\underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ into the sub-graph G. Thus, for now we can identify from the beginning each binary variable z_j with the jth vertex in V and with either an original Boolean variable $x_k \in \underline{\mathbf{x}}$ or as an ancilla variable $a_\ell \in \underline{\mathbf{a}}$, and we will write that $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$.

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

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \stackrel{def}{=} \theta_0 + \sum_{i \in V} \theta_i z_i + \sum_{(i,j) \in E} \theta_{ij} z_i z_j \tag{5.1}$$

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

$$\forall \underline{\mathbf{x}} \;\; min_{\{\underline{\mathbf{a}}\}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \begin{cases} = 0 & \text{if } F(\underline{\mathbf{x}}) = \top \\ \geq g_{min} & \text{if } F(\underline{\mathbf{x}}) = \bot \end{cases} \tag{5.2}$$

where $z_i, z_j \in \underline{\mathbf{z}}$*, while* $\theta_0 \in (-\infty, +\infty)$ *("*offset*"),* $\theta_i \in [-2, 2]$ *("*biases*") and* $\theta_{ij} \in [-1, 1]$ *("*couplers*") and* g_{min} *are rational-valued parameters. Operation ranges of biases and couplers are bounded by current hardware limitations and can be subject to change. The largest* g_{min} *such that* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ *satisfies* (5.2) *is called the* **gap** *of* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$*.*

The offset value θ_0, absent in the original Ising model definition, is added to set the value of $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to zero when $F(\underline{\mathbf{x}}) = \top$, so in practice $-\theta_0$ corresponds to the energy of the ground states of (3.18). To simplify the notation we assume that $\theta_{ij} = 0$ when $(i, j) \notin E$, and use $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ when $\underline{\mathbf{a}} = \emptyset$.

For clarification, we follow with several examples of penalty functions.

Example 3. *Consider the equivalence between two variables, $F(\underline{\mathbf{x}}) \stackrel{def}{=} (x_1 \leftrightarrow x_2)$, can be encoded without ancillas with a single coupling between two connected vertices, with zero biases:*

$$P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) \stackrel{def}{=} 1 - x_1 x_2$$

In fact, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 0$ if x_1, x_2 have the same value; $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 2$ otherwise. This penalty function has $g_{min} = 2$.

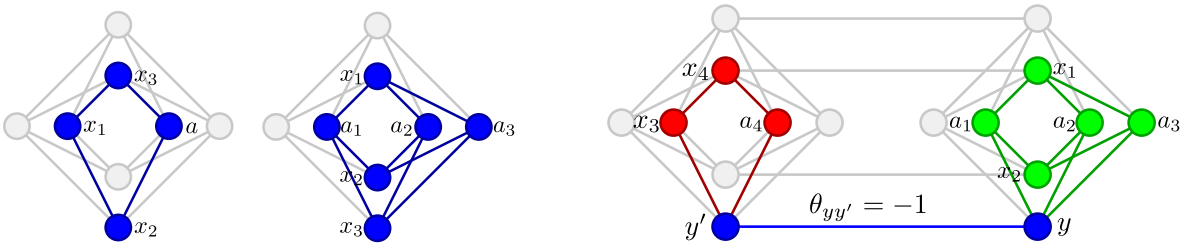
Penalty $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$ in Example 3 is also called a **(equivalence) chain** connecting x_1, x_2, as it forces them to have the same value. The following two examples show that ancillary variables are necessary for encoding some Boolean function $F(\underline{\mathbf{x}})$, even when $F(\underline{\mathbf{x}})$ is a small formula or when G is a complete graph.

Example 4. *Consider the AND function, $F(\underline{\mathbf{x}}) \stackrel{def}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$. If G is a complete 3-clique, then $F(\underline{\mathbf{x}})$ can be encoded without ancillas with gap $g_{min} = 2$ by setting:*

$$P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = \frac{3}{2} - \frac{1}{2}x_1 - \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_1 x_2 - x_1 x_3 - x_2 x_3$$

With this, it is easy to see that $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 0$ if x_1, x_2, x_3 verify $F(\underline{\mathbf{x}})$, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 6$ if $x_1 = x_2 = -1$ and $x_3 = 1$, $P_F(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = 2$ otherwise.

Since the Chimera graph is bipartite and has no 3-cliques, for such a graph the above AND function needs (at least) one ancilla a, becoming:



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

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

(c) $x_4 \leftrightarrow (x_3 \wedge (x_1 \oplus x_2))$ obtained by combining 5.1(b) and 5.1(a).

Figure 5.1: Example of mappings within the Chimera cell. Penalty functions use only colored edges. 5.1(c) combines 5.1(a) and 5.1(b) using chained proxy variables y, y'. The penalty function of the composition is obtained by rewriting $x_4 \leftrightarrow (x_3 \wedge (x_1 \oplus x_2))$ into its equisatisfiable formula $(x_4 \leftrightarrow (x_3 \wedge y')) \wedge (y \leftrightarrow (x_1 \oplus x_2)) \wedge (y' \leftrightarrow y)$.

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = \frac{5}{2} - \frac{1}{2}x_1 - \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_1x_2 - x_1x_3 - x_2a - x_3a$$

This version still has gap $g_{min} = 2$ and can be embedded on Chimera, as in Figure 5.1(a).

Example 5. *Consider the XOR function $F(\underline{\mathbf{x}}) \stackrel{def}{=} x_3 \leftrightarrow (x_1 \oplus x_2)$. Even considering a complete graph, $F(\underline{\mathbf{x}})$ has no ancilla-free encoding. Within the Chimera graph though, $F(\underline{\mathbf{x}})$ can be encoded with three ancillas a_1, a_2, a_3 as:*

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 5+x_3+a_2-a_3+x_1a_1-x_1a_2-x_1a_3-x_2a_1-x_2a_2-x_2a_3+x_3a_2-x_3a_3$$

This has gap $g_{min} = 2$ and is embedded, as in Figure 5.1(b).

We can make a few observations on Definition 1. First, a penalty function is an Ising model that separates satisfying assignments from non-satisfying ones by a energy gap of at least g_{min}. Thus, the following fact is a straightforward consequence of Definition 1.

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

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

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

The consequence of Proposition 1 is that we can use the QA hardware as a satisfiability checker for $F(\underline{\mathbf{x}})$ by minimizing the Ising model defined by penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. A ground state such that $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0$ implies that it is an assignment satisfies $F(\underline{\mathbf{x}})$. Conversely, if the QA hardware were to guarantee minimality, then a returned value of $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}$ would imply that $F(\underline{\mathbf{x}})$ is unsatisfiable. However, since quantum annealer do not guarantee minimality (as seen in Chapter 3), if $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}$ then there is still a chance that that the quantum annealer reached an excited state and thus $F(\underline{\mathbf{x}})$ is satisfiable. Nevertheless, the larger g_{min} is, the less likely this false negative case occurs [13].

The magnitude of the gap is directly related to the bounds on the parameters. A penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is **normal** if $|\theta_i| = 2$ for at least one θ_i or $|\theta_{ij}| = 1$ for at least one θ_{ij}. In order to maximize g_{min}, penalty functions fed to the QA hardware should be normal so that to exploit the full range of the $\underline{\boldsymbol{\theta}}$ parameters. Any penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ can be normalized by multiplying all its coefficients by a **normalization factor**:

$$c \stackrel{\text{def}}{=} \min \left\{ \min_i \left(\frac{2}{|\theta_i|} \right), \min_{\langle ij \rangle} \left(\frac{1}{|\theta_{ij}|} \right) \right\}. \tag{5.3}$$

Note that if $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is a valid penalty function but non-normal, then $c > 1$, so that the gap can be increased with normalization. $c \cdot g_{min} > g_{min}$. Normalization also works in scale down a $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ that is not valid because its θ's do not fit into the allowable ranges (in which case $c < 1$). Hereafter we assume without loss of generality that all penalty functions are normal.

5.3 Properties of Penalty Functions and Problem Decomposition

As will be shown in the next chapter, finding the values for $\underline{\theta}$ given a variable assignment requires solving an SMT formula composed by a set of linear equations for every model of $F(\underline{\mathbf{x}})$ plus a set of linear inequalities for each counter-model of $F(\underline{\mathbf{x}})$. Thus, the number of linear constraints grows exponentially in n. Since the θ's grow proportionally to $(n+h)$ the number of ancillas required to satisfy (5.2) grows very rapidly with the size of the formula. This makes a direct search based on (5.1)-(5.2) intractable beyond a certain size. We address this issue by building penalty functions by composition, at the expense of a larger final Ising model size.

We assert some properties of penalty functions:

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

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

Property 2. *Let $F^*(\underline{\mathbf{x}}) \stackrel{def}{=} F(x_1, ..., x_{r-1}, \neg x_r, x_{r+1}, ..., x_n)$ for some index r. Assume $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is a penalty function for $F(\underline{\mathbf{x}})$ with gap g_{min} and with variable placement of $\underline{\mathbf{x}}$ into V. Then $P_{F^*}(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}^*)$, where $\underline{\boldsymbol{\theta}}^*$ is defined as follows for every $z_i, z_j \in \underline{\mathbf{x}}, \underline{\mathbf{a}}$:*

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

Notice that since the previously defined bounds over $\underline{\boldsymbol{\theta}}$ (namely $\theta_i \in [-2, 2]$ and $\theta_{ij} \in [-1, 1]$) are symmetric with respect to 0, if $\underline{\boldsymbol{\theta}}$ is in range then $\underline{\boldsymbol{\theta}}^*$ is as well.

Property 2 states that negating a x_i variable of a Boolean function is equivalent to flipping the value a x_i variable in the QUBO problem. Furthermore, if σ is a permutation over variables and $\sigma F = F(\sigma \underline{\mathbf{x}})$, the penalty function of σF (assuming a complete hardware graph or at least a permutation that is an automorphism for it) is:

$$P_{\sigma F}(\underline{\mathbf{x}}|\underline{\boldsymbol{\theta}}) = P_F(\sigma \underline{\mathbf{x}}|\underline{\boldsymbol{\theta}})$$

Two Boolean functions that become logically equivalent by these permutations or negations of their variables are called **NPN-equivalent** [38]. Given the penalty function for a Boolean formula assuming a complete graph, any other NPN equivalent formula can be encoded trivially by applying Property 2. Notice that checking NPN equivalence is an intractable problem in theory, but in practice it takes a negligible time for small n (i.e., $n \leq 16$) [53]. The process of negating a single variable when referring to an Ising model as in Property 2 is also known as a **spin-reversal transform**. Let's consider an example.

Example 6. *Consider the OR function* $F(\underline{\mathbf{x}}) \stackrel{def}{=} x_3 \leftrightarrow (x_1 \vee x_2)$. *We notice that, as it can be rewritten as* $F(\underline{\mathbf{x}}) = \neg x_3 \leftrightarrow (\neg x_1 \wedge \neg x_2)$, *it is NPN-equivalent to the function of Example 4. Thus, by Property 2 a penalty function for* $F(\underline{\mathbf{x}})$ *can be defined by taking the* $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ *in Example 4 and toggling the signs of the coefficients of the* x_i*'s:*

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = \frac{5}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - x_3 + \frac{1}{2}x_1x_2 - x_1x_3 + x_2a + x_3a$$

It can be placed as in Figure 5.1(a) and has the same gap $g_{min} = 2$.

Property 3. *Let* $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^k})$ *be a Boolean formula on Boolean variables* $\underline{\mathbf{x}} = \cup_k \underline{\mathbf{x}^k}$, *where the* $\underline{\mathbf{x}^k}$*s may be non-disjoint. Suppose that each sub-formula* F_k *has a penalty function* $P_{F_k}(\underline{\mathbf{x}^k}, \underline{\mathbf{a}^k}|\underline{\boldsymbol{\theta}^k})$ *with minimum gap* g^k_{min}, *where the* $\underline{\mathbf{a}^k}$*s are all disjoint. Given a list* w_k *of positive rational*

values such that, for every $z_i, z_j \in \underline{\mathbf{x}} \cup \bigcup_{k=1}^{K} \underline{\mathbf{a}^k}$*:*

$$\theta_i \stackrel{def}{=} \sum_{k=1}^{K} w_k \theta_i^k \in [-2, 2], \quad \theta_{ij} \stackrel{def}{=} \sum_{k=1}^{K} w_k \theta_{ij}^k \in [-1, 1], \tag{5.4}$$

then we can build a penalty function for $F(\underline{\mathbf{x}})$ *in the following way:*

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}^1 ... \underline{\mathbf{a}}^K | \underline{\boldsymbol{\theta}}) = \sum_{k=1}^{K} w_k P_{F_k}(\underline{\mathbf{x}^k}, \underline{\mathbf{a}^k} | \underline{\boldsymbol{\theta}^k}). \tag{5.5}$$

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

Property 3 states that a penalty function for a conjunction of subformulas can be obtained as a sum of the penalty functions of the subformulas. The choice of the values of weights w_k is not unique in general. Also, note that g_{min} may be greater than $\min_{k=1}^{K} w_k g_{min}^k$, as it might happen that $g_{min} = w_k g_{min}^k$ for some unique k and no truth assignment violating F_k with cost $w_k g_{min}^k$ satisfies all other F_i's.

The composition is performed using weights w_k because penalty functions of formulas can share variables that sum up biases or couplings, possibly resulting into out-of-range values (5.4), effectively requiring renormalization. If the w_k's are smaller than 1, then the gap g_{min} of the final penalty function may become smaller. Furthermore, Property 3 requires placing variables into qubits that are shared among conjunct subformulas, ignoring constraints given by the hardware graph. This may limit the chances of finding valid placements for the variables in the graph.

An alternative way of composing subformula while avoiding this problem is to map shared variables into multiple distinct qubits that are forced to be equal by chains of equivalences. Consider $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^k})$ as in Property 3. We can replace all the occurrences of x_i in any F_k with a fresh variable ${x_i}^{k^*}$. This gives us the formula $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^{k^*}})$, with $\underline{\mathbf{x}^{k^*}}$ all disjoint. Let us define $F^*(\underline{\mathbf{x}^*})$ in the following way:

$$F^*(\underline{\mathbf{x}}^*) \stackrel{\text{def}}{=} \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^{k^*}}) \wedge \bigwedge_{\langle {x_i}^{k^*}, {x_i}^{k'^*}\rangle \in Eq(x_i)} ({x_i}^{k^*} \leftrightarrow {x_i}^{k'^*}) \tag{5.6}$$

where $\underline{\mathbf{x}}^* = \cup_k \underline{\mathbf{x}^{k^*}}$, and $Eq(x_i)$ is a set of pairs $\langle {x_i}^{k^*}, {x_i}^{k'^*}\rangle$ of variables replacing x_i such that (5.6) states that for every k all the x_i^k are equivalent. By construction, $F(\underline{\mathbf{x}})$ is satisfiable if and only if $F^*(\underline{\mathbf{x}}^*)$ is satisfiable, and from every model $\underline{\mathbf{x}}^*$ for $F^*(\underline{\mathbf{x}}^*)$ we have a model $\underline{\mathbf{x}}$ for $F(\underline{\mathbf{x}})$ by assigning to each x_i the value of all of the corresponding ${x_i}^{k^*}$s.

Now assume we have a penalty function $P_{F_k}(\underline{\mathbf{x}}^{k^*}, \underline{\mathbf{a}^k}|\underline{\boldsymbol{\theta}}^k)$ for each F_k with disjoint $\underline{\mathbf{a}^k}$. We have seen in Example 3 that $(1 - {x_i}^{k^*}{x_i}^{k'^*})$ are penalty functions of gap 2 for the $({x_i}^{k^*} \leftrightarrow {x_i}^{k'^*})$ chain clauses in (5.6). Thus we can apply Property 3 and write a penalty function for $F^*(\underline{\mathbf{x}}^*)$ in the following way:

$$P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = \sum_{k=1}^{K} P_{F_k}(\underline{\mathbf{x}}^{k^*}, \underline{\mathbf{a}^k}|\underline{\boldsymbol{\theta}}^k) + \sum_{\langle {x_i}^{k^*}, {x_i}^{k'^*}\rangle \in Eq(x_i)} (1 - {x_i}^{k^*}{x_i}^{k'^*}). \tag{5.7}$$

Note that all the θ's remain in the valid range because the $\underline{\mathbf{x}^{k^*}}$s and $\underline{\mathbf{a}^k}$s are all disjoint and the biases of the $(1 - {x_i}^{k^*}{x_i}^{k'^*})$ terms are zero, so distinct sub-penalty functions P_{F_k} in (5.7) involve disjoint groups of biases and couplings. Thus we can state the following:

Property 4. *$P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ in (5.7) is a penalty function for $F^*(\underline{\mathbf{x}}^*)$ in (5.6). The gap of $P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is $g_{min} \geq \min(\min_{k=1}^{K} g_{min}^k, 2)$.*

With this method we can represent a single variable x_i with a series of qubits connected by a chain of strong couplings $(1 - x_i x_i')$ (For $x_i \leftrightarrow \neg x_i'$, we simply use $(1 + x_i x_i')$). Notice that it is not necessary to explicitly force every pair of copies $\langle x_i^k, x_i^{k'}\rangle$ to be equivalent; rather it suffices that the equivalences form a connected graph. Moreover, we can introduce additional

copies of x_i as necessary to facilitate variable placement on the hardware graph G. A set of qubits representing the same variable in this way is called a **chain** and is the subject of Section 5.5. Thus, $P_{F^*}(\underline{\mathbf{x}}^*, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ can be implemented in (5.7) by placing the distinct penalty functions $P_{F_k}(\underline{\mathbf{x}}^{k*}, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ into separate sub-graphs of G and then connecting them with chains.

In Section 3.3.1 we have seen that we can always decompose a formula $F(\underline{\mathbf{x}})$ into a conjunction of smaller formulas $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ with Tseitin's transformation (3.20). Combined with Properties 3 and 4, this means that we can decompose $F(\underline{\mathbf{x}})$ into multiple and smaller conjuncts, encode these separately and then reconstruct the final penalty function. With this approach we only need to encode a set of Boolean functions $(y_i \leftrightarrow F_i(\underline{\mathbf{x}}^i, \underline{\mathbf{y}}^i))$, each small enough to allow a search for an efficient penalty function. Their reduced size allows us to search for penalty function with good gaps, and then their combination keeps the gap of the penalty function for the original function essentially as large as possible. Let us make an example.

Example 7. *Consider the function:*

$$F(\underline{\mathbf{x}}) \stackrel{def}{=} x_4 \leftrightarrow (x_3 \wedge (x_1 \oplus x_2))$$

.

Applying (3.20) *and* (5.6) *this can be rewritten as*

$$\mathbf{F}^*(\underline{\mathbf{x}}, y, y') = (x_4 \leftrightarrow (x_3 \wedge y')) \wedge (y \leftrightarrow (x_1 \oplus x_2)) \wedge (y' \leftrightarrow y)$$

The penalty functions of the three conjuncts can be produced as seen in

Examples 4, 5 and 3, and can be conjoined as in Property 4:

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

As previously stated, there is no interaction between the biases and couplings of the three components, only the offset is summed up. The resulting gap is $min\{2, 2, 2\} = 2$*. On a Chimera hardware graph, they can be placed as in Figure 5.1(c).*

Taking these properties in consideration we can build a "divide-and-conquer" approach for the SATtoIsing problem:

(i) Use Tseitin's decomposition on the input formula, and rewrite every conjunct $F(\underline{\mathbf{x}})$ which is not small enough into an equivalently-satisfiable one $F^*(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ as in (3.20) until penalty functions for all its conjuncts can be easily computed;

(ii) rename shared variables and compute the global penalty functions as in Property 4;

(iii) place the sub-penalty functions into subgraphs of the hardware graph and connect using chains equivalent qubits, representing shared variables between conjuncts.

5.4 Exact Penalty Functions and MaxSAT

If we want to encode a MaxSAT problem into a QUBO problem we require a stronger version of the penalty function in Definition 1: we need an

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

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

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

The following is a simple example of an exact penalty function.

Example 8. *The penalty function of $F(\underline{\mathbf{x}}) \stackrel{def}{=} (x_1 \leftrightarrow x_2)$ in Example 3 is exact, whereas those of $F(\underline{\mathbf{x}}) \stackrel{def}{=} x_3 \leftrightarrow (x_1 \wedge x_2)$ and $F(\underline{\mathbf{x}}) \stackrel{def}{=} x_3 \leftrightarrow (x_1 \oplus x_2)$ in Examples 4 and 5 are not exact.*

As a consequence of Property 3 and Definition 2, Exact penalty functions allow us to encode weighted MaxSAT problems, with some restrictions.

Proposition 2. *Let*

$$F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^k})$$

be a Boolean formula s.t. $\underline{\mathbf{x}} = \cup_k \underline{\mathbf{x}^k}$, and

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \stackrel{def}{=} \sum_{k=1}^{K} P_{F_k}(\underline{\mathbf{x}^k}, \underline{\mathbf{a}^k}|\underline{\boldsymbol{\theta}^k}),$$

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

We can use Proposition 2 to encode a generic weighted MaxSAT problem $\{\langle F_k, c_k\rangle\}_k$ by setting $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \sum_{k=1}^{K} w_k P_{F_k}(\underline{\mathbf{x}^k}, \underline{\mathbf{a}^k}|\underline{\boldsymbol{\theta}^k})$ where $w_k \stackrel{\text{def}}{=} \frac{c_k}{g_k} \cdot c$

and c is a normalization factor (5.3). Penalty functions in Proposition 2 $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k|\underline{\boldsymbol{\theta}}^k)$ were not exact then a solution $\underline{\mathbf{x}}$,$\underline{\mathbf{a}}$ that minimizes $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ might not be optimal for MaxSAT as it could prefer to violate some additional F_k to reach a lower final energy state. Since exact penalty functions are more difficult to obtain than regular penalty functions, in principle one could use non-exact penalty functions to produce sub-optimal but useful solutions.

In the previous section on SATtoIsing we outlined a "divide-and-conquer" approach based on the idea of mapping single variables into multiple distinct qubits which are then connected by chains of equivalences. Applying the same approach to MaxSAT is not as straightforward, because the penalty function recomposition of Property 4 cannot always be combined with exact penalty functions in a useful way. Suppose we want to use Equation 5.7 to solve a MaxSAT problem $\{\langle F_k, g_k\rangle\}_k$ using Proposition 2. As the following example shows, there may be minimum-energy solutions of Equation 5.7 which violate some equivalence $({x_i}^{k*} \leftrightarrow {x_i}^{k'*})$ in (5.6). This can happen if the solution avoids violating one or more of the F_k's whose sum of gaps is greater than 2. Such QUBO solution would not be *not* a solution of the MaxSAT problem, because it would contain different truth values to distinct instances of the same variable in the original problem.

Example 9. *Consider the trivial MaxSAT problem* $\{\langle F_i(x), c\rangle\}_{i=1}^4$ *for some penalty value* $c > 0$ *where* $F_1(x) = F_2(x) \stackrel{def}{=} x$, *and* $F_3(x) = F_4(x) \stackrel{def}{=} \neg x$. *The two possible solutions* $x = \top$ *and* $x = \bot$ *are both optimum with penalty* $2c$ *and falsify* F_3, F_4 *and* F_1, F_2 *respectively. We have the following normal and exact penalty functions, each of gap* $g_i = 4$*:*

$$P_{F_1}(x) = P_{F_2}(x) = 2 - 2x$$

$$P_{F_3}(x) = P_{F_4}(x) = 2 + 2x$$

Suppose we want to encode the problem in such a way to fit into a linear

chain of 4 qubits adopting the encoding in Property 4. We introduce four copies of x, namely x^1, x^2, x^3, x^4, and obtain:

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

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

In general, the problem arises when it is energetically cheaper to violate some chain equivalence $(x_i^{k*} \leftrightarrow x_i^{k'*})$ in Equation 5.6 rather than to violate all the penalty functions $\{F_k(\underline{\mathbf{x}^k}) : x_i \in \underline{\mathbf{x}^k}\}$ on one side of the equivalence. One solution to this problem is to scale down the P_{F_k}'s with sufficiently small weights $w_k < 1$, at the cost reducing the gaps g_k.

Let $\mathcal{I} = \{k : x_i \in \underline{\mathbf{x}^k}\}$, and suppose that all chains form a tree on the hardware graph. An equivalence $(x_i^{k*} \leftrightarrow x_i^{k'*})$ splits the chain into two subchains, and splits $\mathcal{I}$ into two subsets $\mathcal{I}_k$ and $\mathcal{I}_{k'}$. Assume we have a desired gap $g_{desired} > 0$ between Ising solutions with broken chains from solutions that can be applied to the original MaxSAT problem. Then a sufficiently large gap for the equivalence $(x_i^{k*} \leftrightarrow x_i^{k'*})$ is:

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

This gap ensures that it is $g_{desired}$ cheaper to violate all the constraints in $\mathcal{I}_k$ or $\mathcal{I}_{k'}$ rather than to violate $(x_i^{k*} \leftrightarrow x_i^{k'*})$. To ensure that all equivalence constraints are not violated, a sufficient gap for the entire chain is

$$g_{chain} = \max_{(x_i^{k*}, x_i^{k'*}) \in Eq(x_i)} g_{(k,k')}. \tag{5.8}$$

Recall from Equation 5.6 that $Eq(x_i)$ is the set of variable pairs $(x_i^{k*}, x_i^{k'*})$ that form equivalences $(x_i^{k*} \leftrightarrow x_i^{k'*})$ in the chain of x_i. Furthermore, each equivalence has gap 2, thus we update the weight definition in Proposition 2 for each $k \in \mathcal{I}$:[1]

$$w_k = \frac{2 \cdot c_k}{g_k \cdot g_{chain}} \tag{5.9}$$

As a chain may connect a large number of constraints, the necessary chain gap may be much larger than the gaps of the original penalty functions, resulting in a small final g_{min} after normalization. A paper by Choi [33] provides an alternative bound on g_{chain}. In the paper, the author focuses on converting QUBO problems with different graph topologies (this problem will be explained in details in Section 5.5). A bound for the chain strength is provided in order to ensure that all minima of an embedded QUBO problem can be mapped to a minimum of the original QUBO problem. Let $\theta_i^* = \sum_k w_k \theta_i$ be the bias value obtained by sharing the x_i variable as in Property 3. Assume that it is necessary to replace x_i with a chain with l_i leaves (i.e. vertices of the tree graph of degree 1). In this case, QUBO minima are preserved if the chain gap is the following:

$$g_{chain} \geq 2\frac{l_i - 1}{l_i} \left(\sum_{(i,j) \in E} |\theta_{ij}^*| - |\theta_i^*| \right) + g_{desired} \tag{5.10}$$

This alternative bound is sometimes lower than (5.8), especially when $|\theta_i^*|$ is high. Note that, as the original paper explains, if the bound value is negative then P_{F^*} is monotonic on x_i. If that is the case, then $x_i = -sgn(\theta_i^*)$ always minimizes P_{F^*}, so the minimum value of x_i is trivial and the variable can be simplified away.

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

In general, neither (5.8) nor (5.10) are typically very tight bounds on required chain gap, and the smallest viable chain gap depends heavily on the characteristic of the original MaxSAT problem instance. In practice g_{chain} is often determined empirically; this is discussed further in the experimental chapter.

Furthermore it is difficult to directly encode hard constraints for partial weighted MaxSAT. We can simulate hard constraint using very high costs for hard clauses at a very high cost in gap size. Overall, the MaxSATtoIsing problem requires the usage of *exact* penalty functions for its sub-formulas, which are more difficult to obtain, and the high required gaps on chains typically results in smaller gaps after normalization.

5.5 Embedding into a QA Architecture

The process of representing a single variable x_i by a collection of qubits connected in chains of strong couplings is known as **embedding**, in reference to the minor embedding problem of graph theory [33, 35]. Let $P_F(\underline{\mathbf{z}}|\underline{\boldsymbol{\theta}})$ be a penalty function whose interactions define an induced graph G_F (i.e. x_i and x_j are adjacent iff $\theta_{ij} \neq 0$) and let G_H be a QA hardware graph. A **minor embedding** of G_F in G_H is a function from vertices of the induced graph to set of vertices of the hardware graph $\Phi : V_{G_F} \to 2^{V_{G_H}}$. The image $\Phi(x_i)$ of a G_F-vertex is a chain, and the edges of G_H that connect these qubits are used to force them to be equivalent. Φ has the following properties:

- for each G_F-vertex x_i, the subgraph induced by $\Phi(x_i)$ (i.e the chain for x_i) is connected;

- for all distinct G_F-vertices x_i and x_j, $\Phi(x_i)$ and $\Phi(x_j)$ are disjoint;

- for each edge (x_i, x_j) in G_F, there is at least one edge between $\Phi(x_i)$ and $\Phi(x_j)$.

Embedding generic graphs is a computationally difficult problem [2], although certain structured problem graphs may be easily embedded in the Chimera graph [20, 97] and heuristic algorithms may also be used [25]. A reasonable goal in embedding is to minimize the sizes of the chains, as quantum annealing becomes less effective as more qubits are included in chains [61].

In our "divide-and-conquer" approach, we can tackle the problem of embedding in two different ways. The one that will be used in this thesis is **placement and routing**, in reference to the simpler problem of positioning and connection of components in a integrated circuit [12]. With this approach each sub-problem P_{F^*} is encoded in order to fit a sub-graph of the hardware graph. During the embedding phase, each P_{F^*} is assigned to a separate subgraph of the hardware, and then chains that respect the previously mentioned properties are sought.

First, the **placement** part decides a position for each component in such a way to minimize the average chain length. Usually heuristic methods are used, such as simulated annealing [91], continuous optimization [28], and recursive min-cut partitioning [82]. These algorithms need to be slightly adapted to be used on graph embedding, as they make different assumptions. For example, some algorithms allow an expansion of the available planar area when necessary, or assume that some space is always available for routing (in which case if Boolean functions are packed too tightly there will be no space for routing).

The routing steps consist in building the chains connecting variable instances, using as few qubits as possible. The problem can be formalized as follows. Assume a single variable x_i has been assigned to a set of vertices $T_i \subseteq V$, called its **terminals**, during the previous placement step. Let

C_i be the chain of qubits for variable x_i. If the following conditions are respected, we have a valid routing solution:

- all chains are disjunct: $C_i \cap C_j = \emptyset$;
- every chain contains all its terminals: $T_i \subseteq C_i$;
- C_i induces a connected sub-graph on the hardware graph.

Finding a single C_i with the minimum number of vertices is an instance of the **Steiner tree problem** [24], then C_i is a Steiner tree. Among routing solutions, we try to minimize the total number of vertices of G used or the size of the largest chain.

Routing to minimize the total number of vertices used is NP-hard, but polynomial-time approximation algorithms exist [50]. In practice, heuristic routing algorithms scale to problem sizes much larger than current QA architectures [96, 81, 31, 32, 30].

A different approach to finding models for $F(\underline{\mathbf{x}})$, **global embedding**, is based on first finding a penalty function on a complete graph G_F on $n+h$ variables, and after that embedding G_F into a hardware graph G_H using chains (e.g., using [20]). In this way chains can be used also for qubits contained in a single P_{F^*} and for ancillary variables, allowing greater flexibility. Thus, global embeddings usually need fewer qubits than placement and routing [13]; however, due to issues similar to the ones encountered in the previous MaxSAT section, the final gap of the penalty function obtained in this way is generally smaller and difficult to compute exactly. This thesis will focus on the former approach.

Chapter 6

SMT and OMT for Small Boolean Formulas

This chapter will focus on the process of using SMT and OMT solvers to find effective encodings of small Boolean formulas. As we will see, in this step we can choose the formula that will be encoded, and we prioritize effectiveness over efficiency. We want encodings with very few ancillary qubits, and we want to find a solution within a reasonable timeframe.

6.1 Penalty Functions Search via SMT/OMT($\mathcal{LRA}$).

We can take the definitions stated in the previous chapter on theoretical foundations to create a SMT formula. Having $\underline{\mathbf{x}} \stackrel{\text{def}}{=} \{x_1, ..., x_n\}$, $\underline{\mathbf{a}} \stackrel{\text{def}}{=} \{a_1, ..., a_h\}$, a Boolean function $F(\underline{\mathbf{x}})$, a variable placement $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ and some gap $g_{min} > 0$ with the same definitions of Section 5.1, the problem of finding a penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}})$ as in (5.1) corresponds to the following quantified SMT problem:

$$\bigwedge_{i,j} \theta_i \in [-2,2],\ \theta_{ij} \in [-1,1]$$
$$\wedge\ \forall \underline{\mathbf{x}}. \left[\begin{array}{l} (\ \ F(\underline{\mathbf{x}}) \rightarrow \exists \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0)) \wedge \\ (\ \ F(\underline{\mathbf{x}}) \rightarrow \forall \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq 0)) \wedge \\ (\neg F(\underline{\mathbf{x}}) \rightarrow \forall \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min})) \end{array} \right] \tag{6.1}$$

We can apply Shannon's expansion to the previous formula (6.1) to get the following quantifier-free SMT($\mathcal{LRA}$) problem:

$$\Phi(\underline{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \bigwedge_{z_i \in \underline{\mathbf{x}}, \underline{\mathbf{a}}} (-2 \leq \theta_i) \wedge (\theta_i \leq 2) \wedge \bigwedge_{\substack{z_i, z_j \in \underline{\mathbf{x}}, \underline{\mathbf{a}} \\ i<j}} (-1 \leq \theta_{ij}) \wedge (\theta_{ij} \leq 1) \tag{6.2}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0) \tag{6.3}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq 0) \tag{6.4}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) \geq g_{min}). \tag{6.5}$$

Then, the same formula $\Phi(\underline{\boldsymbol{\theta}})$ can be used for the optimization version of the encoding problem. In particular, the problem of finding the penalty function $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ that maximizes the gap g_{min} is the OMT($\mathcal{LRA}$) maximization problem $\langle \Phi(\underline{\boldsymbol{\theta}}), g_{min} \rangle$. Notice that, since a maximum g_{min} is sought, the OMT solver implicitly normalizes $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. Furthermore, if $\underline{\mathbf{a}} = \emptyset$, then the OMT($\mathcal{LRA}$) maximization problem $\langle \Phi(\underline{\boldsymbol{\theta}}), g_{min} \rangle$ reduces to a linear program because the disjunctions in (6.3) disappear.

To force $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to be an *exact* penalty function, we add another conjunct in the quantified part of (6.1):

$$(\neg F(\underline{\mathbf{x}}) \rightarrow \exists \underline{\mathbf{a}}.(P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = g_{min})), \tag{6.6}$$

This conjunct forces $P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ to be exactly equal to g_{min} on all counter-models for at least one $\underline{\mathbf{a}}$. The quantifier-free version of (6.6) can be then trivially obtained by adding the relevant constraints to the Shannon's expansion in (6.2)-(6.5):

$$(6.2)\text{-}(6.5) \;\wedge \bigwedge_{\{\underline{\mathbf{x}}\in\{-1,1\}^n | F(\underline{\mathbf{x}})=\bot\}} \bigvee_{\underline{\mathbf{a}}\in\{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = g_{min}). \quad (6.7)$$

6.2 Improving SMT Encoding using Variable Elimination

Disclaimer: The contents of this section is not a contribution of this thesis; rather, this was already proposed in [13]. We report it here to make the narration self-contained, since the techniques described here have been actually implemented and used in our work.

In the previous SMT/OMT($\mathcal{LRA}$) formulation (6.2)-(6.5), $\Phi(\underline{\boldsymbol{\theta}})$ grows exponentially with the number h of hidden variables. For practical purposes, this typically implies that $\Phi(\underline{\boldsymbol{\theta}})$ becomes impractical to solve when the number of ancillas exceeds about 10. By using a more efficient formulation we can reduce the issue. Here we describe an alternative SMT formulation whose size grows slower when increasing h. This formulation grows as $O(h2^{\mathbf{tw}})$, where **tw** is the tree-width (as defined in Section 3.2.3) of the subgraph of G induced by the ancillary variables in the variable assignment, G_a. For Chimera graphs, this means that even when h is as large as 32, **tw** is at most 8 and therefore still of tractable size.

The reformulation is based on the use of the **variable elimination** technique [39] on G_a to solve an Ising problem. This method is a form of dynamic programming, which involves storing tables in memory describing all possible outcomes of a sub-problem. Rather than using numerical

tables, our formulation replaces each of its entries with a $\mathcal{LRA}$ variable constrained by linear inequalities. Each ancilla is processed in a specific order, called variable elimination order. When the tree-width is **tw**, there exists a variable elimination order guaranteeing that each table contains at most $O(2^{\mathbf{tw}})$ entries. In principle, we need to solve an Ising problem for each $\underline{\mathbf{x}} \in \{-1,1\}^n$, thus generating $O(2^n h 2^{\mathbf{tw}})$ continuous variables. However many of these continuous variables are equal thanks to the local nature of the variable elimination process. This leads to a reduction of as much as an order of magnitude of the SMT formulation.

We can reformulate equations (6.4)-(6.5) by introducing **witness** binary variables $\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}}) : \{-1,1\}^n \to \{-1,1\}^h$ that will represent ground states for ancillary variables. The equality constraints of (6.3) are then modified into the form $P_F(\underline{\mathbf{x}}, \underline{\boldsymbol{\beta}}(\underline{\mathbf{x}})|\theta) = 0$. Thus, we can rewrite $\Phi(\underline{\boldsymbol{\theta}})$ as the SMT problem $\Phi(\underline{\boldsymbol{\theta}}, \underline{\boldsymbol{\beta}})$ in the following way:

$$\begin{aligned}\Phi(\underline{\boldsymbol{\theta}}, \underline{\boldsymbol{\beta}}) \;&\stackrel{\text{def}}{=}\; (6.2) \wedge (6.4) \wedge (6.5) \\ &\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \; \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} \big(\, (\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}}) \equiv \underline{\mathbf{a}}) \wedge (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = 0) \, \big).\end{aligned}$$

At first, consider the case of a penalty function with no ancilla-ancilla interaction, thus the graph G_a has no edges. If, for $i = 1, \ldots, h$, we define $f_i(a_i|\underline{\mathbf{x}})$ as the contribution of ancilla a_i to the penalty function in the following way:

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

Then, conversely, we can rewrite the penalty function in the following way:

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

Here $c(\underline{\mathbf{x}})$ does not depend on the ancillary variables. Thus we have the following property:

$$\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}}) = c(\underline{\mathbf{x}}) + \sum_{i=1}^{h} \min_{a_i \in \{-1,1\}} f_i(a_i|\underline{\mathbf{x}}). \tag{6.8}$$

When $\underline{\boldsymbol{\theta}}$ is known solving (6.8) is straightforward. However, since in the encoding $\underline{\boldsymbol{\theta}}$ is a variable, we express the contribution $\min_{a_i \in \{-1,1\}} f_i(a_i|\underline{\mathbf{x}})$ as a function of $\underline{\boldsymbol{\theta}}$, for each $i = 1, \ldots, h$. Each of these minima will be represented as a continuous variable $m_i(\emptyset|\underline{\mathbf{x}})$ referred to as a **message variable**. To define message variables $m_i(\emptyset|\underline{\mathbf{x}})$ in the SMT problem we can impose the following constraints:

$$m_i(\emptyset|\underline{\mathbf{x}}) \leq f_i(-1|\underline{\mathbf{x}}) \quad \wedge \quad m_i(\emptyset|\underline{\mathbf{x}}) \leq f_i(1|\underline{\mathbf{x}}). \tag{6.9}$$

When $F(\underline{\mathbf{x}}) = \bot$ the message variables are lower bounds on the true minima of (6.8). Thus, to enforce the constraints of (6.5) we need simply add the equivalent constraint:

$$c(\underline{\mathbf{x}}) + \sum_{i=1}^{h} m_i(\emptyset|\underline{\mathbf{x}}) \geq g_{min}. \tag{6.10}$$

When $F(\underline{\mathbf{x}}) = \top$ instead we need to ensure that the message variables take the minima of (6.8). To do this we can make use of the witness variables $\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}})$. To relate the values of $\underline{\boldsymbol{\beta}}(\underline{\mathbf{x}})$ and the message variables $m(\emptyset|\underline{\mathbf{x}})$ we add the following SMT constraints:

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

Here variable $\beta_i(\underline{\mathbf{x}})$ identifies the value of the ancillary variable i that achieves the minimum in (6.8). Finally, to impose (6.3) and (6.4), we require the following condition:

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

Message variables require defining a $\mathcal{LRA}$ variable for each assignment of $\underline{\mathbf{x}}$. Since G is usually sparse though, it is likely that two binary states $\underline{\mathbf{x}}$ and $\underline{\mathbf{x}}'$ agree on the bits adjacent to a fixed ancillary variable i. In this case, it is clear that $m_i(\emptyset|\underline{\mathbf{x}}) = m_i(\emptyset|\underline{\mathbf{x}}')$, and we can use a single message variable for both states. This observation holds even when G_a has edges and will allow us to reduce the size of the SMT problem formulation.

Next we will consider the general case, when $|E(G_a)| > 0$. In what follows, $c(\underline{\mathbf{x}})$ and $f_i(a_i|\underline{\mathbf{x}})$ are defined as above. Given $\underline{\mathbf{x}}$, we want to solve the Ising model $\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$. Variable elimination proceeds in order, replacing one ancillary variable at a time with its message variable. Suppose that ancillary variables are eliminated in the order $h, h-1, \ldots, 1$. Each ancillary variable i is associated with a set $\mathcal{F}_i$ of **factors**, which are functions that depend on ancillary variable i and on a set of ancillary variables that will be eliminated after i, thus with a lesser index. The sets of factors $\mathcal{F}_i$ are called **buckets**. For convenience from now on we will represent each edge as an ordered pair of vertices (i, k) with $k < i$.

Initially, each $\mathcal{F}_i$ consists of simple factors that contains the terms of ancilla-ancilla edges: $f_{i,k}(a_i, a_k) = \theta_{ik}\ a_i a_k$ for $ik \in E(G_a)$, $k < i$. Let $\mathcal{V}_i$ denote the set of ancillary variables involved in the factors of bucket $\mathcal{F}_i$ other than variable i itself , and let $\underline{\mathbf{a}}_{\mathcal{U}}$ denote $\{a_i : i \in \mathcal{U}\}$ where $\mathcal{U}$ is a subset of ancilla and $\underline{\mathbf{a}}$ is a fixed ancilla assignment.

Variable h is eliminated first. Note that once variables in $\mathcal{V}_h$ are instantiated to $\underline{\mathbf{a}}_{\mathcal{V}_h}$, the ground value of variable h is the following:

$$g_h(\underline{\mathbf{a}}_{\mathcal{V}_h}, \underline{\mathbf{x}}) = \min_{a_h} f_h(a_h|\underline{\mathbf{x}}) + \sum_{f_{i,k} \in \mathcal{F}_h} f_{i,k}(\underline{\mathbf{a}}_{\mathcal{V}_h}, a_h). \tag{6.11}$$

Here we abuse notation and write $f_{i,k}(a_i, a_h)$ as $f_{i,k}(\underline{\mathbf{a}}_{\mathcal{V}_h}, a_h)$. $\underline{\mathbf{a}}_{\mathcal{V}_h}$ has $2^{|\mathcal{V}_h|}$ possible values,. These values define a new factor g_h, function of variables $\underline{\mathbf{a}}_{\mathcal{V}_h}$. g_h is then added to the bucket $\mathcal{F}_i$ of variable i with largest index in $\mathcal{V}_h$. As in the case when G_a had no edges, a message variable $m_h(\underline{\mathbf{a}}_{\mathcal{V}_h}|\underline{\mathbf{x}})$ will correspond with (a lower bound on) the minimum of (6.11). For each instantiation of $\underline{\mathbf{a}}_{\mathcal{V}_h}$ we define the message $m_h(\underline{\mathbf{a}}_{\mathcal{V}_h}|\underline{\mathbf{x}})$ as $g_h(\underline{\mathbf{a}}_{\mathcal{V}_h})$.

In this way we proceed in eliminating the next variable in the order $h-1$, and so on iteratively. Eliminating variable i is accomplished by generating the following new factor for each setting of $\underline{\mathbf{a}}_{\mathcal{V}_i}$:

$$g_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) = \min_{a_i} f_i(a_i|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, a_i) \tag{6.12}$$

Then, for each one of the $2^{|\mathcal{V}_i|}$ possible values of g_i we can define the message variable $m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}})$ as $g_i(\underline{\mathbf{a}}_{\mathcal{V}_i})$. As above, factor g_i is then added to bucket $\mathcal{F}_k$ where k is the largest index in $\mathcal{V}_i$. When $V_i = \emptyset$, Equation 6.12 takes the following form:

$$g_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) = \min_{a_i} f_i(a_i|\underline{\mathbf{x}}) + \sum_{f \in \mathcal{F}_i} f(a_i) \tag{6.13}$$

g_i determines the optimal value of a_i; the corresponding message is $m_i(\emptyset|\underline{\mathbf{x}})$. At termination one or more variables will have $V_i = \emptyset$ and the final formulation of the Ising problem $\min_{\underline{\mathbf{a}}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}}|\underline{\boldsymbol{\theta}})$ is the following:

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

Notice that the number of additional messages is $O(\sum_i 2^{|\mathcal{V}_i|})$. When G_a has tree-width t, there is an elimination order for which each $|\mathcal{V}_i| \leq t$,

which, considering Chimera and Pegasus topologies, will be much smaller than 2^h.

As $\underline{\boldsymbol{\theta}}$ are $\mathcal{LRA}$ variables, the messages can be defined as $\mathcal{LRA}$ expressions. Since these message variable represent minimums, we can upper bound the message variables adding the following constraints:

$$\begin{aligned} m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) &\leq f_i(-1|\underline{\mathbf{x}}) + \sum_{f\in\mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, -1) \\ m_i(\underline{\mathbf{a}}_{\mathcal{V}_i}|\underline{\mathbf{x}}) &\leq f_i(1|\underline{\mathbf{x}}) + \sum_{f\in\mathcal{F}_i} f(\underline{\mathbf{a}}_{\mathcal{V}_i}, 1). \end{aligned}$$

As before, when $F(\underline{\mathbf{x}}) = \bot$ the constraint (6.5) can be replaced with the following:

$$c(\underline{\mathbf{x}}) + \sum_{i:\mathcal{V}_i=\emptyset} m_i(\emptyset|\underline{\mathbf{x}}) \geq g_{min}, \tag{6.14}$$

When $F(\underline{\mathbf{x}}) = \top$ instead we must ensure that all the message variables are tightly defined. Let $\underline{\boldsymbol{\beta}}_{\mathcal{U}}(\underline{\mathbf{x}})$ be defined in a way similar to $\underline{\mathbf{a}}_{\mathcal{U}}$: $\underline{\boldsymbol{\beta}}_{\mathcal{U}}(\underline{\mathbf{x}}) = \{\beta_i(\underline{\mathbf{x}}) : i \in \mathcal{U}\}$. Thus, we must add the following constraints for all $\underline{\mathbf{a}}_{\mathcal{V}_i}$:

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

Thus, the final re-formulation of the constraints in equation (6.3) and (6.4) will be the following:

$$c(\underline{\mathbf{x}}) + \sum_{i:\mathcal{V}_i=\emptyset} m_i(\emptyset|\underline{\mathbf{x}}) = 0. \tag{6.15}$$

As noted previously, some message variables will be equivalent to each other. In fact, identifying message variables that have to be the same across

many states $\underline{\mathbf{x}}$ can accomplish a significant additional model reduction. Because G is often sparse, the number of message variables can typically be reduced by an order of magnitude or more.

The variable elimination lower bounds of Equation 6.14 can be relaxed using weaker lower bounds from a linear programming relaxation of the corresponding Ising problem, that requires $O(|V| + |E|)$ continuous variables and inequalities per $\underline{\mathbf{x}}$, $F(\underline{\mathbf{x}}) = \perp$. Consider the following formulation of a QUBO problem:

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

Here variables have been slightly renamed and we use $\{0, 1\}$ for binary variables. We can get express it as an integer linear programming problem, and its relaxation is the following:

$$\text{Minimize} \quad \sum_{i \in V} c_i y_i + \sum_{e \in E} q_e \; z_e \tag{6.16}$$

subject to

$$z_e - y_i - y_j \geq -1 \qquad \text{for each } e = ij \in E,\, i < j \quad (\lambda_e) \tag{6.17}$$

$$-z_e + y_i \geq 0 \qquad \text{for each } e = ij \in E, i < j \quad (\lambda^h_{e,i}) \tag{6.18}$$

$$-z_e + y_j \geq 0 \qquad \text{for each } e = ij \in E, i < j \quad (\lambda^t_{e,j}) \tag{6.19}$$

$$-y_i \geq -1 \qquad \text{for each } i \in V \qquad (\alpha_i) \tag{6.20}$$

$$y_i, z_e \geq 0 \tag{6.21}$$

Its linear programming dual is given by

$$\text{Maximize} \quad -\sum_{e\in E}\lambda_e - \sum_{i\in V}\alpha_i \tag{6.22}$$

subject to

$$\lambda_e - \lambda^h_{e,i} - \lambda^t_{e,j} \leq q_e \quad \text{for each } e = ij \in E, i < j \tag{6.23}$$

$$-\sum_{e:i\in e}\lambda_e + \sum_{e=ik\in E,i<k}\lambda^h_{e,i} + \sum_{e=ki\in E,k<i}\lambda^t_{e,i} - \alpha_i \leq c_i \quad \text{for each } i \in V \tag{6.24}$$

$$\lambda_e, \lambda^h_{e,i}, \lambda^t_{e,i}, \alpha_i \geq 0 \tag{6.25}$$

Notice that if c and q (that can be trivially written as linear expressions of $\underline{\boldsymbol{\theta}}$) are $\mathcal{LRA}$ variables, the dual problem is still composed by $\mathcal{LRA}$ assertions. Thus, we can impose bounds on the minimum value of the QUBO g_{min} with the following set of linear inequalities:

$$-\sum_{e\in E}\lambda_e - \sum_{i\in V}\alpha_i \geq g_{min} \tag{6.26}$$

$$(6.23), (6.24), (6.25) \tag{6.27}$$

This upper bound can help the SMT solver to restrict the search space. Note that we can always take

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

6.3 Inequivalent Variable Placements

So far we have assumed that a variable placement has been provided by the user. The formula $\Phi(\underline{\boldsymbol{\theta}})$ in (6.2)-(6.7) can be built only knowing where each

$z_i \in \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ has been placed. In general there will be many possible placements, but by exploiting the symmetries of G encoded in its automorphism group $Aut(G)$, we can reduce the effective number of placements that have to be tested.

Let $\underline{\mathbf{v}} \stackrel{\text{def}}{=} (v_1, ..., v_{n+h})$ denote a variable placement, so v_i is the vertex of V onto which qubit z_i is placed. Two variable placements $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}' \stackrel{\text{def}}{=} (v'_1, ..., v'_{n+h})$ are **equivalent** if there is a graph automorphism $\phi \in Aut(G)$ that maps $\underline{\mathbf{v}}$ to $\underline{\mathbf{v}}'$; that is, $v_i = \phi(v'_i)$ for all $i \leq n$. If $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ are equivalent, then a penalty function for $\underline{\mathbf{v}}$ can be transformed into a penalty function for $\underline{\mathbf{v}}'$ by applying ϕ, and as $\phi \in Aut(G)$ all architectural constraints remain unchanged. Therefore it is sufficient to enumerate all inequivalent placements in order to search for a penalty function of maximal gap among all placements.

Example 10. *Suppose we want to encode a penalty function with* $n+h = 8$ *variables into an 8-qubit Chimera tile. There exist* $8! = 40320$ *possible variable placements. However, the Chimera tile graph is highly symmetric: any permutation of* $\underline{\mathbf{v}}$ *that either flips horizontal qubits with vertical qubits or reorders horizontal and/or vertical qubits is an automorphism. Each placement is in a class of* $|Aut(G)|$ *equivalent placements, and the order of* $Aut(G)$ *is* $2 \times 4! \times 4! = 1152$*. Since all placements are partitioned in sets of cardinality* $|Aut(G)|$*, there are at most* $8!/|Aut(G)| = 35$ *inequivalent placements to consider.*

This equivalence relation is dependent only on the hardware graph and assumes no symmetries on the penalty function. In general though, ancillary qubits are interchangeable and many Boolean functions are highly symmetric. The notion of variable placement equivalence can be extended by taking advantage of NPN-equivalence. We define variables x_1 and x_2 in a Boolean function F to be **NPN-symmetric** if an equivalent formula can be produced by swapping the two variables and negating any of the two

variables. These symmetries, similarly to automorphism groups in graphs, define an equivalence relation between variables of F: for any x_i and x_j in the same equivalence class, there is a permutation-and-negation that maintains F and maps x_i to x_j while keeping other variables the same. We say that two variable placements $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ are **equivalent up to NPN-symmetry** if there is a graph isomorphism ϕ of G and a NPN-symmetry permutation σ of F such that $\underline{\mathbf{v}} = \sigma(\phi(\underline{\mathbf{v}}'))$. That is, for all $i \leq n$, there exists a $j \leq n$ such that x_i and x_j are NPN-symmetric and $v_i = \phi(v'_j)$.

Example 11. *Consider the following function:*

$$\text{AND}(x_1, \ldots, x_4) = x_1 \wedge x_2 \wedge x_3 \wedge x_4$$

The variables $x_1, \ldots, x_4$ in AND *are all NPN-symmetric. Suppose we have a penalty function with $h = 4$ auxiliary variables with a placement on the 8-qubit Chimera tile. Just considering graph automorphisms, it suffices to consider 35 variable placements. Adding NPN-symmetry into consideration we notice that any two variable placements $\underline{\mathbf{v}}$ and $\underline{\mathbf{v}}'$ that map the same number of x_i's to horizontal qubits are equivalent, since there is a NPN-symmetry that will map the x_i's in $\underline{\mathbf{v}}$ to the x_i's in $\underline{\mathbf{v}}'$. Moreover, a placement mapping $k \leq 4$ of the x_i's to horizontal qubits is equivalent to one mapping $4 - k$ of the x_i's to horizontal qubits, by swapping horizontal and vertical qubits. As a result, there are only 3 inequivalent variable placements up to NPN-symmetry to consider, in which 0, 1 or 2 of the x_i's are mapped to horizontal qubits.*

We can use vertex-coloured graph isomorphisms to check for equivalent variable placements. Recall that a vertex coloring c is a function on vertices of the graph. Two vertex-coloured graphs (G, c) and (G', c') are **vertex-coloured graph-isomorphic** if there is an isomorphism ϕ mapping $V(G)$ to $V(G')$ in such a way that every vertex of G is mapped to a vertex of the same colour in G' ($\forall v \in V$, $c'(\phi(v)) = c(v)$). Using a variable placement

$\underline{\mathbf{v}}$ and NPN-symmetry, we can define a vertex-coloring c of the hardware graph G as follows:

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

In practice we use the software package NAUTY [67] to compute a canonical form for each vertex-colored graph and check if two variable assignment have the same canonical form. NAUTY is based on vertex-coloured canonical forms and uses them natively as part of its graph isomorphism algorithm. It is very efficient and is able to compute canonical forms for graphs with thousands of vertices.

6.4 Placing Variables & Computing Penalty Functions via SMT/OMT($\mathcal{LRIA} \cup \mathcal{UF}$).

The formula $\Phi(\underline{\boldsymbol{\theta}})$ in (6.2)-(6.7) can be built only after a variable placement is chosen, so that each variable $z_j \in \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ has been previously placed in some vertex $v_j \in V$. The variable elimination technique also requires a specific variable placement to define its constraints. As an alternative to enumerating equivalent variable placements, we can encode the constraints for a variable placement by means of SMT/OMT($\mathcal{LRIA} \cup \mathcal{UF}$)(i.e., the combined theories of linear arithmetic over rationals and integers plus uninterpreted function symbols, as seen in Section 3.3.5). This allows us to use the SMT solver to search a penalty function over all legal variable placements, instead of having to enumerate all possible placements and calling multiple times the solver on each placement.

Suppose that we want to find the penalty function of a Boolean function F that is relatively small. We represent the $n+h$ vertices of the hardware graph as indices $V \stackrel{\text{def}}{=} \{1, ..., n+h\}$, and we introduce a list of $n+h$ vari-

ables $\underline{\mathbf{v}} \stackrel{\text{def}}{=} \{v_1, ..., v_{n+h}\}$ such that $v_i \in V$. v_i will represent the vertex into which z_j is placed. We can guarantee that no vertices is mapped to multiple qubits with the standard SMT constraint $\mathsf{Distinct}(v_1, ..., v_{n+h})$. Then we rewrite the encoding formula (6.1) replacing the θ_i and θ_{ij} for biases and couplings, with a more complex construct. We introduce the **uninterpreted function symbols** $\mathsf{b} : V \longmapsto \mathbb{Q}$ ("bias map") and $\mathsf{c} : V \times V \longmapsto \mathbb{Q}$ ("coupling map"). Each bias θ_j will change to $\mathsf{b}(v_j)$ and each coupling θ_{ij} to $\mathsf{c}(v_i, v_j)$ s.t $v_i, v_j \in [1, .., n + h]$.

Conversely, the SMT($\mathcal{LRA}$) problem (6.2)-(6.5) can be rewritten into the $SMT(\mathcal{LRIA} \cup \mathcal{UF})$ problem (6.28)-(6.38) shown in Listing 1. In this formula the constraint (6.34) is necessary because we could have $\mathsf{c}(v_i, v_j)$ s.t. $v_i > v_j$. We can create placements for exact penalty function as well, by adding Equation 6.39 to the SMT problem:

$$\bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \quad \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) = g_{min}) \tag{6.39}$$

Furthermore, notice that the solution to the OMT($\mathcal{LRA} \cup \mathcal{UF}$) problem $\langle \Phi(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}), g_{min} \rangle$ provides the optimal values of biases b and couplings c, but for all possible variable placements.

We follow with an example of this SMT encoding technique.

Example 12. *Consider the Boolean function:*

$$F(\underline{\mathbf{x}}) \stackrel{def}{=} (x_3 \leftrightarrow (x_1 \wedge x_2))$$

with $\underline{\mathbf{x}} \stackrel{def}{=} \{x_1, x_2, x_3\}$ *and* $\underline{\mathbf{a}} \stackrel{def}{=} \{a_1\}$*, on a Chimera tile subgraph with 2 horizontal and 2 vertical qubits, so* $V \stackrel{def}{=} \{1, 2, 3, 4\}$ *and* $E \stackrel{def}{=} \{(1, 3), (1, 4), (2, 3), (2, 4)\}$*. Finally,* z_1*,* z_2*,* z_3 *and* z_4 *will denote* x_1*,* x_2*,* x_3 *and* a_1 *respectively (as* $\underline{\mathbf{z}} = \underline{\mathbf{x}} \cup \underline{\mathbf{a}}$ *), and* $\underline{\mathbf{v}} \stackrel{def}{=} \{v_1, v_2, v_3, v_4\}$ *denotes the variable placement. In*

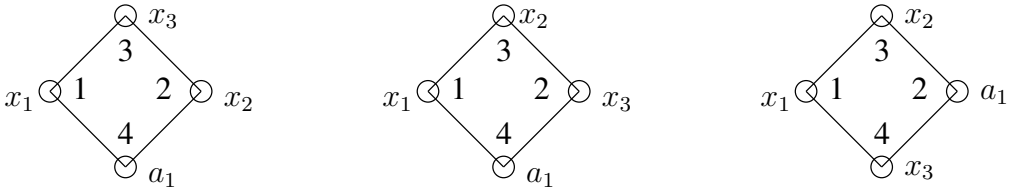


Figure 6.1: 3 possible placements of $\underline{\mathbf{z}} \stackrel{\text{def}}{=} \{x_1, x_2, x_3\} \cup \{a_1\}$ into a 4-qubit Chimera half-tile. All $4! = 24$ placements are equivalent to one of these.

(6.28)-(6.38) *we will have the following:*

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

A possible solution is given below. The variable placement can be seen in Figure 6.1 (center).

g	v_1	v_2	v_3	v_4
2	1	3	2	4

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

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

When using an SMT/OMT solver to search for penalty functions across all variable placements as in (6.28)-(6.38), we may restrict the search space by considering only one variable placement from each equivalence class

under the automorphisms of G. We can extend the previous examples in the following way.

Example 13. *In Example 10, we reduced the number of variable placements under consideration for a Chimera tile from* $8! = 40320$ *to* $\binom{7}{3} = 35$ *using automorphisms. The simplest way to force the SMT/OMT solver to restrict the search is adding as a constraint the disjunction of the following 35 cubes, each representing one placement.*

$$
\begin{aligned}
&(\overbrace{v_1 = 1 \wedge v_2 = 2 \wedge v_3 = 3 \wedge v_4 = 4}^{\substack{\textit{subset of } \{v_1,\ldots,v_8\} \\ \textit{mapped to horizontal qubits}}} \wedge \overbrace{v_5 = 5 \wedge v_6 = 6 \wedge v_7 = 7 \wedge v_8 = 8}^{\substack{\textit{complementary subset} \\ \textit{mapped to vertical qubits}}}) \vee \\
&(v_1 = 1 \wedge v_2 = 2 \wedge v_3 = 3 \wedge v_5 = 4 \wedge v_4 = 5 \wedge v_6 = 6 \wedge v_7 = 7 \wedge v_8 = 8) \vee \\
&\ldots \\
&(v_1 = 1 \wedge v_6 = 2 \wedge v_7 = 3 \wedge v_8 = 4 \wedge v_2 = 5 \wedge v_3 = 6 \wedge v_4 = 7 \wedge v_5 = 8).
\end{aligned}
$$

Note that if we add this constraint, the first conjunction in (6.32) *can be dropped.*

Example 14. *Picking the problem of Example 12, we have* $4! = 24$ *possible placements on a half-tile. Considering symmetries as above, there are only* 3 *inequivalent placements, which are shown in Figure 6.1. We can then add the following disjunction:*

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

$$\Phi(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \mathsf{Range}(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \wedge \mathsf{Distinct}(\underline{\mathbf{v}}) \wedge \mathsf{Graph}() \tag{6.28}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \geq 0) \tag{6.29}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \top\}} \bigvee_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) = 0) \tag{6.30}$$

$$\wedge \bigwedge_{\{\underline{\mathbf{x}} \in \{-1,1\}^n | F(\underline{\mathbf{x}}) = \bot\}} \bigwedge_{\underline{\mathbf{a}} \in \{-1,1\}^h} (P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \geq g_{min}) \tag{6.31}$$

where:

$$\mathsf{Range}(\theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \bigwedge_{1 \leq j \leq n+h} (1 \leq v_j) \wedge (v_j \leq n + h) \tag{6.32}$$

$$\wedge \bigwedge_{1 \leq j \leq n+h} (-2 \leq \mathsf{b}(j)) \wedge (\mathsf{b}(j) \leq 2) \tag{6.33}$$

$$\wedge \bigwedge_{1 \leq j \leq n+h} (\mathsf{c}(j,j) = 0) \wedge \bigwedge_{1 \leq i < j \leq n+h} (\mathsf{c}(i,j) = \mathsf{c}(j,i)) \tag{6.34}$$

$$\wedge \bigwedge_{1 \leq i < j \leq n+h} (-1 \leq \mathsf{c}(i,j)) \wedge (\mathsf{c}(i,j) \leq 1) \tag{6.35}$$

$$\mathsf{Distinct}(v_1, ..., v_{n+h}) \stackrel{\text{def}}{=} \bigwedge_{1 \leq i < j \leq n+h} \neg(v_i = v_j) \tag{6.36}$$

$$\mathsf{Graph}() \stackrel{\text{def}}{=} \wedge \bigwedge_{\substack{1 \leq i < j \leq n+h \\ \langle i,j \rangle \notin E}} (\mathsf{c}(i,j) = 0) \tag{6.37}$$

$$P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \theta_0, \mathsf{b}, \mathsf{c}, \underline{\mathbf{v}}) \stackrel{\text{def}}{=} \theta_0 + \sum_{1 \leq j \leq n+h} \mathsf{b}(v_j) \cdot z_j + \sum_{1 \leq i < j \leq n+h} \mathsf{c}(v_i, v_j) \cdot z_i \cdot z_j. \tag{6.38}$$

Listing 1: Complete formulation of SMT ($\mathcal{LRIA} \cup \mathcal{UF}$) encoding with automatic placement.

Chapter 7

Encoding Larger formulas

7.1 Introduction

In Section 5.1 we pointed out that large Boolean functions cannot be encoded using the SMT technique described in the previous chapter, as the number of constraints and variables in the model grows exponentially with the number of variables n of the Boolean function. Thus in this chapter we will describe the complete "divide-and-conquer" approach that will be used on complex SAT or maxSAT problems

This approach consists in pre-computing a library of encoded Boolean functions using the techniques of the previous chapter, and rewriting an input Boolean function $F(\underline{\mathbf{x}})$ as a conjunction of pre-encoded components $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$. The pre-computed penalty functions $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$ for these components may then be combined using chains as described in Section 5.5. In terms of effectiveness, this method has been shown to outperform other encoding methods when encoding Boolean circuits (see also [13, 14, 90]). The general schema of this approach is shown in Figure 7.1.

Again, we closely follow the steps of [16]. Each stage will be described in the next sections.

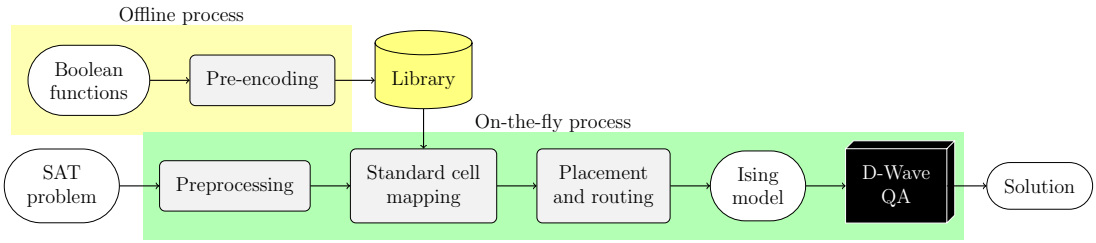


Figure 7.1: Schema of the divide-and-conquer encoding process.

7.2 Pre-encoding

Pre-encoding is performed ahead-of-time on a collection of small selected Boolean functions. The goal of this phase is to generate a database of efficient encodings for Boolean functions that appear commonly in input functions. The AND gate is sufficient to encode all possible boolean functions (see section 3.3.2), and we can add several gates, such as all 2,3 or 4 input basic gates, half and full adders and many others. Finding these encodings can be computationally expensive, but it needs only to be performed once for each NPN-inequivalent Boolean function.

There may exist many different penalty functions for any Boolean function with different trade-offs. Penalty functions with more ancilla have larger gaps, but can result in longer chains, so choosing the best option is not trivial. A reasonable heuristic is to choose the smallest penalty functions with the same gap of a chain $g_{min} = 2$.

We can improve encoding results using knowledge of the target hardware graph. For example, a natural choice of pre-computed gates for Chimera graphs is the set of Boolean functions that can fit in single 8-qubit tile. In particular, all 3-input, 1-output gates (all 3-feasible cuts) can be inserted in one tile.

7.3 Preprocessing

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

In Section 3.3.2 we introduced the and-inverter graph representation of circuits. Most of the state-of-the-art in preprocessing uses AIGs as data structure to handle Boolean formulas (see Section 3.3.2). Preprocessing is a well-studied problem, and mature algorithms are available [69, 71]. For our purposes we will use **DAG-aware minimization** as implemented by the logic optimizer ABC [21].

DAG-aware minimization attempts to find an AIG equivalent to the original with a minimal number of nodes by repeatedly identifying small sub-graphs that can be replaced with a smaller sub-graph without affecting the output. DAG-aware minimization identifies a 4-feasible cut C for a node and replaces the subgraph induced by C with the smallest subgraph representing the same Boolean function. There are 222 NPN-inequivalent 4-input Boolean functions, a small enough number to be checked exhaustively. See [44] for more details.

7.4 Standard cell mapping

The goal of the standard cell mapping phase is to decompose the previously-simplified function $F(\underline{\mathbf{x}})$ into component functions $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^k})$ where F_k is taken from the library of pre-encoded functions. The simplest method is to build the library out of the gates used in the desired formula, so to ensure each $F_k(\underline{\mathbf{x}^k})$ is found in the library possibly decomposing missing function

into simpler components. However, there are more advanced techniques that have been devised for digital logic synthesis. **Technology mapping** is the process of mapping a Boolean circuit to a network physical gates that can be placed in a digital circuit [44, 70].

A technology mapping algorithm takes as input the library gates $F_k(\underline{\mathbf{x}^k})$, each with an associated cost, creates a Boolean gate network that is equivalent to $F(\underline{\mathbf{x}})$ and attempts to minimize the total cost of the components in $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^k})$. When applied to digital circuits, technology mapping is often used to reduce issues such as chip are a used, circuit delay and load, and takes them into account in the cost calculation. Delay and load do not play a role in the context of QAs, while area minimization implies minimization of qubits used in the encoding, and is thus important to increase effectiveness and simplify the subsequent placement and routing phase. We define the cost of a gate F_k to be the number of qubits used by the penalty model P_{F_k}, so that the total cost $F(\underline{\mathbf{x}}) = \bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}^k})$ is the number of qubits used to represent $F(\underline{\mathbf{x}})$ before adding chains.

Here, we apply the technology mapping algorithm outlined in [44]. The technique relies on the definition of k-feasible cuts outlined in Section 3.3.2. Let be $F(\underline{\mathbf{x}})$ a Boolean formula in AIG form D. A **mapping** M of an AIG D is a function that maps every node a_i of D to a k-feasible cut $M(a_i)$. We say a_i is **active** when $M(a_i)$ is not trivial and **inactive** otherwise. A mapping M is **proper** if:

1. every output a_o of D is active;
2. if a_i is active, then every $a_j \in M(a_i)$ is active;
3. if a_j is not an output and $a_j \notin M(a_i)$, then a_j is inactive.
4. for each active node a_k there is a Boolean function $F_k(\underline{\mathbf{z}^k})$ represented by the cut $M(a_k)$ such that $F_x(\underline{\mathbf{z}^k})$(or a NPN-equivalent) appears in the pre-computed library.

Each active node a_j will become the output of a gate in our library, while $M(a_j)$ will become its inputs. Thus, a successful mapping decomposes the original Boolean function $F(\underline{\mathbf{x}})$ in the following way (as in Section3.3.1)

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

The simplest proper mapping is the trivial mapping, in which each a_i is mapped to the cut consisting of its two input nodes. The trivial mapping is equivalent to a naive encoding of the function $F(\underline{\mathbf{x}})$ using AND gates (Example 4 shows it can be trivially assumed that AND is in the pre-computed library).

The algorithm in [44] iteratively improves mapping M in the following way. Each node a_i is associated with a list $L(a_i)$ of k-feasible cuts, ordered by cost. Traverse the graph from inputs $\underline{\mathbf{x}}$ to primary output a_o. For each node a_i, recalculate the costs of the cuts in $L(a_i)$ based on the costs of its children. Next, if a_i is active and the current cut $M(a_i)$ is not the cut in $L(a_i)$ of lowest cost, update $M(a_i)$. To do this, first inactivate a_i (which recursively inactivates nodes in $M(a_i)$ if they are no longer necessary) and then reactivate a_i (which reactivates nodes in $M(a_i)$, also recursively). The cost of a cut is calculated using the **area-flow heuristic**. It is calculated by summing the cost of using a particular gate, plus the best estimate cost of activating all the nodes in the cut $M(a_i)$.

Finally, standard cell mapping algorithms typically take advantage of NPN-equivalence of Boolean functions [70], so the library of available Boolean functions need only contain one representative from each NPN-equivalence class. [68]. This is done in a way similar to the one described in Section 5.1.

7.5 Placement and routing

After the technology mapping phase $F(\underline{\mathbf{x}})$ is decomposed into smaller functions $\bigwedge_{k=1}^{K} F_k(\underline{\mathbf{x}}^k)$, each with known penalty functions $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$. The last encoding step provides the final variable placement necessary to embed the formula onto the QA hardware as seen in Equation 5.7. In Section 5.5 we outlined various methods to perform embedding. Here we will use the placement and routing technique. This process has two parts: **placement**, in which each $P_{F_k}(\underline{\mathbf{x}}^k, \underline{\mathbf{a}}^k | \underline{\boldsymbol{\theta}}^k)$ is assigned to a disjoint subgraph of the QA hardware graph; and **routing**, in which chains are built in order to ensure that distinct qubits x_i and x_i' representing the same variable take consistent values (using equivalence constraints with penalty functions of the form $1 - x_i x_i'$). Both placement and routing are very well-studied in design of digital circuits [12]. This stage is a computational bottleneck for encoding large Boolean functions. Some placement and routing approaches have been outlined in Section 4.2.

In Section 4.2 we have seen several approaches for placement and routing, where the placement and routing stages of embedding are typically performed separately. However, given the currently available Chimera and Pegasus architectures with limited qubits a combined place-and-route algorithm can be more effective [14]. The approach chosen here is using a modified Bonn-routing with a custom heuristic for placement. As the placement heuristic relies on routing, the latter algorithm will be described first.

First, routing in the context of embedding differs from the one in digital circuit design, mainly because that vertices (qubits) are the sparse resource that variables compete for, rather than edges. As a result, vertex-weighted Steiner tree algorithms should be used rather than edge-weighted ones and vertex-weighted Steiner is harder to approximate [23, 60]. In

practice, simple algorithms for edge-weighted Steiner trees can be adapted to the vertex-weighted problem. This section describes a modification of the routing algorithm BonnRoute [50] for vertex-weighted Steiner trees.

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

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

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

To solve the min-max resource allocation, first a weighted-Steiner tree approximation algorithm is used multiple times to approximate the convex hull. After each Steiner tree is generated, the weights of the vertices in that Steiner tree are increased to discourage future Steiner trees from reusing them (see Listing 2 for details). The generated trees form a probability distribution over the Steiner trees for each x_i.

The BonnRoute algorithm produces good approximate solutions in reasonable time. More precisely, if vertex-weighted Steiner tree approximations are approximated within a factor σ of optimal, for any $\omega > 0$ Listing 2 computes a $\sigma(1+\omega)$-approximate solution to min-max resource allocation problem using $O((\log |V|)(n+|V|)(\omega^{-2} + \log\log |V|))$ tree approxima-

Require: Graph G, Steiner tree terminals $\{T_1, \ldots, T_n\}$, number of iterations t, weight penalty $\alpha > 1$

Ensure: For each i, a probability distribution p_{i,S_i} over all Steiner trees S_i for terminals T_i

```
function BonnRoute(G,{T_1, ..., T_n})
    for each v ∈ V(G) do
        w_v ← 1
    end for
    for each Steiner tree S_i for terminals T_i, i ∈ [n] do
        z_{i,S_i} ← 0
    end for
    for j from 1 to t do
        for each i ∈ [n] do
            Find a Steiner tree S_i for terminals T_i with vertex-weights w_v
            z_{i,S_i} ← z_{i,S_i} + 1
            w_v ← w_v * α for all v ∈ S_i
        end for
    end for
    Return p_{i,S_i} ← z_{i,S_i}/t
end function
```

Listing 2: BonnRoute Resource Sharing Algorithm [50].

tions [72].

Once a solution to the min-max resource allocation has been found, a solution to the original routing problem is recovered by randomized rounding on the probability distributions.

When applying routing to graphs with a Chimera or Pegasus topology we can exploit the symmetry within each unit tile. In these cases it is convenient to work with a **reduced graph** in which the horizontal qubits in each unit tile are identified as a single qubit, and similarly for the vertical qubits. As a result the scale of the routing problem is reduced by a factor of 4. This necessitates the use of vertex capacities within the routing

algorithm (each reduced vertex has a capacity of 4), and variables are assigned to individual qubits within a tile during a secondary, detailed routing phase.

Given a partial embedding, it is possible to estimate the cost of placing a new F_k in a particular position, by checking the shortest distance that each new chain would have to traverse. Together with an estimation of which sub-problem contribute the most to a particular embedding, this allows us to improve a placement using a **rip-and-reroute** technique [14]. In rip-and-reroute, a constraint (usually the most expensive) is removed from the embedding and moved into the best available spot. We can also make use of tabu list to avoid repeating movements on a cluster of badly-placed gates.

For placement we use an iterative approach based on rip-and-reroute. The graph of F_k components is traversed breadth-first; Each element is placed using the previously mentioned heuristic. For each new element placed, we try to improve the placement early by performing a single rip-and-reroute step. The choice of the first component to be placed can be random, or a metric can be chosen. In the libraries of Chapter 8 the F_k with the highest betweenness [48] is used so that at the beginning, the most "central" sub-problem is placed in a "central" spot in the hardware.

Chapter 8

Implementation

8.1 Introduction

All the steps outlined in the previous chapters have been implemented as various Python programs and libraries. A first implementation targeted Python version 2, then all the code has been ported to support Python 3 as well. Some of the tasks can be computing intensive, so it is advisable to use the `PyPy` implementation of Python as interpreter. As the most computationally intensive step, placement and routing, is critical for overall speed, the author wrote a high performance version of the algorithm in C++ using the Boost libraries. Notice that the Boost Python library does not allow linking with `PyPy`, thus it currently works with the standard `CPython` interpreter only. All code is released under MIT license and available at `https://bitbucket.org/StefanoVt/`.

This chapter describes each library and explains their implementation details. First the basic file formats are described, together with the relative handling libraries. Then the implementation of each step of Chapter 7 is described.

8.2 Basic libraries

8.2.1 SMT-lib

SMT-lib is the standard format for interacting with SMT (and OMT) solvers. It uses **S-expressions**, a structured data format inherited from lisp. S-expressions consists of atoms that can be symbols or numbers or of lists of sub-S-expressions separated by a space and surrounded by parentheses. SMT-lib semantics are specified by the official standard [8].

```
(set-logic QF_UF)
(declare-fun p () Bool)
(assert (and p (not p)))
(check-sat)
```

Listing 3: Examples of S-expressions from a SMT-lib file.

The most complete Python library for SMT-lib is `pySMT`, but it is not suited for our goal because it does not support OMT and interaction with SMT solvers is hard to customize. Rather than using `pySMT` I wrote a simpler library, called `smtutils`, that allows to transform python expressions into SMT-lib S-expressions and present a simple wrapper to call a SMT solver as an external process. Listing 3 shows a small example of using `smtutils` for producing SMT-lib formulas and calling a solver. Python expression are automatically converted to S-expressions, and a simple wrapper spawn a SMT solver subprocess, sends the formula to it and waits for the solution.

8.2.2 RBC library

Another fundamental task is to store and manipulate Boolean formulas and circuits. To handle that, I wrote a library called `pyrbc`. The main two ways

to represent Boolean circuits are as a network of functions (represented as truth tables) and as and-inverter graphs. Boolean networks are **directed acyclic graphs (DAGs)** where nodes are Boolean functions, and inputs and outputs are connected through edges. As we have seen, and-inverter graphs can be considered Boolean networks as well, but they have a single node type and edges can be negated. In section 7.3 we have seen that the state-of-the-art of circuit simplification uses and-inverter graphs. For reference, Figure 3.7 shows a simple AIG.

Another important representation of Boolean circuits is as a **reduced boolean circuit (RBC)**. These are similar to AIGs, but use two types of nodes, XOR and AND. The characteristic of RBC is not a particular encoding format, but their circuit simplification technique. While a RBC is being constructed, nodes are stored in a hash table, so that identical sub-graphs can be de-duplicated. This technique is simple and does not detect functionally-equivalent nodes but is fast and effective. The `pyrbc` library handles AIGs using this reduction technique.

As a file format for Boolean networks the code will use the BLIF format [22], while AIGs are usually stored using the AIGER standard format [54]. The BLIF format is a simple text format generally used in digital circuit specification. It allows the definition of function networks, both from a standard function library and as truth tables. The AIGER format encodes and-inverter graphs, either as binary (more concise and useful for larger circuits) or text (clearer for humans to understand). Listings 5 and 6 compare the same circuit under the same format.

The main components of `pyrbc` are the main nodes implementation, the node database for RBC reduction and the import-export libraries for AIGER files and external graph libraries. The main nodes are implemented with the goal of easy initialization and manipulation. The database interface allows a straightforward creation of graphs. It is possible convert the

circuit into a `networkx` graph in order to run graph algorithms, and read and write AIGER files. Listing 7 shows an example of using `pyrbc` to read an AIGER file and traverse its nodes.

8.2.3 GENLIB format

Prepared gates are saved in a text file in GENLIB format. GENLIB is the main format used by ABC for storing the gates library for technology mapping. GENLIB file contain a text-base list of gates, with information such as name, formula representation, area used, pin maximum load and pin delays.

The format is tought for technology mapping of digital circuits, as it contains information about electrical loads and delays that have no used for SATtoIsing encoding. In the same way the format does not contain info about the penalty function of a gate. In order to maintain interoperability with ABC, the pre-encoding step produces a slightly modified format, where the penalty function of a gate is stored in JSON format in a GENLIB comment next to the description, and pin load/delay information is set to 0. Such a GENLIB file can be directly used to perform technology mapping using ABC without need to modify the software. Listing 8 shows an example of such a format.

8.2.4 Graph Algorithms

Some graph algorithms are used in several contexts during the encoding process. k-feasible cut (in short, k-cut) enumeration is an important step in technology mapping. The algorithm for k-cut enumeration has been outlined in Section 3.3.2. This algorithm has been implemented in the `kcut` Python library, relying on the `networkx` graph library. It uses dynamic programming to avoid re-calculating cuts for the same nodes and it is

fairly efficient.

Graph isomorphism is useful for symmetry reduction in SMT pre-encoding and during technology mapping. As hinted in Section 6.3, the main tool for checking Graph isomorphism is the NAUTY library, with a custom wrapper for ease of use with `networkx` classes.

8.3 Pre-encoding

8.3.1 SMT encoding

Given a Boolean relation as a Python function a penalty function can be sought using the `pfencoding` Python library. `pfencoding` provides some utilities to express penalty functions and their constraints as Python and SMT-lib expression. It provides a `PenaltyFunction` and a `MovablePenaltyFunction` to organize the variables used in Equations 6.2 and 6.38. Furthermore a set of classes represents constraints, such as `RangeConstraint` or `ArchitectureConstraint`. A class `ExpansionGapConstraint` provides the constraints on the penalty function obtained by Shannon expansion, while `VariableEliminationConstraint` provides an implementation of Section 6.2.

A utility module, called `search_pf` aggregates these classes to provide several procedures, such as `search_pf` that check if a penalty function exists, or `search_pf_smallest` that searches for the penalty function with the smallest number of ancillas. Listing 9 shows an example of usage for the library.

8.3.2 Gate selection

The gate that are chosen for addition generally depend on the problem domain that is meant to be tackled. In general it is relatively simple to

enumerate small functions with up to 4 inputs [38], and that provides several gate when limiting oneself to functions fitting to a single Chimera tile.

Alternatively the library `gatecollector` exploits k-cut enumeration to collect the most frequent gates in a dataset of functions of AIG format. The list of k-input function is put in a directory, ordered by decreasing frequency. Listing 10 provides a simple example of using the library for generating a list of gates out of a set of circuits..

8.4 Simplification and technology mapping

The most mature freely-available software for performing technology mapping is ABC. The software has a command line interface that allows the user to perform several operations on circuits. Listing 11 shows an example of using ABC to perform simplification first and then technology mapping. ABC loads AIGER files for circuits and GENLIB files for pre-encoded gate library.

However, ABC is is not tailored for penalty functions so I coded an alternative Python library to perform tech mapping, aptly named `techmapping`. The libraries parses the genlib text databases, and replicates the basic algorithm used by ABC for technology mapping. This implementation is less effective than ABC in technology mapping, and thus the use of ABC is recommended.

The algorithm relies on k-cut enumeration and boolean matching. Given two Boolean functions, the check for NPN-equivalence is called **Boolean matching**. It is possible to perform Boolean mapping by reframing NPN-symmetry as graph isomorphism. This allows the use of NAUTY to check for NPN-equivalence and furthermore it allows the creation of a NPN-canonical Boolean function. This NPN-canonical function allows for an

efficient Boolean matching of pre-encoded libraries.

8.5 Placement and Routing

The placement and routing process as described in Section 7.5 has been implemented in another Python library called `placeandroute`. As this step is the most computationally expensive, the core algorithm has been re-implemented in C++. The library implements Bonn routing as described in the same section. It has been implemented both in Python and C++.

The library works by setting an initial placement, finding the routing and then applying different strategies for improving the placement in turn. First several round of rip-and-reroute are applied, where badly-placed elements are moved. When a certain number of round yield no improvement, a global rerouting step is performed where all chains are removed and an alternative routing is sought. The effort placed by the algorithm in finding a better encoding is tuneable by the user. This effort includes precision in approximating Steiner Trees during Bonn routing and number of rip and reroute attempts.

8.5.1 Simplified Graphs and Detailed Routing

In order to exploit symmetries in Chimera and Pegasus graphs placement and routing are performed on simplified graphs. In Chimera a tile is compressed on two nodes, each of capacity 4, while in Pegasus two adjacent nodes are paired in a single node of capacity 2. To join a set of nodes into a single node it is necessary that the nodes compressed together are perfectly interchangeable. Possible placements are defined according on what class of hardware topology is used and how single constraints are defined. In this implementation, a placement is defined as a tile on Chimera and a 4-clique on Pegasus. In this way constraints are placed in two nodes and

a edge of the simplified graph.

This graph simplification is possible because a solution on the simplified graph can be easily converted into a solution on the hardware graph (this is possible when not considering missing qubits, but again merged nodes must be perfectly interchangeable while missing qubits always break symmetries of the hardware graph). The problem of finding a routing from the simplified graph is called **detailed routing**. It is possible to show that for Pegasus and Chimera the detailed routing problem is equivalent to interval graph coloring. Thus if the simplified solution assigns to each simplified vertex a number of variables that is lower than its capacity then a solution exists.

This property can be shown in the following way. Consider a solution on the simplified graph where a variable x_j is mapped to nodes a and b that are connected by an edge Suppose that a and b represent merged nodes A and B respectively . An issue in detailed routing arises only if there is no way to pick qubits from $z_a \in A$ and $z_b \in B$ such that a chain can be created. When $A \cup B$ is a complete bipartite graph we have no issues. $A \cup B$ is not bipartite only along the vertical and horizontal couplings between different tiles. Such connections form linear paths in the simplified graphs of Chimera and Pegasus topologies. The detailed routing problem limited to these paths is equivalence to the graph coloring problem on an interval graph. Interval graph coloring is straightforward (using a greedy algorithm) and is guaranteed to find a solution with the maximum capacity.

This method can provide detailed routing when no qubits are missing in the hardware. If missing qubits are few enough not to break symmetries in the hardware graph (i.e. at most one for each row/column), they can be modeled as single-qubit chains, otherwise the greedy detailed routing can fail.

8.5.2 Placement Heuristics

The library allows many options for deciding an initial placement. The most straightforward option is random initial placement; every gate is placed in a random position and the task of finding a viable solution is left to the placement improvement strategies.

The library provides an alternative initial placement procedure that reuses Bonn routing and rip-and-reroute. Gates are selected using breadth-first search on the Boolean network, starting with the most "central". Each selected element is placed in turn using the heuristic of rip-and-reroute to find the best candidate spot. Furthermore, each time that a new constraint is added, the worst connected gate is moved using rip-and-reroute. This heuristic is quite good in finding a placement but multiple round of rip-and-reroute can significantly improve the placement.

8.6 CLI scripts

To use the libraries above, several scripts are available. Furthermore, two scripts allow for generation of a pre-computed library in GENLIB format, and one scripts perform a divide-and-conquer encoding of an AIG.

For preparing the pre-encoded library, the `preencode.py` script file accepts as arguments the directory containing candidate gates as AIGER files, and a output filename. It uses the `gatecollector` and `pfencoded` library, essentially as shown in Listing 10. The script produces a GENLIB file containing the pre-encoded library. Listing 12 shows the usage documentation.

Regarding the divide-and-conquer encoding, the `encode.py` script perform the complete process. The script requires the input formula, the pre-encoded GENLIB library, and the target hardware. First ABC is called to perform preprocessing and technology mapping, then the result is parsed

and placed on the selected hardware. The encoded problem is returned as an array of biases and couplings that is ready to be sent to the hardware using D-Wave API libraries. Listing 13 shows the usage documentation for the script.

```
from smtutils.process import Solver, get_msat_path
from smtutils.formula import SmtFormula, Symbol
from smtutils.parsing import SmtResponseParser

# Create a new formula
formula = SmtFormula()

# Declare variables
a = Symbol("Int", "a")
b = Symbol("Int", "b")

# Add assertions
formula.assert_(a+b == 5)
formula.assert_(a < 100)

# Check satisfiability, retrieve model for a and b
formula.check_sat()
formula.get_values(a, b)

# Print final SMT-lib formula to screen
print(str(formula))

# Run the solver
solver = Solver(get_msat_path("optimathsat"))
result =  solver.run_formula(f)

# Parse the SMT solver response
p = SmtResponseParser(res)
print(p.result, p.model)
```

Listing 4: Example usage of the smtutils library.

```
.model 74283.isc
.inputs W1 W2 W3 W4 W5 W6 W7 W8 W9
.outputs W32 W36 W37 W38 W39
.gate gate10 B=W9 C=W8 D=W7 A=n15
.gate gate10 B=n15 C=W6 D=W5 A=n16
.gate gate10 B=n16 C=W4 D=W3 A=n17
.gate gate10 B=n17 C=W2 D=W1 A=W32
.gate gate14 B=n17 C=W2 D=W1 A=W36
.gate gate14 B=n16 C=W4 D=W3 A=W37
.gate gate14 B=n15 C=W6 D=W5 A=W38
.gate gate14 B=W9 C=W8 D=W7 A=W39
.end
```

Listing 5: Circuit representation in BLIF format. Here gate are specified in a separate file (see Listing 8)

```
c comment lines start with a c
c 74283.aig
aag 41 9 0 5 32
c list of inputs
2
c [...]
18

c list of outputs
50
c [...]
82

c list of gates
20 5 3
22 4 2
24 9 7
c [...]
82 81 79
```

Listing 6: Circuit representation in ASCII AIGER format.Gates are indexed by number $2n$ while $2n + 1$ represent the negated nth gate.

```python
from pyrbc.aiger import parse_aig
from pyrbc.graph import build_graph
import networkx
import matplotlib.pyplot as plt

with open("example.aig", "rb") as f:
    outputs = parse_aig(f)

# Traverse the DAG, print all the nodes, avoid duplicates
visited = set()
for output in outputs:
    for node in output.iter_nodes(visited):
        print(node)

# Display the AIG using networkx
graph = build_graph(outputs)
networkx.draw(graph)
plt.show()
```

Listing 7: Example of usage for the pyrbc library.

```
#  GATE gate_name gate_area gate_formula;# json_penalty_function
#      PIN unused_pin_delay_information

GATE gate10 6.000 A = (C + D) * (B + D) * (B + C);# {"(bias ...
    PIN * INV 0 0 0 0 0 0

GATE gate14 8.000 A = (B + C + D) * (D + !B + !C) * (C + !B +...
    PIN * INV 0 0 0 0 0 0
```

Listing 8: Gate specification in GENLIB format, with extra data in JSON format.

```
from pfencoding.searchpf import search_pf_smallest
from pfencoding.utils import print_pf
from networkx import complete_bipartite_graph

graph = complete_bipartite_graph(4,4)
nx = 4
na = 4

def and3(x):
    return x[0] == (x[1] and x[2] and x[3])


model, pf  = search_pf_smallest(nx, na, graph, and3, gap=2)

print(model)
if model:
    print_pf(pf, model, and3)
```

Listing 9: Searching the smallest penalty function for the function $x_0 = (x_1 \wedge x_2 \wedge x_3)$ that fits inside a Chimera tile using `pfencoding`.

```
from glob import glob
from os.path import dirname
from gatecollector.encodeintiles import encode_aig,describe_aig
from gatecollector import database
from json import dumps

db = database.DelayedFunctionDatabase()
ngates = 100

for fn in  glob(dirname(__file__)+ "/dataset/*.aig"):
    #read all gates from file
    with open(fn, "rb") as f:
        db.read_function(f, 6)

# save to a file the most common gates
for i, gate   in enumerate(db.most_common(ngates)):
    (size, func, inputs), count = gate
    model = encode_aig(func)
    desc = describe_aig(func)
    if model:
        jsondata = {k: float(v) for k, v in model.items()}
        print ("""GATE gate{} {}.00 {};#{}
        PIN * INV 0 0 0 0 0 0
        """.format(i,
                model["ancilla_used"] + 1 + len(inputs),
                desc,
                dumps(jsondata)))
```

Listing 10: Example of extracting the most common gates out of a dataset using `gatecollector`.

```
UC Berkeley, ABC 1.01 (compiled Jan 27 2019 18:13:48)
abc 01> read ../datasets/74x/74283.isc.aig
abc 02> read ../datasets/generated.genlib
Entered genlib library with 34 gates from file "generated.genlib".
abc 02> source ../datasets/simplify.abc
74283.isc: i/o = 9/5  lat = 0  and = 32  lev =  9
abc 211> source ../datasets/convert.abc
[...]  area = 56.00 lev = 4
abc 212> write 74283.blif
```

Listing 11: Example of usage of ABC for technology mapping(edited for clarity).

```
usage: preencode.py [-h] [--output OUTPUT] [--numgates NUMGATES]
        [--cell {chimera,pegasus}]
        input_dir

Build a library of pre-encoded gates

positional arguments:
input_dir             directory containing circuits to be
                                                    analyzed

optional arguments:
-h, --help            show this help message and exit
--output OUTPUT, -o OUTPUT
            GENLIB output file
--numgates NUMGATES, -n NUMGATES
            Number of gates to get (default: all)
--cell {chimera,pegasus}, -c {chimera,pegasus}
            type of cell (default: chimera tile)
```

Listing 12: Usage documentation of `preencode.py`.

```
usage: encode.py [-h]
        [--hardware {chimera8,chimera12,chimera16,
                     pegasus6,pegasus8,pegasus12}]
        input library

Encode a SAT problem into a QA model.

positional arguments:
  input                 input SAT problem as AIG file
  library               pre-encoded library as GENLIB file

optional arguments:
  -h, --help            show this help message and exit
  --hardware {chimera8,chimera12,chimera16,
                pegasus6,pegasus8,pegasus12}
        -g   {chimera8,chimera12,chimera16,
                pegasus6,pegasus8,pegasus12}
                        QA hardware (default: chimera16)
```

Listing 13: Usage documentation of `encode.py`.

Chapter 9

Experimental evaluation

Using the software described in Chapter 8, the chapter provides a preliminary empirical validation of the proposed methods for SATtoIsing encoding and SAT solving by evaluating the performance of D-Wave's 2000Q quantum annealer in solving certain hard SAT problems (Section 9.1); a similar evaluation is performed on MaxSAT problems as well (Section 9.2).

Currently the most time-onerous step in the on-the-fly part of the process outlined in Chapter 7 is the placement and routing step. Encoding the problems in the following evaluation requires approximately 20 minutes on an Intel i7-5600U CPU. However software heuristics are heavily tunable in order to trade off efficiency and effectiveness of the place-and-route process. As long as the encoding process does not become untractable, we can focus on the time taken by the quantum annealer to reach the final solution state. Furthermore, in certain contexts (e.g. fault diagnosis [14]), hardware embeddings are reusable and therefore can be thought as a one-time cost.

This evaluation has a number of requirements. First, we require instances that can be entirely encoded in the qubits of a currently available quantum annealer, i.e. a Chimera graph of around 2000 qubits (although algorithms for solving larger CSP with QA have been proposed [13, 14]).

Furthermore, SAT solvers are already quite effective on average case problems, thus we need concrete problems that can get state-of-the-art solvers stuck. Another important consideration in solving [Max]SAT instances is that the QA hardware cannot reason on the input or produce proofs. Thus unsatisfiable SAT instances are not suitable for evaluation.

QA hardware behaves more like an SLS solver than a CDCL-based one: for this reason we solved the same problems with the state-of-the-art UBCSAT SLS SAT solver using the best performing algorithm, namely SAPS [92]. UBCSAT was run on a computer using a 8-core Intel® Xeon® E5-2407 CPU, at 2.20GHz.

The results reported in this section are not intended as a performance comparison between D-Wave's 2000Q system and UBCSAT, or any other classic computing tool. There are issues in comparing specialized vs. off-the-shelf hardware and different timing mechanisms and timing granularities. Rather than that the aim is to provide an experimental assessment of the potential use of quantum annealing for [Max]SAT solving.

Experimental data, problem files, translation files, demonstration source code and supplementary material can be accessed from a publicly available website.[1] A D-Wave 2000Q machine is publicly accessible through D-Wave's Leap cloud service.[2]

9.1 SAT Experiments

9.1.1 Choosing the benchmark problems

Due to previously mentioned limitations in size and connectivity of current QA systems, we require SAT problems which have a small number of variables but are hard for standard SAT solvers in general.

[1] `https://bitbucket.org/aqcsat/aqcsat`.

[2] `https://cloud.dwavesys.com/leap/`.

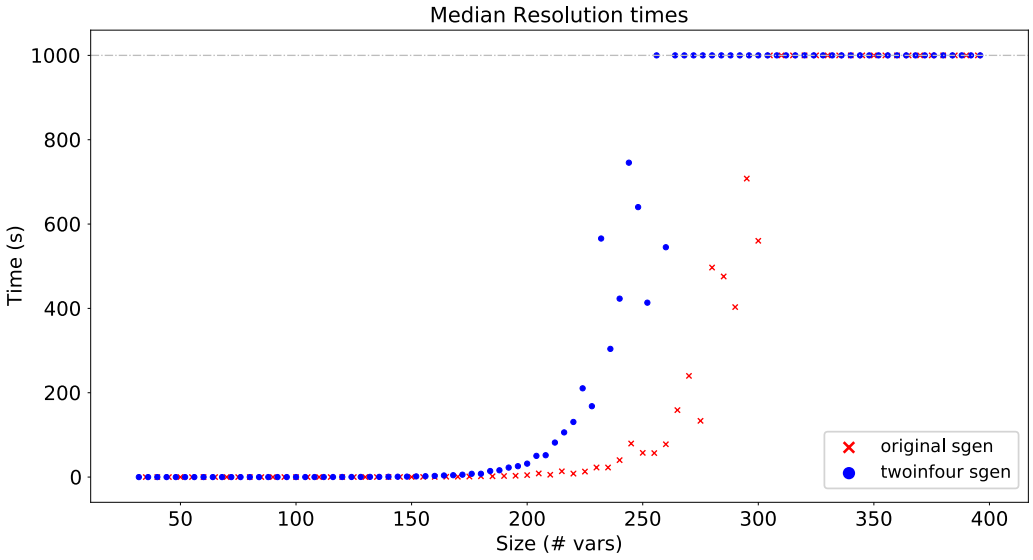


Figure 9.1: Median times vs. problem size for the best-performing SLS algorithm, on two variants of the SGEN problem on UBCSAT (SAPS). Timeout is at 1000 seconds. The figure report times on a 8-core Intel® Xeon® E5-2407 CPU, at 2.20GHz.

To this end the chosen benchmarks are created with the tool SGEN [88] with some modifications. SGEN is the current state of the art for generating the small unsolvable problems in recent SAT competitions. Furthermore, the problems have a structure that is suited for problem embedding, as they are composed of a single type of constraint that can be embedded very efficiently, and problems with few hundreds of variables are considerably hard. The SGEN family of generators received many improvements over the years, but satisfiable instances are generated in the same way [49, 89]. SGEN creates problems by setting cardinality constraints over different partitions of the set of variables. The tool requires as an input the desired problem size for the output (and a random number generator). Given this, the generator operates as follows:

1. A satisfying assignment is decided at random.

2. The tool partitions the variable set into sets of 5 elements in such a

way that each subset contains exactly one true variable for the desired solution.

3. For each subset we guarantee that at most one variable is true (10 2-CNF clauses).

4. The partition is shuffled into a new one. The tool ensures that each new subset contain exactly one true variable, and minimizes the similarity with the previous partition.

5. For each partition subset we ensure that at least one variable is true (a single CNF clause).

6. The previous two steps are repeated one more time, to further restrict the solution space.

In Figure 9.1, the red dots represent median resolution times for UBCSAT SAPS on random SGEN problems. Notice that with > 300 variables the solver reaches the timeout of 1000 seconds for all problems.

In our validation experiments, we modify the tool to better suit the annealer hardware. We use exactly-2-in-4 constraints on partitions with sets of size 4 instead of size 5 partitions, with exactly two true variables per subset. This constraint has a very efficient embedding and furthermore the modified problems are slightly harder with the same number of variables (see the blue dots on Figure 9.1, where UBCSAT SAPS reaches timeout with > 270 variables).

9.1.2 Experiments and Results

We generated several problem instances with multiple problem sizes. 100 different problems are generated per size, with size ranging from 32 variables to 80, the biggest size on which embedding has been successfully

performed. Furthermore problems have been generated in the original version (with at-least-one-in-five and at-most-one-in-five) and a two-in-four version (using exactly-two-in-four constraints). These SAT instances are encoded and embedded using the divide-and-conquer method.

For the experiment, a fixed number of samples/instance (5, 10, 20) is drawn from the quantum annealer. Annealing was executed at a rate of 10 µs per sample, for a total of 50 µs, [resp. 100 µs and 200 µs] of anneal time per instance respectively. Total time used by the D-Wave processor includes programming and readout; this amounts to about 150 µs per sample, plus a constant 10 ms of overhead. Table 9.1(a) shows the results using the D-Wave 2000Q annealer. The quantum annealer solves almost all problems with 5 samples (i.e. within 50 µs of total anneal time), and all of them are solved with 20 samples (i.e. within 200 µs of total anneal time). In order to contextualize the results, the same problems are solved with the UBCSAT SLS SAT solver, using SAPS [92]. These computations were performed using an 8-core Intel® Xeon® E5-2407 CPU, at 2.20GHz. Table 9.1(b) shows that the problems are nontrivial despite the small number of variables, and the run-times increase significantly with the size of the problem. (See also Figure 9.1.)

9.2 MaxSAT experiments

Exact penalty functions (Section 5.1) allow for the encoding of weighted MaxSAT problems, with some restrictions. To demonstrate the performance of the QA hardware in this regime, we generated weighted MaxSAT instances that have many distinct optimal solutions.

9.2.1 Choosing the benchmarks

The weighted MaxSAT problems were generated from the previous 2-in-4-SAT instances by removing part of the constraint, turning into unsatisfiable instances and then adding constraints on single variables with smaller weight. More precisely:

1. Start with the 2-in-4-SAT instances of the previous section.

2. Remove one of the partitions of the variable set, and change one 2-in-4 constraint to 1-in-4. This makes the SAT problem unsatisfiable: for an n variable problem, the first partition demands exactly $n/2$ true variables, while the second demands exactly $n/2 - 1$.

3. Each constraint is assigned a soft weight of 3 and for each variable single literal constraint with random polarity is created and assigned weight 1.

4. Multiple MaxSAT instances of this form are generated until an instance has the optimal solution with exactly one violated clause of weight 3 and at least $n/3$ violated clauses of weight 1, and with at least 200 distinct optimal solutions.

9.2.2 Experiments and Results

The problems are encoded with the same divide-and-conquer method, using exact penalty functions. As discussed, finding analytically the smallest approprate chain gap for encoding is unfeasible. Chain gaps that are too small result in a large number of broken chains, while high gaps reduce excessively the relative constraint below the noise level. For this experiment the optimal gaps have been found experimentally by sweeping over a range of values and choosing the best. The chosen chain gap was always in the

range $g_{chain} \in [2, 6]$, relative to normal exact penalty functions (Section 5.1).

The D-Wave processor is used to generate a single optimal MaxSAT solution and Table 9.2 summarizes the results. Annealing was executed at a rate of 10 µs per sample, for a total of 1 ms of anneal time per instance. Again, the run-times for various high-performing SLS MaxSAT solvers are added. Classical computations were performed on an Intel i7 2.90GHz × 4 processor. The solvers gw2sat, rots, and novelty are as implemented in UBCSAT [92]. The QA hardware solves almost all problems with 100 samples/instance (i.e. within 1 ms of anneal time).

Table 9.3 considers instead the performance in generating distinct optimal solutions. For each solver and problem size, the table indicates the number of distinct solutions found in 1 second, averaged across 100 problem instances of that size. For the smallest problems, 1 second is sufficient for all solvers to generate all solutions, while the diversity of solutions found varies widely as problem size increases. The D-Wave processor returns less optimal solutions for MaxSAT instances compared to the SAT instances, but it is still effective in providing distinct optimal solutions due to the rapid sampling rate.

9.3 SGEN Problems on Pegasus

All the previous experiments have been performed on the currently available hardware, that uses the Chimera topology. Whereas it is not yet possible to run the same experiments on the improved Pegasus topology, we can analyze the impact of the new architecture by checking the maximum size that would fit for the same problem class.

Table 9.4 shows the maximum problem size that can be encoded for Pegasus chips of different sizes. For reference the previous experiment

were run on a 16×16 Chimera grid with 2048 qubits. We can notice that the new architecture allows to encode bigger problem with fewer qubits, in particular a 6×6 Pegasus hardware with 720 qubits can contain problems of size roughly equal to the D-Wave 2000Q hardware. Furthermore, larger future Pegasus chips 12×12 and 16×16 can hold problems that require hundreds of seconds and more to be solved by UBCSAT (Figure 9.1).

(a)

D-Wave 2000Q				
Problem size	**# solved 5 samples**	**# solved 10 samples**	**# solved 20 samples**	**% optimal samples**
32 vars	100	100	100	97.4
36 vars	100	100	100	96.4
40 vars	100	100	100	94.8
44 vars	100	100	100	93.8
48 vars	100	100	100	91.4
52 vars	100	100	100	93.4
56 vars	100	100	100	91.4
60 vars	100	100	100	88.2
64 vars	100	100	100	84.6
68 vars	100	100	100	84.4
72 vars	98	100	100	84.6
76 vars	99	99	100	86.6
80 vars	100	100	100	86.0

(b)

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

Table 9.1: (a) Number of SATtoIsing problem instances (out of 100) solved by the QA hardware using 5 samples [resp. 10 and 20] and average fraction of samples from the QA hardware that are optimal solutions.
(b) Run-times in ms for SAT instances solved by UBCSAT using SAPS, averaged over 100 instances of each problem size.

(a)

D-Wave 2000Q		
Problem size	**# solved**	**% optimal samples**
32 vars	100	78.7
36 vars	100	69.0
40 vars	100	60.2
44 vars	100	49.9
48 vars	100	40.4
52 vars	100	35.2
56 vars	100	24.3
60 vars	100	22.3
64 vars	99	17.6
68 vars	99	13.0
72 vars	98	9.6
76 vars	94	6.6
80 vars	93	4.3

(b)

MaxSAT solvers: avg time (ms)				
Problem size	**g2wsat**	**rots**	**maxwalksat**	**novelty**
32 vars	0.020	0.018	0.034	0.039
36 vars	0.025	0.022	0.043	0.060
40 vars	0.039	0.029	0.056	0.119
44 vars	0.049	0.043	0.070	0.187
48 vars	0.069	0.054	0.093	0.311
52 vars	0.122	0.075	0.115	0.687
56 vars	0.181	0.112	0.156	1.319
60 vars	0.261	0.130	0.167	1.884
64 vars	0.527	0.159	0.207	4.272
68 vars	0.652	0.210	0.270	8.739
72 vars	0.838	0.287	0.312	14.118
76 vars	1.223	0.382	0.396	18.916
80 vars	1.426	0.485	0.430	95.057

Table 9.2: (a) Number of MaxSATtoIsing problem instances (out of 100) solved by the QA hardware and average fraction of samples that are optimal.
(b) Average time in ms taken to find an optimal solution by various inexact weighted MaxSAT solvers.

(a)

D-Wave 2000Q		
Size	**wall-clock**	**anneal only**
32 vars	448.5	443.9
36 vars	607.0	579.9
40 vars	1007.9	922.0
44 vars	1322.6	1066.6
48 vars	1555.4	1111.8
52 vars	3229.0	1512.5
56 vars	2418.9	1147.4
60 vars	4015.3	1359.3
64 vars	6692.6	1339.1
68 vars	6504.2	1097.1
72 vars	3707.6	731.7
76 vars	2490.3	474.2
80 vars	1439.4	332.7

(b)

MaxSAT solvers				
Size	**g2wsat**	**rots**	**maxwalksat**	**novelty**
32 vars	448.5	448.5	448.5	448.5
36 vars	607.0	606.9	606.9	606.8
40 vars	1007.7	1006.3	1005.3	1005.0
44 vars	1313.8	1307.1	1311.7	1255.5
48 vars	1515.4	1510.7	1504.9	1320.5
52 vars	2707.5	2813.0	2854.6	1616.2
56 vars	2021.9	2106.2	2186.6	969.8
60 vars	2845.6	3061.7	3289.0	904.4
64 vars	3100.0	4171.0	4770.0	570.6
68 vars	2742.2	3823.3	4592.4	354.8
72 vars	1841.1	2400.2	2943.4	212.6
76 vars	1262.5	1716.0	2059.2	116.4
80 vars	772.2	1111.1	1363.9	66.7

Table 9.3: Distinct optimal solutions found in 1 second by various MaxSAT solvers, averaged across 100 instances. "anneal only" accounts for only the 10 µs per sample anneal, while "wall-clock" accounts for the full time, including programming and readout. (b) Classical computations were performed as in Table 9.2(b).

Pegasus	# qubits	SGEN size	# constrs
4x4	288	44	33
6x6	720	88	66
8x8	1344	128	96
12x12	3168	212	159
16x16	5760	320	240

Table 9.4: Maximum size of encoded SGEN problems on Pegasus topologies. In comparison, on a Chimera 16x16 having 2048 qubits, the maximum SGEN size is 80.

Chapter 10

Conclusions

In this thesis a method for encoding SAT and MaxSAT problems into Quantum Annealing problems has been presented. It employs several new techniques to improve effectiveness, and it shows promise in encoding problems that challenge state-of-the-art SAT solvers.

The described approach has still some limitations. First, the online phase is currently quite slow in producing an encoding, taking several minutes to hours. This is due to the necessity of searching a very efficient encoding for problems that are as big as possible. SMT optimal encoding of Boolean function becomes harder on the newer hardware topology, while the pre-encoded library approach benefits from having pre-encoded gates that are as big as possible. Finally, currently either the SAT problem encoding result fits into the hardware or the process fails, thus still showing a certain lack of flexibility.

These limitations provides cues for further research. Progresses in technology mapping and placement and routing could directly provide more effective encodings. It is also interesting to consider alternative approaches in SMT solving for pre-encoding. Finally, a problem decomposition approach tuned to SAT could increase flexibility by allowing encoding of large problems.

Bibliography

[1] SAT competitions. `http://satcompetition.org/`. Accessed: 2017-04-04.

[2] Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Faster parameterized algorithms for minor containment. In *Proceedings of the 12th Scandinavian Conference on Algorithm Theory*, SWAT'10, pages 322–333, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Tameem Albash and Daniel A Lidar. Adiabatic quantum computing. *arXiv preprint arXiv:1611.04471*, 2016.

[4] M. H. Amin. Searching for quantum speedup in quasistatic quantum annealers. *Physical Review A*, 92(5):052323, November 2015.

[5] Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum Boltzmann machine. *Physical Review X*, 8:021050, May 2018.

[6] Evgeny Andriyash, Zhengbing Bian, Fabian Chudak, Marshall Drew-Brook, Andrew D. King, William G. Macready, and Aidan Roy. Boosting integer factoring performance via quantum annealing offsets. Technical report, 2017.

[7] Francisco Barahona. On the computational complexity of Ising spin

glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[8] Clark Barrett, Silvio Ranise, Aaron Stump, and Cesare Tinelli. The satisfiability modulo theories library (SMT-LIB). `www.SMT-LIB.org`, 2008.

[9] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*. 2009.

[10] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[11] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum Cryptography*. 2018.

[12] Vaughn Betz and Jonathan Rose. VPR: A new packing, placement and routing tool for FPGA research. In *International Workshop on Field Programmable Logic and Applications*, pages 213–222. Springer, 1997.

[13] Zhengbing Bian, Fabian Chudak, Robert Israel, Brad Lackey, William G. Macready, and Aidan Roy. Discrete optimization using quantum annealing on sparse Ising models. *Frontiers in Physics*, 2:56, 2014.

[14] Zhengbing Bian, Fabian Chudak, Robert Brian Israel, Brad Lackey, William G. Macready, and Aidan Roy. Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. *Frontiers in ICT*, 2016.

[15] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and MaxSAT with a quantum annealer: Foundations and a preliminary report. In *The 11th International Symposium on Frontiers of Combining Systems, FroCoS'17*, volume 10483 of *LNCS*. Springer, 2017.

[16] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and MaxSAT with a quantum annealer: Foundations and a preliminary report. 2017. Extended version. `https://bitbucket.org/aqcsat/frocos2017/raw/HEAD/sat2ising_extended.pdf`.

[17] Zhengbing Bian, Fabian Chudak, William G. Macready, Lane Clark, and Frank Gaitan. Experimental determination of Ramsey numbers. *Physical Review Letters*, 111:130505, September 2013.

[18] Zhengbing Bian, Fabián A. Chudak, William G. Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and MaxSAT with a quantum annealer: Foundations, encodings, and preliminary results. 2018. `https://arxiv.org/abs/1811.02524` . Under submission for journal publication.

[19] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, February 2009.

[20] Tomas Boothby, Andrew D. King, and Aidan Roy. Fast clique minor generation in chimera qubit connectivity graphs. *Quantum Information Processing*, 15(1):495–508, January 2016.

[21] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pages 24–40. Springer, 2010.

[22] Robert King Brayton. *BLIF-MV: An interchange format for design verification and synthesis.* Electronics Research Laboratory, College of Engineering, University of California, 1991.

[23] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 583–592, 2010.

[24] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1):6:1–6:33, February 2013.

[25] Jun Cai, William G Macready, and Aidan Roy. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741*, 2014.

[26] Matthew Campagna, Lidong Chen, Özgür Dagdelen, Jintai Ding, Jennifer K Fernick, Nicolas Gisin, Donald Hayford, Thomas Jennewein, Norbert Lütkenhaus, Michele Mosca, et al. Quantum safe cryptography and security. *ETSI White Paper*, 8, 2015.

[27] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, September 2017.

[28] T. F. Chan, J. Cong, Tianming Kong, and J. R. Shinnerl. Multilevel optimization for large-scale circuit placement. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD - 2000. IEEE/ACM Digest of Technical Papers (Cat. No.00CH37140)*, pages 171–176, November 2000.

[29] Nicholas Chancellor, Stefan Zohren, Paul A. Warburton, Simon C. Benjamin, and Stephen Roberts. A direct mapping of max k-SAT and high order parity checks to a Chimera graph. *Scientific reports.*, 6:37107, November 2016.

[30] Y. J. Chang, Y. T. Lee, and T. C. Wang. NTHU-Route 2.0: A fast and stable global router. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 338–343, November 2008.

[31] H. Chen, C. Hsu, and Y. Chang. High-performance global routing with fast overflow reduction. In *2009 Asia and South Pacific Design Automation Conference*, pages 582–587, January 2009.

[32] Minsik Cho, Katrina Lu, Kun Yuan, and D. Z. Pan. Boxrouter 2.0: architecture and implementation of a hybrid and robust global router. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 503–508, Nov 2007.

[33] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, October 2008.

[34] Vicky Choi. Different adiabatic quantum optimization algorithms for the NP-complete exact cover and 3SAT problems. *Quantum Information and Computation*, 11(7-8):638–648, July 2011.

[35] Vicky Choi. Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design. *Quantum Information Processing*, 10(3):343–353, June 2011.

[36] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT 5 SMT solver. In *Tools and Al-*

gorithms for the Construction and Analysis of Systems, TACAS'13., volume 7795 of *LNCS*, pages 95–109. Springer, 2013.

[37] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[38] Vinícius P Correia and André I Reis. Classifying n-input Boolean functions, 2001.

[39] R. Dechter. *Bucket Elimination: A Unifying Framework for Probabilistic Inference*, pages 75–104. Springer Netherlands, Dordrecht, 1998.

[40] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[41] Adam Douglass, Andrew D. King, and Jack Raymond. *Constructing SAT Filters with a Quantum Annealer*, pages 104–120. Springer International Publishing, Cham, 2015.

[42] Raouf Dridi and Hedayat Alghassi. Prime factorization using quantum annealing and computational algebraic geometry. *Scientific Reports*, 7(1), February 2017.

[43] D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2004.

[44] Niklas Een, Alan Mishchenko, and Niklas Sörensson. Applying logic synthesis for speeding up SAT. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing –*

SAT 2007, pages 272–286, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[45] Edward Farhi, David Gosset, Itay Hen, A. W. Sandvik, Peter Shor, A. P. Young, and Francesco Zamponi. Performance of the quantum adiabatic algorithm on random instances of two optimization problems on regular hypergraphs. *Physical Review A*, 86:052334, Nov 2012.

[46] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, September 2009.

[47] Austin G. Fowler, Ashley M. Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A*, 80:052312, November 2009.

[48] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[49] Allen Van Gelder and Ivor Spence. Zero-one designs produce small hard SAT instances. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing – SAT 2010*, Lecture Notes in Computer Science, pages 388–397. Springer Berlin Heidelberg.

[50] Michael Gester, Dirk Müller, Tim Nieberg, Christian Panten, Christian Schulte, and Jens Vygen. BonnRoute: Algorithms and data structures for fast and good VLSI routing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):32, 2013.

[51] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[52] Itay Hen and A. P. Young. Exponential complexity of the quantum adiabatic algorithm for certain satisfiability problems. *Physical Review E*, 84:061152, Dec 2011.

[53] Zheng Huang, Lingli Wang, Yakov Nasikovskiy, and Alan Mishchenko. Fast boolean matching based on NPN classification. In *International Conference on Field-Programmable Technology (FPT)*, 2013.

[54] Swen Jacobs. Extended AIGER format for synthesis. *arXiv preprint arXiv:1405.5793*, 2014.

[55] S. Jiang, K. A. Britt, A. J. McCaskey, T. S. Humble, and S. Kais. Quantum annealing for prime factorization. *arXiv preprint*, April 2018.

[56] Stephen Jordan. Quantum algorithm zoo. `https://quantumalgorithmzoo.org/`. Accessed: 2018-12-27.

[57] Andrew D King, Juan Carrasquilla, Jack Raymond, Isil Ozfidan, Evgeny Andriyash, Andrew Berkley, Mauricio Reis, Trevor Lanting, Richard Harris, Fabio Altomare, et al. Observation of topological phenomena in a programmable lattice of 1,800 qubits. *Nature*, 560(7719):456, 2018.

[58] Andrew D King, Trevor Lanting, and Richard Harris. Performance of a quantum annealer on range-limited constraint satisfaction problems. *arXiv preprint*, 2015.

[59] James King, Sheir Yarkoni, Mayssam M Nevisi, Jeremy P Hilton, and Catherine C McGeoch. Benchmarking a quantum annealing processor with the time-to-target metric. *arXiv preprint arXiv:1508.05087*, 2015.

[60] P. Klein and R. Ravi. A nearly best-possible approximation algorithm

for node-weighted Steiner trees. *Journal of Algorithms*, 19(1):104 – 115, 1995.

[61] T. Lanting, R. Harris, J. Johansson, M. H. S. Amin, A. J. Berkley, S. Gildert, M. W. Johnson, P. Bunyk, E. Tolkacheva, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, E. M. Chapple, C. Enderud, C. Rich, B. Wilson, M. C. Thom, S. Uchaikin, and G. Rose. Cotunneling in pairs of coupled flux qubits. *Physical Review B*, 82(6):060512, 2010.

[62] Chu Min Li and Felip Manyà. *MaxSAT, Hard and Soft Constraints*, chapter 19, pages 613–631. In Biere et al. [19], February 2009.

[63] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014.

[64] Stephen M. Majercik. *Stochastic Boolean Satisfiability*, chapter 27, pages 887–925. In Biere et al. [19], February 2009.

[65] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. In Biere et al. [19], February 2009.

[66] Catherine C. McGeoch and Cong Wang. Experimental evaluation of an adiabatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, pages 23:1–23:11, New York, NY, USA, 2013. ACM.

[67] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014.

[68] A. Mishchenko, S. Chatterjee, and R. K. Brayton. Improvements to technology mapping for LUT-based FPGAs. *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, 26(2):240–253, Feb 2007.

[69] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, pages 532–535, New York, NY, USA, 2006. ACM.

[70] Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, Xinning Wang, and Timothy Kam. Technology mapping with Boolean matching, supergates and choices. 2005.

[71] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, and Robert K Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, 2005.

[72] Dirk Müller, Klaus Radke, and Jens Vygen. Faster min–max resource sharing in theory and practice. *Mathematical Programming Computation*, 3(1):1–35, 2011.

[73] Florian Neukart, Gabriele Compostella, Christian Seidel, David von Dollen, Sheir Yarkoni, and Bob Parney. Traffic flow optimization using a quantum annealer. *Frontiers in ICT*, 4:29, 2017.

[74] Michael A. Nielsen and Isaac Chuang. Quantum computation and quantum information. *American Journal of Physics*, 70(5):558–559, 2002.

[75] Bryan O'Gorman, Eleanor Gilbert Rieffel, Minh Do, Davide Venturelli, and Jeremy Frank. Comparing planning problem compilation approaches for quantum annealing. *The Knowledge Engineering Review*, 31(5):465–474, 2016.

[76] Scott Pakin. A quantum macro assembler. In *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*, pages 1–8, 2016.

[77] Alejandro Perdomo-Ortiz, Joseph Fluegemann, Sriram Narasimhan, Rupak Biswas, and Vadim N Smelyanskiy. A quantum annealing approach for fault detection and diagnosis of graph-based systems. *The European Physical Journal Special Topics*, 224(1):131–148, 2015.

[78] K. L. Pudenz, G. S. Tallant, T. R. Belote, and S. H. Adachi. Quantum annealing and the satisfiability problem. *ArXiv e-prints*, December 2016.

[79] Eleanor G. Rieffel, Davide Venturelli, Bryan O'Gorman, Minh B. Do, Elicia M. Prystay, and Vadim N. Smelyanskiy. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing*, 14(1):1–36, 2015.

[80] Gili Rosenberg, Poya Haghnegahdar, Phil Goddard, Peter Carr, Kesheng Wu, and Marcos López de Prado. Solving the optimal trading trajectory problem using a quantum annealer. In *Proceedings of the 8th Workshop on High Performance Computational Finance*, WHPCF '15, pages 7:1–7:7, New York, NY, USA, 2015. ACM.

[81] J. A. Roy and I. L. Markov. High-performance routing at the nanometer scale. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1066–1077, June 2008.

[82] Jarrod A. Roy, David A. Papa, Saurabh N. Adya, Hayward H. Chan, Aaron N. Ng, James F. Lu, and Igor L. Markov. Capo: Robust and scalable open-source min-cut floorplacer. In *Proceedings of the 2005 International Symposium on Physical Design*, ISPD '05, pages 224–226, New York, NY, USA, 2005. ACM.

[83] Siddhartha Santra, Gregory Quiroz, Greg Ver Steeg, and Daniel A Lidar. Max 2-SAT with up to 108 qubits. *New Journal of Physics*, 16(4), 2014.

[84] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Transaction on Computational Logics – TOCL*, 2015.

[85] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A tool for optimization modulo theories. In *International conference on computer aided verification*, 2015.

[86] Franco Selleri. *Quantum mechanics versus local realism: the Einstein-Podolsky-Rosen paradox*. Springer Science & Business Media, 2013.

[87] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. Ieee, 1994.

[88] Ivor Spence. Sgen1: A generator of small but difficult satisfiability benchmarks. *Journal of Experimental Algorithmics (JEA)*, 2010.

[89] Ivor Spence. Weakening cardinality constraints creates harder satisfiability benchmarks. *Journal of Experimental Algorithmics (JEA)*, 2015.

[90] J. Su, T. Tu, and L. He. A quantum annealing approach for boolean satisfiability problem. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.

[91] Wern-Jieh Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):349–359, March 1995.

[92] Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *SAT*, 2004.

[93] Immanuel Trummer and Christoph Koch. Multiple query optimization on the D-Wave 2X adiabatic quantum computer. *Proceedings of the VLDB Endowment*, 9(9):648–659, May 2016.

[94] G.S. Tseitin. *On the complexity of derivations in the propositional calculus*, page 115–125. Consultants Bureau, 1968. Part II.

[95] D. Venturelli, D. J. J. Marchand, and G. Rojo. Quantum annealing implementation of job-shop scheduling. *arXiv preprint*, 2015.

[96] Yue Xu, Yanheng Zhang, and Chris Chu. Fastroute 4.0: Global router with efficient via minimization. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ASP-DAC '09, pages 576–581. IEEE Press, 2009.

[97] Arman Zaribafiyan, Dominic J. J. Marchand, and Seyed Saeed Changiz Rezaei. Systematic and deterministic graph minor embedding for Cartesian products of graphs. *Quantum Information Processing*, 16(5):136, April 2017.

[98] Kenneth M. Zick, Omar Shehab, and M. French. Experimental quantum annealing: Case study involving the graph isomorphism problem. *Scientific Reports 5, 11168*, 2015.