Adiabatic Quantum Computing with QUBO Formulations

by

Richard Hua

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
in
Computer Science

University of Auckland

This thesis is dedicated to Heather, my dearest friend.
I would not be here without you and I will miss you everyday.

Rest in peace now.

ABSTRACT

We study two types of problems in this thesis, graph covering problems including the Dominating Set and Edge Cover which are classic combinatorial problems and the Graph Isomorphism Problem with several of its variations. For each of the problems, we provide efficient quadratic unconstrained binary optimization (QUBO) formulations suitable for adiabatic quantum computers, which are viewed as a real-world enhanced model of simulated annealing.

The number of qubits (dimension of QUBO matrices) required to solve the graph covering problems are $O(n + n \lg n)$ and $O(m + n \lg n)$ respectively, where n is the number of vertices and m is the number of edges. We also extend our formulations for the Minimum Vertex-Weighted Dominating Set problem and Minimum Edge-Weighted Edge Cover problem.

For the Graph Isomorphism Problem, we provide two QUBO formulation through two approaches both requiring $O(n^2)$ variables. We also provide several different formulations for two extensions of the Graph Isomorphism Problems each requiring a different number of variables ranging from $O(n_1 n_2)$ to $O((n_1 + 1)n_2)$.

We also provide some experimental results using a D-Wave 2X quantum computer with 1098 active qubit-coupled processors on the problems studied here for a selection of known common graphs.

ACKNOWLEDGMENTS

I wish to thank Dr Michael Dinneen for his guidance and help throughout this project. I would also like to thank Professor Cristian Calude for his useful comment and advices in preparing this thesis.

Special thanks go to my family and friends. Thank you for all your support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With development of Big Data and the resurgence of Deep Learning in recent years, there is also a surge in related technologies and applications being developed and deployed in practice [31, 42, 50]. From the well known four V's of Big data (Volume, Variety, Velocity and Veracity, see [16]) to the need of building Artificial Neural Networks of increasing scales for more complex problems, one thing that naturally comes with the emergence of such technologies in the industry is the demand for more computational power.

The famous Moore's Law states that the number of transistors in a integrated circuit doubles in about every 18 months. The number of transistors in a chip is often used as the measurement of computational power of modern computers. Even tho Moore's Law has successfully predicted the growth of the field for over 40 years now [56], there are experts who are concerned about the growth of the industry in the foreseeable future [54, 51, 8]. Even Gorden Moore himself has stated that he would put the 'life expectancy' of his prediction at around the year 2020 to 2030 (see [19]).

In recent years, different institutes and organizations have proposed a variety of specialized hardwares specifically designed for certain problems, or have been leveraging existing product to comprehend the need. For example, the Xeon Phi processor family from Intel is specifically designed to address some of the common issues of Deep Learning algorithms with standard chipsets and researches have shown that the this particular design indeed do outperform others on specific problems [32, 53].

Another approach to address these issues is to change the fundamental computation model. The concept of Quantum Computing has been constantly evolving even since its introduction back in the 1980s [23]. Many different models of Quantum Computing exists, the most well-known one is probably the *Quantum Circuit* model which uses interconnected quantum gates as building blocks for quantum circuits that can be used to solve various problems [39]. Research conducted on this model shows much promise. For instance, the famous Grover's algorithm can search for a particular item in a collection of randomly ordered items in time $O(\sqrt{n})$ which can not be done with conventional computers [26]. And Shor's integer factorization algorithm takes only polynomial time on a quantum circuit computer whereas the exact time complexity of the problem remains unknown with classic algorithms [49]. Despite its successful theoretical development in the last 30 years, there is one major obstacle preventing this model from becoming relevant in practice, the difficulty in engineering one of practical scale. Currently, the largest instance of integer factorization (in terms of the integer being factorized) using Shor's algorithm on actual quantum circuits is only 21 (see [38]). Since the quantum circuit gate model is not the focus of this thesis, we will not formally introduce it here.

Adiabatic Quantum Computing (AQC) takes on a different model of computation. AQC is based on the process of evolving a ground state of a Hamiltonian representing a problem to a minimum-energy solution state [22, 21]. It has been shown to be equivalent to the more traditional quantum circuit model (only in terms of computability, not efficiency see [2]). Other introductory details about the application of AQC may be found in [47, 9]. Even though AQC can only simulate quantum circuit algorithms with polynomial overhead, it has attracted a lot of attention recently. The advantage of AQC is that a particular type of physical device that can be used for AQC known as *quantum annealer* is relatively easy to build.

D-Wave computers are produced by the Canadian company D-Wave Systems that use quantum annealing as a computation method. D-Wave One (2011) operates on a 128-qubit chipset; D-Wave Two (2013) works with 512 qubits. The latest and most advanced machine is D-Wave 2X$^{\text{TM}}$ System [15]. The current family of D-Wave computers can solve problems formulated in either *Ising* form or *Quadratic Unconstrained Binary Optimization* (QUBO) form, defined later. The paper by Lucas [37] provides a good foundation of Ising/QUBO formulations of many hard combinatorial problems. Some of these initial formulations have recently be improved by several authors, motivated by the limitations on the number of actual available qubits in existing hardwares. D-Wave qubits are loops of superconducting wire, the coupling between qubits is magnetic wiring and the machine itself is supercooled. See more in [39, 9]. The latest model, D-Wave 2X$^{\text{TM}}$, has 1100 qubits chilled close to absolute zero to get quantum effects [15].

The computer architecture consists of qubits arranged with a host configuration known as *Chimera graphs* which consists of an $M \times N$ two-dimensional lattice of blocks, with each block consisting of $2L$ vertices (a complete bipartite graph $K_{L,L}$), in total $2MNL$ vertices. A more formal definition of the family of Chimera graphs can be found in [10, 12], the Python script in Appendix H can also be used to generate standard adjacency list format of the Chimera graph for any M, N and L. The current D-Wave API (Application Programming Interface) provides the function of querying the quantum machine to solve problems formulated either in the Ising or QUBO (logical) form [12]. A formal definition of QUBO and Ising will be given in Chapter 2.

This thesis will not go into details of the physical nature of the actual computation. We will try to present as less physics knowledge as possible and only provide a high level mathematical interpretation whenever we can, so that the premise of this thesis remains simple: assuming we have this quantum oracle that can provide consistent solutions to the QUBO problem, what practical purposes can we use it for? So the goal here is to formulate or reduce hard graph theory problems in their QUBO form so we can solve them using a quantum annealer.

The main chapters of this thesis are organized as follows:

- Chapter 2 provides the necessary mathematics background knowledge as well as an overview on how the D-Wave system fits into the current framework.
- Chapter 3 provides efficient QUBO formulations of several graph covering problems including their proofs of correctness. This chapter is partially attained from [18], a conference version of it has been submitted.
- Chapter 4 provides several QUBO formulations of the Graph Isomorphism Problem and also studies several variations of the problem. This chapter is partially attained from [11].

Some relevant experimental results on a D-Wave 2X machine are also provided in Chapter 3 and 4.

Chapter 2

Definitions and Background

This chapter provides the basic definitions and background knowledge for graph theory and other mathematical notations necessary for the rest of this thesis. Please note that due to the specialty of the problems studied here, some of the definitions we provide here may differ slightly from a more standard source such as [28] or [29].

2.1 Mathematical prerequisite

Note that within the scope of this thesis, all logarithms denoted by lg or log are of base 2 unless specified otherwise.

Definition 1. *The **cardinality** of a set X, denoted by $|X|$, is the number of elements in X.*

Definition 2. *Given a set X, the **power set** of X, denoted by 2^X is the set containing all subsets of X.*

Definition 3. *The **Cartesian product** of two sets A and B, denoted by $A \times B$ is the set $\{(a, b) \mid a \in A \text{ and } b \in B\}$. The Cartesian product of a set A with it self is denoted by A^2.*

Definition 4. *The **integer set** $\mathbb{Z}_n$ is the set of integers $\{0, 1, \cdots, n-1\}$.*

Definition 5. *A **simple undirected graph** $G = (V, E)$ consists a finite non-empty set V of **vertices** together with a set E of **edges**, which are unordered 2-element subsets of V. The vertex set of a graph G is denoted by $V(G)$ and similarly the edge set is denoted by $E(G)$.*

The problems studied in this thesis focus on simple undirected graphs, and therefore the term graph will always be referring to a simple undirected graph unless specified otherwise.

Definition 6. *Let $G = (V, E)$ be a graph, the number of vertices of G, $|V|$ is called the* ***order*** *of G. And the number of edges, $|E|$ is called the* ***size*** *of G. The order and size of G are typically denoted by n and m respectively.*

For convention, we normally label the n vertices of a graph G by

$$V = \{0, 1, 2, \cdots, n-1\}$$

or $V = \{v_0, v_1, v_2, \cdots, v_{n-1}\}$. An edge could be denoted by a 2-set as $\{i, j\}$ or $\{v_i, v_j\}$ (or sometimes simply as ij or v_iv_j for convenience). Please note that by Definition 5, an edge is considered as a set hence we have $\{i, j\} = \{j, i\}$ for all $i, j \in V$. Since it is generally assumed that a set does not contain duplicated terms, at most one of the two terms $\{i, j\}$, $\{j, i\}$ can be in E for all $i, j \in V$. Another property about simple undirected graphs is that for all $v \in V$, $\{v, v\} \notin E$.

Definition 7. *Consider a graph $G = (V, E)$ and $\{u, v\}$ an edge in E. We say in this case that the two vertices u and v are* ***adjacent*** *to each other (or u and v are neighbors of each other). And the edge $\{u, v\}$ is said to* ***incident*** *to the vertices u and v. Similarly, we say that u, v are* ***non-adjacent*** *if $\{u, v\} \notin E$.*

Definition 8. *Let $G = (V, E)$ be a graph and v be a vertex in V. The* ***neighborhood*** *of v, denoted by $N(v)$, is a subset of V consists of all vertices adjacent to v. The set of edges incident to u is denoted by $I(v)$.*

Definition 9. *In a graph $G = (V, E)$, the* ***degree*** *of a vertex $v \in V$, denoted by $\Delta(v)$, is the number of vertices adjacent to v.*

Definition 10. *In a graph $G = (V, E)$, a* ***walk*** *is a sequence of vertices $v_0v_1 \cdots v_n$ where $v_i \in V$ for all $0 \leq i \leq n$ and $\{v_i, v_{i+1}\} \in E$ for all $0 \leq i < n$. A* ***path*** *is a walk in which all the vertices in the sequence are distinct. Furthermore, a* ***cycle*** *is a path consists at least 3 different vertices except that $v_0 = v_n$.*

Definition 11. *A graph $G = (V, E)$ is* ***connected*** *if for all $u, v \in V$, there is a path connecting u and v.*

Definition 12. *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. We say that G_1 and G_2 are* ***isomorphic*** *if there exists an* **edge in-variant** *bijective function $f : V_1 \to V_2$ such that $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$.*

Definition 13. *Let $G = (V, E)$ be a graph and $\{u, v\}$ an edge in E. The* ***contraction*** *of $\{u, v\}$ results in a new graph $G' = (V', E')$ such that $V' = V \setminus \{u, v\} \cup \{w\}$ and*

$$E' = E \backslash (\{\{u, x\} \mid x \in N(u)\} \cup \{\{v, x\} \mid x \in N(v)\}) \cup \{\{w, x\} \mid x \in (N(u) \cup N(v)) \backslash \{u, v\}\}.$$

Definition 14. *Let $G = (V, E)$ be a graph. A* ***graph minor*** *of G is a graph obtained by repeatedly deleting vertices and edges or contracting edges of G.*

Definition 15. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. A* ***minor embedding*** *of G_1 onto G_2 is a function $f : V_1 \to 2^{V_2}$ such that:*

1. *For all $v \in V_1$, the set of vertices v maps to under f are disjoint.*
2. *For all $v \in V_1$, there is a subset of edges $E' \in E_2$ such that $G' = (f(v), E')$ is connected.*
3. *If $\{u, v\} \in E_1$, then there exist $u', v' \in V_2$ such that $u' \in f(u)$, $v' \in f(v)$ and $\{u', v'\}$ is an edge in E_2.*

Within the scope of a minor embedding, the graphs G_1 and G_2 are referred to as the guest and host graph respectively.

Definition 16. *The family of* ***complete graphs*** *is defined as follows: A complete graph with order n, denoted by $K_n = (V, E)$ has $V = \{0, 1, \cdots, n-1\}$. And for all pairs of vertices i and j, $\{i, j\} \in E$.*

Definition 17. *A* ***complete bipartite graph*** *with $n_1 + n_2$ vertices, denoted by K_{n_1,n_2}, has $V = V_1 \cup V_2$ where $V_1 \cap V_2 = \emptyset$ and $|V_1| = n_1$, $|V_2| = n_2$. For every pair of vertices $\{i, j\}$ where $i \in V_1$ and $j \in V_2$, $\{i, j\} \in E$.*

We will introduce other concepts and notations when we need them in later chapters as well.

2.2 Quadratic Unconstrained Binary Optimization

Quadratic Unconstrained Binary Optimization, or QUBO for short, is an NP-hard [55] mathematical optimization problem of minimizing a quadratic objective function $F : \mathbb{Z}_2^n \to \mathbb{R}$. The objective function is defined by an upper-triangular $n \times n$ matrix Q and is of the form $F(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$, where $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ is a n-vector of binary (Boolean) variables. Formally, QUBO problems are of the form:

$$x^* = \min_{\mathbf{x}} \sum_{i \le j} x_i Q_{(i,j)} x_j, \text{ where } x_i \in \mathbb{Z}_2. \tag{2.1}$$

In other words, the goal is to find a binary value assignment of variables $\mathbf{x} = (x_0, x_1, \cdots, x_{n-1})$ such that the value of $F(\mathbf{x})$ is minimum. We typically use x^* to denote the minimum value of $F(\mathbf{x})$ and $\mathbf{x}^* = (x_0^*, x_1^*, \cdots, x_{n-1}^*)$ to denote the value assignment of the n variables that yield x^*.

We also need to use the following vector operation:

Definition 18. *Let $\mathbf{x} = (x_0, x_1, \cdots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \cdots, y_{m-1})$ be two vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively. The* ***concatenation*** *of $\mathbf{x}$ and $\mathbf{y}$ denoted by $\mathbf{xy}$ is a new vector $\mathbf{z} \in \mathbb{R}^{n+m}$ where $\mathbf{z} = (x_0, x_1, \cdots, x_{n-1}, y_0, y_1, \cdots, y_{m-1})$.*

Although the focus of this thesis is based on the QUBO model, some concepts and ideas from the Ising model and Integer Programming (IP) are also used, so we will introduce them here as well. The Ising model is a physics problem in nature. As mentioned in Chapter 1, we do not want to be overly concerned with the physical nature of the hardware, so only an abstract mathematical formulation of the problem will be given here. At a high level, the Ising Minimization Problem is an NP-hard optimization problem [5]. It has been studied extensively in the past, a more detailed study of the problem can be found in [41].

Let $G = (V, E)$ be a graph where each vertex $i \in V$ is associated with a value h_i and each edge $\{i, j\} \in E$ is associated with a value $J_{(i,j)}$, we assume all h_i and $J_{(i,j)}$ are real numbers. An Ising Minimization Problem with $n = |V|$ variables associated with G has a variable vector $\mathbf{x} = (x_0, x_1, \cdots, x_{n-1})$ and is of the following form:

$$x^* = \min_{\mathbf{x}} \sum_{\{i,j\} \in E} x_i J_{(i,j)} x_j + \sum_{i \in V} h_i x_i, \text{ where } x_i \in \{-1, 1\}. \tag{2.2}$$

Similar to the QUBO problem, the goal is to find a variable assignment of the vector $\mathbf{x}$ that has a minimum value in the objective function.

There is a strong similarity between objective functions 2.1 and 2.2. To be specific, the objective function of the Ising problem contains quadratic terms as well as linear terms while QUBO only has the former. Note that we can actually interchange linear terms and square terms in a QUBO problem without changing the optimality of its solutions since all variables in the objective function are binary, this is discussed in more details in Section 2.4. This property can also be exploited to provide a simple translation between the QUBO and Ising model. Let x^* and $\mathbf{x}^*$ be the optimal solution of a QUBO instance with n variables, this QUBO instance can be transformed into an equivalent Ising instance with optimal solution y^* and $\mathbf{y}^*$ such that $y_i = 2x_i - 1$ for all $0 \le i < n$ and $y^* = x^* + c$ where c is a real constant offset. More details of this transformation can be found in [12].

Integer Programming (see [57]) is also an optimization problem that has many different equivalent definitions, we will provide a formal definition for use within the scope of this thesis.

Integer Programming Optimization Problem (canonical form):

Instance: A $n \times m$-matrix A, a n-vector $\mathbf{c}$ and a m-vector $\mathbf{b}$ of integers.
Question: Find a n-vector $\mathbf{x}$ of integers such that the *objective function* $\mathbf{c}^T\mathbf{x}$ is minimum subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge 0$.

Most of the problems discussed in this thesis can be formulated in IP efficiently (i.e. polynomial time reduction). Since IP is a well-known problem and many well optimized libraries exist (e.g. [52]), we primarily use it to verify some of the QUBO formulations we present here.

2.3 General methodology

Now, we have introduced all the tools we need to describe the general approach we take to solve practical problems using the D-Wave quantum computers. As mentioned in Chapter 1, the current D-Wave computers can solve problems in either QUBO or Ising form. As a computer scientist, it is more natural for us to think in terms of a binary model, hence the we will only focus on the QUBO formulations here.

Given a problem P, the first thing we need to transform P to an instance of a QUBO problem. The QUBO instance then has to be fitted onto the actual hardware of the quantum machine. This is done by treating the QUBO matrix as an adjacency matrix of a graph and find a minor embedding of the QUBO structure on the host Chimera graph. And then, the D-Wave quantum computer will be queried to solve the QUBO problem and the optimal solution x^* as well as the variable assignment $\mathbf{x}^*$ will be returned. And finally, we need some mechanism to convert x^* and $\mathbf{x}^*$ back into a solution for the original problem. Since solution of P is 'encoded' in x^* and $\mathbf{x}^*$, we will call this mechanism a *decoder* function which maps $\mathbf{x}^*$ (and/or x^*) to a solution of P.

There are several difficulties we need to overcome when applying the process described in the previous paragraph to an actual problem. Let P denote the problem we wish to solve and $sol(P)$ denote its solution space. The problem P may not be an optimization problem in nature (for instance, the Graph Isomorphism Problem studied in Chapter 4), hence here lies the first obstacle; How do we convert a classical combinatorial problem (the focus of this paper) into a specific type of optimization problem? Not to mention we also need to consider the scale, that is, the number of variables, of the QUBO instance and the density of the QUBO graph we obtain after the transformation. As this is related to the next point.

To execute the QUBO instance on the quantum machine, a minor embedding of the QUBO graph on the host architecture has to be found first. In other words, we have to answer the following question:

Minor Containment Problem:

Instance:	Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.
Question:	Find the minor embedding of G_1 onto G_2 if one exists.

The Minor Containment Problem is an NP-complete problem [25], the inevitability of solving the problem (based on the current hardware design) is one of the major blocking stone in achieving any kind of speedup using the quantum computer.

Based on the definition of a graph minor, it is obvious that for a given G_1, the problem is easier to solve if G_2 is sufficiently larger. That is, if the order and size of G_2 are much bigger than those of G_1, it is typically easier to find a minor embedding of G_1 on G_2. Consider the following ideal scenario, if G_2 is a complete graph K_n, then all graphs G_1 with order $n' \leq n$ can be embedded efficiently. Since G_1 will always be a subgraph of K_n, by the definition of a graph minor, we can just remove unneeded vertices and edges. It may not be practical to build a complete hardware structure from an engineering point of view but it is still the motivation for building hardwares with larger scale (order) and higher connectivity (size).

For any generation of the hardware, the chipset is always a fixed architecture. Unlike a classical computer that can store partial results of some computation and use them again later, the D-Wave quantum computers can be seen as capable of executing only one instruction, solving the QUBO problem to be specific, with no way of pausing the computation and read or store any intermediate values. Therefore the scale of the hardware itself limits the biggest QUBO instance it can solve. A trivial example would be a guest graph with order bigger than the number of available qubits of the host; such instance is guaranteed not to be a graph minor of the host. Recall that P is the problem we wish to solve. The way we define the transformation from P to a QUBO instance may lead to different QUBO graphs with different order and size. Ideally, we want the order and size to be as small as possible, but since the variable assignment $\mathbf{x}$ needs to be able to represent all potential candidate in $sol(P)$, we need to have at least $\lg(|sol(P)|)$ binary variables. This theoretical lower bound on the number of variables is often very difficult to achieve however, as will be seen in Chapter 3 and 4. Nonetheless, the study on the Graph Isomorphism Problem in Chapter 4 also shows the drastic difference on both the scale and density between a specifically constructed QUBO instance for the problem and one obtained from a more generic approach.

2.4 Non-quadratic functions

The definition of QUBO problem only allows the objective function $F : 2^n \to \mathbb{R}$ to contain quadratic terms. However, based on the transformation we define from the problem P to QUBO, the objective function F may contain non-quadratic terms. In this section, we provide a general strategy for modifying such F that contains constant and linear as well as quadratic terms in a way such that the optimal solutions of (2.1) can be preserved.

Let $F : 2^n \to \mathbb{R}$ and $F' : 2^n \to \mathbb{R}$ denote the objective function before and after this modification respectively. Two operations will be performed. First, any constant term is ignored. Removing a constant does not change the optimality of $\mathbf{x}^*$ as the value of $F(\mathbf{x})$ is reduced by a constant amount for all $\mathbf{x} \in \mathbb{Z}_2^n$. It does however change the value of x^*, so to obtain the original minimum value of F, the constant has to be added to F' as an offset, that is, $F(\mathbf{x}) = F'(\mathbf{x}) + c$ for some constant offset $c \in \mathbb{R}$. Second, all linear terms x_i in F are replaced by x_i^2. Since all variables $x_i \in \{0, 1\}$, we have $x_i = x_i^2$ and therefore this operation has no effect on the value of x^* nor the optimality of $\mathbf{x}$. After these two steps, F' is a function that only contains quadratic terms which we can obtain a valid matrix form of. The constant c can be computed by summing up all the constant terms in F so we can easily transform the solutions of F' to their corresponding values and variable assignments in F.

More complicated scenarios do occur of course. For instance, F could be a polynomial of degree 3 or higher. In general, the order of F can be reduced at the cost of adding extra variables. Such scenarios are outside the scope of this thesis, so we will discuss them here. A more detailed study can be found in [12].

Chapter 3

Graph Covering Problems

We study two main optimization problems in this chapter. One is NP-hard and the other is polynomial-time solvable, but our QUBO formulations are very similar in complexity (e.g. the two problems require $O(n + n \lg n)$ and $O(m + n \lg n)$ qubits respectively for graphs of order n and size m). We will formally define dominating set and edge cover below:

Definition 19. *Given a graph $G = (V, E)$, a* ***dominating set*** *D of G is a subset of V, such that for every vertex $v \in V$, either $v \in D$ or $w \in D$, where w is a neighbor of v.*

Definition 20. *Given a graph $G = (V, E)$, an* ***edge cover*** *C of G is a subset of E, such that for every vertex $v \in V$, v is incident to at least one edge in C.*

The two problems of interest involve finding the smallest such D and C, that is, a dominating set with the minimum number of vertices and an edge cover with the minimum number of edges. For convenience, we assume all graphs are connected and have at least one edge.

Dominating Set Problem:

Instance:	A graph $G = (V, E)$.
Question:	What is the smallest subset D of V such that D is a dominating set of G?

Edge Cover Problem:

Instance:	A graph $G = (V, E)$.
Question:	What is the smallest subset C of E such that C is an edge cover of G?

The decision version of the Dominating Set Problem was one of the original classic problems included by Garey and Johnson [25]. It is also one of the harder NP-complete problems being classified as W[2]-hard when considering parameterized complexity [3]. An extensive history on this problem may be found in [30]. Contrastly, solving the Edge Cover Problem for graphs (without isolated vertices) is easily achievable in polynomial time. This is done by observing that the smallest edge cover is equal to the order of the graph minus its maximum matching size [36].

3.1 QUBO formulation

3.1.1 Dominating Set

We provide a simple QUBO formulation of the Dominating Set Problem. The best known exact algorithm to solve the Dominating Set Problem has time complexity $O(2^{0.610n})$ [24]. Given a graph $G = (V, E)$ with n vertices, let $V = \{v_0, v_1, \ldots, v_{n-1}\}$, this formulation requires $n + \sum_{v_i \in V}(\lfloor \lg(\Delta(v_i)) \rfloor + 1)$ binary variables, that is, for every vertex v_i in G, we need one variable x_i to represent v_i as well as $\lfloor \lg(\Delta(v_i)) \rfloor + 1$ redundant binary variables for each vertex. For the sake of readability, we will label these redundant variables as $y_{i,k}$, where $0 \le k \le \lfloor \lg(\Delta(v_i)) \rfloor$. Thus we have a vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}, y_{0,0}, \ldots, y_{n-1,\lfloor \lg(\Delta(v_n)) \rfloor})$ of named variables.

The objective function that is to be minimized is of the form:

$$F(\mathbf{x}) = \sum_{v_i \in V} x_i + A \sum_{v_i \in V} P_i$$

where

$$P_i = \left(1 - \left(x_i + \sum_{v_j \in N(v_i)} x_j\right) + \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k}\right)^2 \tag{3.1}$$

Assuming the optimal solution of the objective function is obtained. We now need to obtain a solution of the Dominating Set Problem from x^* and $\mathbf{x}^*$, as described in Section 2.3. For the Dominating Set Problem, the decoder function $D(\mathbf{x}) : \mathbb{Z}_2^{|\mathbf{x}|} \to 2^V$ where 2^V is the power set of V is defined as follows:

$$D(\mathbf{x}) = \{v_i \mid x_i = 1\}.$$

The function maps a binary vector $\mathbf{x}$ to a subset of V as the dominating set.

In the objective function, $A > 1$ is a real positive constant and the term $\sum_{v_i \in V} x_i$ represents a penalty for the size of the chosen set, and P_i serves as a penalty if a non-dominating set is chosen. If the assignment of the variables is a dominating set, then for each vertex v_i in G, we have $x_i + \sum_{v_j \in N(v_i)} x_j \ge 1$. And therefore $1 - (x_i + \sum_{v_j \in N(v_i)} x_j) \le 0$. Finally, we use the term $\sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k}$ to counter balance the penalty if more than one vertex in the set $v_i \cup N(v_i)$ is chosen as it does not violate the definition of a dominating set and should not be penalized. In the worst case, $1 - (x_i + \sum_{v_j \in N(v_i)} x_j) = -\Delta(v_i)$ where v_i and all of its neighbors are chosen, so a total number of $(\lfloor \lg(\Delta(v_i)) \rfloor + 1)$ redundant variables are needed to represent integers up to $\Delta(v_i)$. Hence the total number of binary variables of this formulation is $O(n + n \lg n)$ in the worst case.

Theorem 1. *The objective function (3.1) is correct.*

Proof. First, we show that it is always possible to transform a non-dominating set into a dominating set which will have a smaller value in the objective function.

Suppose we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ and $D(\mathbf{x}^*)$ is not a dominating set where $\mathbf{x}^*$ corresponds to the variable assignment yielding x^*. Then there must exist some vertices such that these vertices themselves nor any of their neighbors are present in $D(\mathbf{x}^*)$. Then the corresponding penalty P_i for each of these vertices will be at least 1 by the definition of P_i. Therefore, if we set the corresponding x_i of these vertices to 1, then for each one of them, a penalty of size 1 will be added to the term $\sum_{v_i \in V} x_i$ while the corresponding P_i will be reduced to 0 by setting $y_{i,0}$ to 1. So $F(\mathbf{x}^*)$ will be reduced by at least $A - 1$. Hence the solution from $D(\mathbf{x}^*)$ will always be a dominating set.

The second part of the proof is to show that an assignment of $\mathbf{x}$ that produces a smaller dominating set will have a smallest value in the objective function. This is trivial as if $D(\mathbf{x})$ is a dominating set, then each P_i will have to be 0, by the argument from the previous paragraph. Therefore the value of the objective function solely depends on the number of 1 entires in the term $\sum_{v_i \in V} x_i$. And from the definition of the decoder function D, it is fairly easy to see that a less sum from $\sum_{v_i \in V} x_i$ will be mapped to a smaller dominating set by D. $\square$

3.1.2 Dominating Set Q_3 example

In this subsection, we will provide an example of the QUBO formulation (3.1) on Q_3. Formally, The hypercube Q_3 is defined as follows. The vertices of Q_3 are $V = \{0, 1, \ldots, 7\}$ and the edges are

$$E = \{\{0,1\},\{0,2\},\{0,4\},\{1,3\},\{1,5\},\{2,3\},\{2,6\},\{3,7\},\{4,5\},\{4,6\},\{5,7\},\{6,7\}\}.$$

It can be visualized as a 3-dimensional cube where the each corner of the cube is a vertex.

Now, by expanding the bracket in objective function (3.1), we get

$$\sum_{v_i\in V} x_i + A\sum_{v_i\in V}\Bigg(1 - x_i - \sum_{v_j\in N(v_i)} x_j + \sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} - x_i + x_i^2 + x_i\sum_{v_j\in N(v_i)} x_j$$
$$-x_i\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} - \sum_{v_j\in N(v_i)} x_j + x_i\sum_{v_j\in N(v_i)} x_j + \Bigg(\sum_{v_j\in N(v_i)} x_j\Bigg)^2 - \sum_{v_j\in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k}$$
$$+\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} - x_i\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} - \sum_{v_j\in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} + \Bigg(\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k}\Bigg)^2\Bigg) \tag{3.2}$$

Note that the function (3.2) has some constant and linear terms, we take the steps described in Section 2.4 and summing up similar terms to obtain

$$(1-A)\sum_{v_i\in V} x_i^2 + A\sum_{v_i\in V}\Bigg(-2\sum_{v_j\in N(v_i)} x_j^2 + 2\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k}^2 + 2x_i\sum_{v_j\in N(v_i)} x_j$$
$$-2x_i\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} + \Bigg(\sum_{v_j\in N(v_i)} x_j\Bigg)^2 - 2\sum_{v_j\in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} + \Bigg(\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k}\Bigg)^2\Bigg) \tag{3.3}$$

Now, we can finally obtain a valid matrix representation of the objective function. Let $A = 2$, the matrix representation of the quadratic objective function (3.3) for Q_3 is shown in Table 3.1. The entries $Q_{i,j}$ where $i \le j$ in the matrix is computed by extracting the coefficient of each quadratic term from the objective function.

After solving for $x^* = \min_{\mathbf{x}} F(\mathbf{x})$, we obtain four optimal solutions.

$$\mathbf{x}_1 = [1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

$$\mathbf{x}_2 = [0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

$$\mathbf{x}_3 = [0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

and

$$\mathbf{x}_4 = [0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

And we have $D(\mathbf{x}_1) = \{0,7\}$, $D(\mathbf{x}_2) = \{1,6\}$, $D(\mathbf{x}_3) = \{2,5\}$ and $D(\mathbf{x}_4) = \{3,4\}$. It can be verified quite easily that these four solutions (pairs of vertices of distance 3) are all minimum dominating sets of Q_3 with the same size.

Table 3.1: Dominating Set QUBO matrix for Q_3

variables	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$y_{0,0}$	$y_{0,1}$	$y_{1,0}$	$y_{1,1}$	$y_{2,0}$	$y_{2,1}$	$y_{3,0}$	$y_{3,1}$	$y_{4,0}$	$y_{4,1}$	$y_{5,0}$	$y_{5,1}$	$y_{6,0}$	$y_{6,1}$	$y_{7,0}$	$y_{7,1}$
x_0	**-7**	8	8	8	8	8	8	0	-4	-8	-4	-8	-4	-8	0	0	-4	-8	0	0	0	0	0	0
x_1		**-7**	8	8	8	8	0	8	-4	-8	-4	-8	0	0	-4	-8	0	0	-4	-8	0	0	0	0
x_2			**-7**	8	8	0	8	8	-4	-8	0	0	-4	-8	-4	-8	0	0	0	0	-4	-8	0	0
x_3				**-7**	0	8	8	8	0	0	-4	-8	-4	-8	-4	-8	0	0	0	0	0	0	-4	-8
x_4					**-7**	8	8	8	-4	-8	0	0	0	0	0	0	-4	-8	-4	-8	-4	-8	0	0
x_5						**-7**	8	8	0	0	-4	-8	0	0	0	0	-4	-8	-4	-8	0	0	-4	-8
x_6							**-7**	8	0	0	0	0	-4	-8	0	0	-4	-8	0	0	-4	-8	-4	-8
x_7								**-7**	0	0	0	0	0	0	-4	-8	0	0	-4	-8	-4	-8	-4	-8
$y_{0,0}$									**6**	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_{0,1}$										**16**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_{1,0}$											**6**	8	0	0	0	0	0	0	0	0	0	0	0	0
$y_{1,1}$												**16**	0	0	0	0	0	0	0	0	0	0	0	0
$y_{2,0}$													**6**	8	0	0	0	0	0	0	0	0	0	0
$y_{2,1}$														**16**	0	0	0	0	0	0	0	0	0	0
$y_{3,0}$															**6**	8	0	0	0	0	0	0	0	0
$y_{3,1}$																**16**	0	0	0	0	0	0	0	0
$y_{4,0}$																	**6**	8	0	0	0	0	0	0
$y_{4,1}$																		**16**	0	0	0	0	0	0
$y_{5,0}$																			**6**	8	0	0	0	0
$y_{5,1}$																				**16**	0	0	0	0
$y_{6,0}$																					**6**	8	0	0
$y_{6,1}$																						**16**	0	0
$y_{7,0}$																							**6**	8
$y_{7,1}$																								**16**

3.1.3 Edge Cover

The QUBO formulation of the Edge Cover Problem here is quite similar to the Dominating Set Problem given in the previous subsection. The Edge Cover Problem can be solved in polynomial time by exploiting the fact that the order of a graph G is equal to the size of its maximum matching plus the size of its minimum edge cover [20, 43].

Given a graph $G = (V, E)$ with n vertices and m edges, let $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E = \{\{i, j\} \mid v_j \in N(v_i)\}$. This formulation requires one binary variable $x_{i,j}$ for each $\{i, j\} \in E$, as well as $\lfloor \lg(\Delta(v_i) - 1) \rfloor + 1$ redundant binary variables for each $v_i \in V$. The objective function that is to be minimized is of the form:

$$F(\mathbf{x}) = \sum_{ij \in E} x_{i,j} + A \sum_{v_i \in V} P_i$$

where

$$P_i = \left(1 - \sum_{ij \in I(v_i)} x_{i,j} + \sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1) \rfloor} 2^k y_{i,k}\right)^2 \tag{3.4}$$

The decoder function we use this time is $C(\mathbf{x}) : \mathbb{Z}_2^{|\mathbf{x}|} \rightarrow 2^E$ where 2^E is the power set of E and we take $C(\mathbf{x}) = \{\{i,j\} \mid x_{i,j} = 1\}$ as the edge cover of G. Again, choosing $A > 1$ is sufficient for this formulation to be correct.

The structure and purpose of each term in the objective function is almost identical to the Dominating Set Problem. One thing to note is that the number of redundant variable required for each vertex is slightly smaller in some cases. As $1 - \sum_{ij \in I(i)} x_{i,j} \leq -(\Delta(v_i)-1)$, only $\lfloor \lg(\Delta(v_i)-1) \rfloor + 1$ redundant variables are needed to counter balance in case more than one edge incident to a vertex v_i is chosen as the edge cover when $\Delta(v_i) > 1$. If $\Delta(v_i) = 1$, then no redundant variables are needed at all as the only edge incident to v_i must be chosen in the edge cover set. In all cases, $1 - \sum_{ij \in I(i)} x_{i,j}$ has to be 0. The argument that will be used here to show the correctness of this formulation is quite similar to the proof in the previous subsection.

Theorem 2. *The objective function (3.4) is correct.*

Proof. First, we show that a solution from $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ will always be an edge cover. Suppose we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ with corresponding binary vector $\mathbf{x}^*$ and $C(\mathbf{x}^*)$ is not an edge cover. Then there must exist a set of vertices $\{u_1, u_2, \ldots, u_l\}$ such that $I(u_i) \cap C(\mathbf{x}^*) = \emptyset$ for all $1 \leq i \leq l$. That is, for each u_i, none of the edges incident to u_i is in $C(\mathbf{x}^*)$. Hence, for each $u_i, 1 \leq i \leq l$, the corresponding P_i is 1. If we change the variable $x_{i,j}$ corresponding to one of these edges to 1, then again, we reduce $F(\mathbf{x})$ by at least $A - 1$.

Now since $C(\mathbf{x}^*)$ where $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ has to be an edge cover as shown in the previous paragraph, it also has to be the smallest edge cover. When $C(\mathbf{x})$ is an edge cover, each P_i in $F(\mathbf{x})$ has to be 0 and therefore the value of $F(\mathbf{x})$ is the size of $C(\mathbf{x})$. Hence by minimizing $F(\mathbf{x})$, we also minimize the size of the edge cover set $C(\mathbf{x})$. $\square$

3.1.4 Edge Cover S_{15} example

Similar to the Dominating Set Problem, we will provide an example of the actual encoding of the objective function (3.4) on a star graph to a QUBO matrix here. The family of star graphs is formally defined below:

Definition 21. *The family of* ***star graphs*** *is defined as follows: A star graph with order $n+1$, denoted by $S_n = (V, E)$ has $V = \{0, 1, \cdots, n\}$ and $E = \{\{0, i\} \mid 1 \leq i \leq n\}$.*

Once again, the objective function (3.4) can not be encoded straight away into QUBO, constant and linear terms have to be replaced just like in the Dominating Set Problem. Doing so would give us

$$\sum_{ij\in E} x_{i,j}^2 + A\sum_{v_i\in V}\Bigg(-2\sum_{ij\in I(v_i)} x_{i,j}^2 + 2\sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1)\rfloor} 2^k y_{i,k}^2 + \sum_{ij\in I(v_i)} x_{i,j}\sum_{ij\in I(v_i)} x_{i,j} \\ -2\sum_{ij\in I(v_i)} x_{i,j}\sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1)\rfloor} 2^k y_{i,k} + \Bigg(\sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1)\rfloor} 2^k y_{i,k}\Bigg)^2\Bigg) \quad (3.5)$$

The encoded QUBO matrix corresponds to objective function (3.5) is shown in Table 3.2. The solution to the minimum Edge Cover Problem is trivial for the family of star graphs, since all vertices labeled from 1 to n are all of degree 1 and is only connected to vertex 0, any edge cover in star graphs would have to consists of all the edges in the graph. And by solving $x^* = \min_{\mathbf{x}} \sum_{i\leq j} x_i Q_{(i,j)} x_j$, we obtain an unique solution in this case where

$$\mathbf{x} = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1] \text{ and } C(\mathbf{x}) = E$$

which can be verified quite easily the only minimum edge cover for S_{15}. Note that the only zero value in $\mathbf{x}$ correspondes to the variable $y_{0,0}$, which allows for $P_0 = (1-15+2+4+8)^2 = 0$.

Table 3.2: Edge Cover QUBO matrix for S_{15}

variables	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{0,4}$	$x_{0,5}$	$x_{0,6}$	$x_{0,7}$	$x_{0,8}$	$x_{0,9}$	$x_{0,10}$	$x_{0,11}$	$x_{0,12}$	$x_{0,13}$	$x_{0,14}$	$x_{0,15}$	$y_{0,0}$	$y_{0,1}$	$y_{0,2}$	$y_{0,3}$
$x_{0,1}$	**-3**	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,2}$		**-3**	2	2	2	2	2	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,3}$			**-3**	2	2	2	2	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,4}$				**-3**	2	2	2	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,5}$					**-3**	2	2	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,6}$						**-3**	2	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,7}$							**-3**	2	2	2	2	2	2	2	2	0	0	0	0
$x_{0,8}$								**-3**	2	2	2	2	2	2	2	0	0	0	0
$x_{0,9}$									**-3**	2	2	2	2	2	2	0	0	0	0
$x_{0,10}$										**-3**	2	2	2	2	2	0	0	0	0
$x_{0,11}$											**-3**	2	2	2	2	0	0	0	0
$x_{0,12}$												**-3**	2	2	2	0	0	0	0
$x_{0,13}$													**-3**	2	2	0	0	0	0
$x_{0,14}$														**-3**	2	0	0	0	0
$x_{0,15}$															**-3**	0	0	0	0
$y_{0,0}$																**6**	4	8	16
$y_{0,1}$																	**16**	16	32
$y_{0,2}$																		**48**	64
$y_{0,3}$																			**160**

3.2 Weighted problems

The formulations provided in the previous section can be modified quite easily to adapt to weighted graphs. The definitions of the input and output of the Dominating Set and Edge Cover Problems are slightly different in weighted graphs. For the Weighted Dominating Set Problem, each vertex v_i in the graph is assigned a real positive weight w_i by a weight function $W : V \to \mathbb{R}^+$ and the goal is to find a dominating set that has a minimum sum of the weights. The weighted sum function $S : 2^V \to \mathbb{R}^+$ is defined as $S(A) = \sum_{v \in A} W(v)$.

Likewise, in the Weighted Edge Cover problem, each edge $\{i, j\}$ in G is associated with a real positive weight $w_{i,j}$ defined $W : E \to \mathbb{R}^+$ by and the goal is to find an edge cover that has a minimum value in the weighted sum function $S : 2^E \to \mathbb{R}^+$ defined as $S(A) = \sum_{ij \in A} W(ij)$. Formally, we have the following definitions.

Weighted Dominating Set Problem:

Instance:	A graph $G = (V, E)$ and a weight function $W : V \to \mathbb{R}^+$.
Question:	Find the dominating set D such that $S(D)$ is minimum over all possible dominating sets.

Weighted Edge Cover Problem:

Instance:	A graph $G = (V, E)$ and a weight function $W : E \to \mathbb{R}^+$.
Question:	Find the edge cover C such that $S(C)$ is minimum over all possible edge covers.

In both problems above we restrict to positive weights since otherwise those non-positive vertices (edges) would always be added to a minimum solution and we could reduce to a strictly positive subproblem.

3.2.1 Weighted Dominating Set

For the Weighted Dominating Set Problem, the objective function is almost identical to the unweighted version. Let $w_i = W(v_i)$, we have

$$F(\mathbf{x}) = \sum_{v_i \in V} w_i x_i + A \sum_{v_i \in V} P_i$$

where

$$P_i = \left(1 - \Big(x_i + \sum_{v_j \in N(v_i)} x_j\Big) + \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k}\right)^2 \tag{3.6}$$

Every term serves the same purpose here except that A has to be picked with the property that $A > \max\{w_i \mid v_i \in V\}$. And once again, we take $D(\mathbf{x}) = \{v_i \mid x_i = 1\}$ as the solution at the end. The following proof of correctness of the above formulation is very similar to the proof of the unweighted version as well.

Theorem 3. *The QUBO formulation in (3.6) is correct.*

Proof. First, we show that it is always possible to transform a non-dominating set into a dominating set which will have a smaller value in the objective function. Suppose we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ with corresponding binary vector $\mathbf{x}^*$ and $D(\mathbf{x}^*)$ is not a dominating set. If this is case, then there must exist some vertices such that these vertices themselves nor any of their neighbors are present in $D(\mathbf{x}^*)$. Then the corresponding penalty P_i for each of these vertices will be 1. Therefore, if we set the corresponding x_i of these vertices to 1, then for each one of them, a penalty of size w_i will be added to the term $\sum_{v_i \in V} x_i$ while the corresponding P_i will be reduced to 0 and so $F(\mathbf{x}^*)$ will be reduced by $A - w_i > 0$ by choice of A. Hence the solution from $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ will always be a dominating set.

The second part of the proof is to show that an assignment of x that produces a smaller dominating set will have a smaller value in the objective function. It is trivial as if $D(\mathbf{x})$ is a dominating set, then each P_i will have to be 0, so the value of the objective function solely depend on the weights of vertices chosen to be in the dominating set. □

3.2.2 Weighted S_5 example

Let us use the star graph again to demonstrate the difference for Weighted Dominating Set Problem. The weight function W is defined as follows:

$$W(v) = \begin{cases} 5, & \text{if } v = 0 \\ 1, & \text{otherwise} \end{cases}$$

Table 3.3: Dominating Set QUBO matrix for weighted S_5

variables	x_0	x_1	x_2	x_3	x_4	x_5	$y_{0,0}$	$y_{0,1}$	$y_{0,2}$	$y_{1,0}$	$y_{2,0}$	$y_{3,0}$	$y_{4,0}$	$y_{5,0}$
x_0	**-115**	80	80	80	80	80	-40	-80	-160	-40	-40	-40	-40	-40
x_1		**-39**	40	40	40	40	-40	-80	-160	-40	0	0	0	0
x_2			**-39**	40	40	40	-40	-80	-160	0	-40	0	0	0
x_3				**-39**	40	40	-40	-80	-160	0	0	-40	0	0
x_4					**-39**	40	-40	-80	-160	0	0	0	-40	0
x_5						**-39**	-40	-80	-160	0	0	0	0	-40
$y_{0,0}$							**60**	80	160	0	0	0	0	0
$y_{0,1}$								**160**	320	0	0	0	0	0
$y_{0,2}$									**480**	0	0	0	0	0
$y_{1,0}$										**60**	0	0	0	0
$y_{2,0}$											**60**	0	0	0
$y_{3,0}$												**60**	0	0
$y_{4,0}$													**60**	0
$y_{5,0}$														**60**

The encoded QUBO matrix is shown in Table 3.3. Solving $x^* = \min_{\mathbf{x}} \sum_{i \leq j} x_i Q_{(i,j)} x_j$ this time gives two different solutions with identical value objective function (3.6). The two solutions are

$$\mathbf{x}_1 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

and

$$\mathbf{x}_2 = [0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0]$$

The number of vertices in these two dominating sets is different, $D(\mathbf{x}_1)$ has only one vertex while $D(\mathbf{x}_2)$ has five vertices.

The solution to the Dominating Set Problem is trivial for the family of star graphs S_n in the unweighted case, since all vertices labeled from 1 to n are all only connected to vertex 0, choosing just vertex 0 as the dominating set is sufficient to cover all vertices in the graph. In the weighted case however, if the sum of weights of vertices 1 to n is smaller than the weight of vertex 0, then the minimum dominating set would actually consists of all vertices labeled 1 to n. In our case provided above, the weight function W is constructed in a way such that the two cases would have the same weighted sum, and as a result, both are accepted as the optimal solution.

3.2.3 Weighted Edge Cover

Similar to the Edge Cover Problem, the Weighted Edge Cover Problem can be reduced to Weighted Perfect Matching problem which is solvable in time $O(n^3)$ [48, 34][1]. Similar to the weighted dominating set formulation in the previous subsection, we only need to do some small modification to the Edge Cover Problem to obtain a QUBO formulation for the weighted version. Let $w_{i,j} = W(ij)$, we have

$$F(\mathbf{x}) = \sum_{ij \in E} w_{i,j} x_{i,j} + A \sum_{v_i \in V} P_i$$

where

$$P_i = \left(1 - \sum_{ij \in I(v_i)} x_{i,j} + \sum_{k=0}^{\lfloor \lg(\Delta(x_i) - 1) \rfloor} 2^k y_{i,k} \right)^2 \tag{3.7}$$

Once again, we need to have $A > \max\{w_{i,j} \mid \{i, j\} \in E\}$ and the function $C : \mathbb{Z}_2^{|\mathbf{x}|} \to 2^E$ from Section 3.1.3 will be used again to obtain the subset of edges. Although the argument may seem almost identical to the unweighted version, for the sake of completeness, we will present the theorem and proof formally below.

[1]We want to clarify that the justification of the reduction given in the references only applies to *minimal* edge covers (not any edge cover).

Theorem 4. *The QUBO formulation in (3.7) is correct.*

Proof. First, we show that a solution from $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ will always be an edge cover. Suppose we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ and $C(\mathbf{x}^*)$ is not an edge cover where $\mathbf{x}^*$ is the corresponding binary variable vector yielding x^*. Then there must exist a set of vertices $\{u_1, u_2, \ldots, u_l\}$ such that $I(u_i) \cap C(\mathbf{x}^*) = \emptyset$ for $1 \leq i \leq l$. That is, for each u_i, none of the edges incident to u_i is in $C(\mathbf{x}^*)$. Hence, for each $u_i, 1 \leq i \leq l$, the corresponding P_i is 1. If we change the variable $x_{i,j}$ corresponding to one of these edges to 1, then again, we reduce the value of the objective function $F(x)$ by at least $A - w_{i,j} > 0$ since A is larger than all $w_{i,j}$. Therefore the optimal solution to the minimization problem of $F(\mathbf{x})$ will always be an edge cover set.

Thus since $C(x^*)$ corresponding to $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ has to be an edge cover as shown in the previous paragraph. It also has to be the smallest edge cover since each P_i in $F(\mathbf{x})$ has to be 0 and therefore the value of $F(\mathbf{x})$ is completely dependent on the weights of the edges chosen to be in $C(\mathbf{x})$. □

3.2.4 Weighted W_5 example

A wheel graph W_n with order $n+1$ is defined similar to a star graph. To be precise, a star graph S_n with $n+1$ vertices is always a subgraph of W_n, with extra edges joining the outer pendent vertices into a cycle of length n. Taking $n = 5$, we have $V = \{0, 1, 2, 3, 4, 5\}$ and $E = \{\{0, i\} \mid 1 \leq i \leq n\} \cup \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 1\}\}$. For the following example, the weight function $W : E \to \mathbb{R}^+$ which assigns a weight to each edge is defined as follows:

$$W(e) = \begin{cases} 6, & \text{if } e = \{0, i\} \text{ where } 1 \leq i \leq n \\ 12, & \text{if } e = \{1, 2\} \\ 15, & \text{otherwise} \end{cases}$$

Let $A = 20$, the QUBO matrix encoded from objective function (3.7) is shown in Table 3.4. Once again, we get two optimal solutions which both have the same value in objective function (3.7) in this case.

$$\begin{aligned} \mathbf{x}_1 &= [0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \\ \mathbf{x}_2 &= [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \end{aligned}$$

The number of edges we obtain in the edge cover are four and five respectively. Although choosing the edge $\{1, 2\}$ to cover vertex 1 and 2 may seem better initially, since it covers two vertices with only one edge. Choosing $\{0, 1\}$ and $\{0, 2\}$ instead makes no difference in this case as $W(\{1, 2\}) = W(\{0, 1\}) + W(\{0, 2\})$, so the weighted sum is identical.

Table 3.4: Edge Cover QUBO matrix for weighted W_5

vars	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{0,4}$	$x_{0,5}$	$x_{1,2}$	$x_{1,5}$	$x_{2,3}$	$x_{3,4}$	$x_{4,5}$	$y_{0,0}$	$y_{0,1}$	$y_{0,2}$	$y_{1,0}$	$y_{1,1}$	$y_{2,0}$	$y_{2,1}$	$y_{3,0}$	$y_{3,1}$	$y_{4,0}$	$y_{4,1}$	$y_{5,0}$	$y_{5,1}$
$x_{0,1}$	**-34**	20	20	20	20	20	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{0,2}$		**-34**	20	20	20	20	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{0,3}$			**-34**	20	20	0	0	20	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{0,4}$				**-34**	20	0	0	0	20	20	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{0,5}$					**-34**	0	20	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{1,2}$						**-28**	20	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{1,5}$							**-25**	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,3}$								**-25**	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,4}$									**-25**	20	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{4,5}$										**-25**	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_{0,0}$											**60**	40	80	0	0	0	0	0	0	0	0	0	0
$y_{0,1}$												**160**	160	0	0	0	0	0	0	0	0	0	0
$y_{0,2}$													**480**	0	0	0	0	0	0	0	0	0	0
$y_{1,0}$														**60**	40	0	0	0	0	0	0	0	0
$y_{1,1}$															**160**	0	0	0	0	0	0	0	0
$y_{2,0}$																**60**	40	0	0	0	0	0	0
$y_{2,1}$																	**160**	0	0	0	0	0	0
$y_{3,0}$																		**60**	40	0	0	0	0
$y_{3,1}$																			**160**	0	0	0	0
$y_{4,0}$																				**60**	40	0	0
$y_{4,1}$																					**160**	0	0
$y_{5,0}$																						**60**	40
$y_{5,1}$																							**160**

3.3 Experimental results and discussion

Experiments were conducted on the D-Wave 2X quantum computer. The chipset is a $12 \times 12 \times 4$ Chimera graph with 1152 vertices. Some physical qubits are inactive, leaving us 1098 available physical qubits to work with. Python scripts which generates the QUBO instances of the objective functions for Dominating Set and Edge Cover are available in Appendices A and B. We used the NetworkX graph package [27] in both scripts in addition to the D-Wave library [13]. We did do software simulations (e.g. conventional evolutionary search) on our QUBO matrices to verify optimal solutions were possible. The conventional evolutionary search algorithm we used was also provided by the D-Wave software package [13].

3.3.1 Results

For both the Dominating Set and the Edge Cover Problem, we did 2500 trials on each of the graphs listed in Tables 3.5 and 3.6. The first three columns of Tables 3.5 and 3.6 contain standard information about the graphs; the *order* and *size* are with respect to the input graphs rather than the QUBO formulations. We used the same graph specifications as in [10]. The next three columns contains information on the embeddings. *Logical qubits* is the number of variables of the formulation and *physical* *qubits* is the number of hardware qubits required after embedding the QUBO instance into the hardware. As can be seen from the table, the difference between the number of variables and the actual number of hardware qubits needed varies quite a lot. The high scaling factor is mostly due to the high density of the QUBO matrices. High density means the size (number of non-zero QUBO entries) of the guest graph which the QUBO matrix represents is high, and since the hardware has a fixed architecture, it is harder to embed such guest graphs with few edge contractions. Hence more active physical qubits are needed. The *embedding max chain* column contains the maximum number of physical qubits a single logical qubit is mapped to.

Note that, minor containment is not the focus of study here, so we did not implement our own minor embedding algorithm. The algorithm used in the study is provided by the D-Wave software package; more details about this particular embedding algorithm can be found in [12] and [7]. Another thing to note here is that we did not try to minimize the number of variables nor the density of the QUBO matrix when developing the objective functions given in Section 3.1. It is quite possible that better formulations exist for these problems, that is, formulations with less number of logical qubits and lower density that will make the minor containment problem on them easier to solve.

The *best answer* column is the best (smallest) solution, in terms of the size of the covering set of vertices or edges respectively for each of the problems, the D-Wave machine was able to find. The *optimal answer* column is the true optimal solution of the particular graph. The optimal solutions for the Dominating Set Problem was computed by first computing an Integer Programming formulation of the problem and then Sage Maths [52] software was used to compute the solution. A script used for computing the optimal solution is given in Appendix C. The optimal solution to the Edge Cover problem was computed by first realizing that the order of a graph G is equal to the sum of the number of edges in its maximum matching and minimum edge cover[43]. The maximum matching for each test graphs was computed using the built-in function provided by the NetworkX package.

As mentioned before, each QUBO instance was executed 2500 times. The *average valid answer* column is the average size of valid covering set found by the machine out of the the 2500 times and the *probability of valid answer* column indicates the probability of the machine finding a valid answer. The last column indicates the proportion in which the best answer (given by the D-Wave machine), not necessarily optimal, was found out of the 2500 times.

3.3.2 Scaling the Ising

The Python program in Appendix D accesses the quantum machine when solving $x^* = \min_{\mathbf{x}} f(\mathbf{x})$. Note that we explicitly converted the QUBO formulation of each test cases to its corresponding Ising form as the D-Wave software package API gives more direct control for more fine-tuned test with respect to the Ising model. The

QUBO to Ising transformation function is also provided by the D-Wave API. We then used two extra parameters s and s_2 to scale the Ising model.

The parameter s_2 is used before the embedding is applied. All entries in the Ising model is scaled linearly by $f(x) : \mathbb{R} \to \mathbb{R}$ where $f(x) = s_2 x/\texttt{maxV}$ and `maxV` is the maximum value over all entries. This scaling is applied because of the current D-Wave hardware coupling restrictions need to be in the range $[-1, 1]$, these entires corresponds to the $J_{(i,j)}$ values in the Ising problem. After the embedding is computed, $0 < s \leq 1$ is used as a factor to scale all entries corresponding to the same logical qubits. Lower values of s emphasizes that it is more important for physical qubits corresponding to the same logical qubit to be in a consistent state (spin). The lowering the value of s was recommended in [13] if the unscaled case $s = 1$ does not provide expected results.

The scaling is essentially the same as multiplying the QUBO objective function by a constant and it should have no affect on the variable assignment of the optimal solution of the original and modified functions.

3.3.3 Embedding with long chains

The entries highlighted in red in Table 3.5 and 3.6 are the test cases where the optimal solution was never found out of the 2500 trials. An interesting observation we can make here is that in Table 3.5, all such entries have a max chain length of bigger than or equal to 10. It coincides with our expectation that longer chain lengths are more likely to lead to less accurate solutions. The embedding of the problem can be seen as a transformation to a different problem which has equivalent optimal solutions[13, 12]. Since one logical qubit could be mapped to several different physical qubits, new constraints have to be introduced to enforce all the physical qubits to be in the same consistent states. This is achieved by adding extra penalties if the set of physical qubits, representing the same logical qubits, are in inconsistent (different) states [12]. Therefore, doing such transformation will make the annealing process harder and lead to a lower probability of finding the optimal solution [12, 46].

For Table 3.6 some of the highlighted entries have a max chain length of less than 10. However, the best answer for such entries are at most 1 bigger than the true optimal solution. So in some sense, it does not contradict what we stated in the previous paragraph, the shorter the chains are, the more accurate (closer to optimal) the solutions become.

3.3.4 The family of Star Graphs

The three rows with dashes in Table 3.6 corresponds to the test cases where no valid solution was found out of all 2500 trials.

Recall the definition of the family of Star graphs from Section 3.1.4. For all $n \in \mathbb{Z}$, the only edge cover of $S_n = (V, E)$ is E. Since all vertices $V \setminus \{0\}$ are of degree 1, and all edges are incident to vertex 0, the only way to cover vertex $1 \leq i \leq n$ is to pick the edge $(0, i)$. Hence all edges have to be picked to have a correct covering set. We suspect that this uniqueness about the solution is what leads to its none-discovery in these three cases. The physical nature of the computation could make the desired unique configuration of qubits impossible to reach for the quantum machine. That is why we can see that from Table 3.6, despite the fact that the Star graphs we used in the experiments being relatively small, in terms of both the order and size, the probability of finding a valid edge cover for S_n decreases dramatically as n increases when compared with other families of graphs we had.

The case of finding a valid dominating set for S_n is slightly different. Since all other vertices are connected with vertex 0, the optimal solution is obviously just the set $\{0\}$. For the dominating set experiment, the optimal solution was found for all S_n we tested. However, a valid dominating set for S_n could also include any number of other vertices. Hence it leads to what we see in Table 3.5, the probability of finding a valid dominating set is relatively high for all S_n, but the probability for finding the optimal solution once again reduces dramatically as the order increases.

3.3.5 Difference between empirical and theoretical result

As can be seen in the result, the overall outcome was positive to some extent. In most of the cases, the D-Wave quantum computer did find the true optimal solutions and the probability of finding a valid solution per trial was relatively high. We note that for the two problems with drastic classical complexities (Dominating Set being NP-complete vs Edge Cover being in P), it seems that the quantum annealing solutions are equivalent in terms of "solvability".

Google has recently published some experimental results on the new D-Wave 2X [17], the same hardware model we have used. The way that the D-Wave quantum annealer was used in the Google study is very similar to ours. To be specific, when benchmarking the (time-wise) performance of the quantum annealer, the probability of obtaining the optimal solution by a single query to the D-Wave hardware was estimated. This probability was then used to calculate the expected number of queries required to be able to obtain the optimal, within na certain degree of confidence, at least once. The Google study also had several design choices which differ from ours. First of all, the test cases Google used in their benchmarking paper were hand crafted instances fitting directly on the actual hardware. In other words, the underlying graphs they used are subgraphs of the Chimera graph, so the minor containment problem did not need to be considered. As we have mentioned in Subsection 3.3.3, longer chains apparently lead to less accurate solutions. Secondly, the lack of minor embedding also means that the solutions returned by D-Wave does not need to be mapped to the variable assignments of the logical qubits, which is then post-processed or decoded to be able to finally obtain a solution to the original problem. Even though these steps can be done in polynomial time, they still contribute to the overall time required to obtain a solution. Since our goal here is not to benchmark the performance of the machine, we did not take these factors into account.

Table 3.5: Results for some small graphs for Dominating Set.

Graph	Order	Size	Logical Qubits	Physical Qubits	Embedding Max Chain	Best Answer	Optimal Answer	Average Valid Answer	Probability of Valid Answer	Probability of Best Answer
BidiakisCube	12	18	36	180	13	4	4	5.77	87.08	2.20
Bull	5	5	13	35	5	2	2	2.70	74.68	23.56
Butterfly	5	6	16	66	11	1	1	2.81	97.64	2.92
C10	10	10	30	92	7	4	4	5.52	78.28	7.36
C11	11	11	33	103	5	4	4	5.52	71.24	7.12
C12	12	12	36	118	7	4	4	6.57	80.04	0.28
C4	4	4	12	30	4	2	2	2.24	85.48	67.08
C5	5	5	15	49	5	2	2	2.58	82.76	37.92
C6	6	6	18	60	6	2	2	3.05	84.84	17.60
C7	7	7	21	67	6	3	3	3.52	77.80	41.80
C8	8	8	24	81	6	3	3	4.25	86.92	13.44
C9	9	9	27	101	8	3	3	4.77	80.28	3.40
Chvatal	12	24	48	353	19	4	4	7.31	97.60	0.12
Clebsch	16	40	64	745	42	5	4	8.22	89.92	0.28
Diamond	4	5	12	30	4	1	1	1.91	94.48	17.76
Dinneen	9	21	36	244	13	2	2	4.75	96.48	0.32
Dodecahedral	20	30	60	409	20	7	6	10.11	63.20	0.36
Durer	12	18	36	210	11	4	4	5.79	73.08	3.92
Errera	17	45	68	809	30	5	3	8.43	95.88	0.52
Frucht	12	18	36	177	10	4	3	5.61	89.64	7.52
GoldnerHarary	11	27	41	350	20	2	2	5.63	84.72	0.04
Grid2x3	6	7	18	58	6	2	2	3.17	85.04	9.64
Grid3x3	9	12	28	145	12	3	3	4.91	74.48	1.72
Grid3x4	12	17	38	175	10	4	4	6.64	82.08	0.56
Grid4x4	16	24	52	274	17	6	4	9.01	69.64	0.36
Grid4x5	20	31	66	453	17	7	6	10.70	63.64	0.08
Grotzsch	11	20	39	251	14	3	3	5.88	80.96	0.08
Heawood	14	21	42	255	13	4	4	6.36	70.24	0.96
Herschel	11	18	36	197	15	3	3	5.66	79.60	0.40
Hexahedral	8	12	24	98	8	2	2	4.63	66.36	1.84
Hoffman	16	32	64	578	29	5	4	8.74	85.80	0.16
House	5	6	15	57	9	2	2	2.55	89.96	46.16
Icosahedral	12	30	48	391	27	3	2	5.50	96.36	1.08
K10	10	45	50	581	35	1	1	4.92	100.0	0.04
K2,3	5	6	15	53	6	2	2	2.55	80.24	42.80
K2	2	1	4	5	2	1	1	1.01	99.92	98.76
K2x1	3	2	7	13	2	1	1	1.31	92.44	64.20
K3,3	6	9	18	68	7	2	2	4.77	68.00	2.28
K3,4	7	12	24	119	9	2	2	3.33	95.56	9.72
K3	3	3	9	23	4	1	1	1.32	99.48	71.36
K4,4	8	16	32	190	12	2	2	4.47	98.72	0.64
K4,5	9	20	36	251	16	2	2	4.82	98.84	0.20
K4	4	6	12	39	4	1	1	1.86	99.96	38.48
K5	5	10	20	92	7	1	1	2.41	100.0	9.40
K5x5	10	25	40	324	17	2	2	5.03	99.76	0.12
K5x6	11	30	44	357	17	2	2	5.38	99.76	0.04
K6	6	15	24	117	9	1	1	2.54	96.44	13.40
K6x6	12	36	48	537	25	2	2	4.85	98.44	0.52
K7	7	21	28	181	10	1	1	4.56	12.00	3.44
K8	8	28	32	260	13	1	1	5.24	75.20	25.84
K9	9	36	45	460	25	1	1	4.39	100.0	0.08
Krackhardt	10	18	34	205	15	3	2	5.69	82.20	0.56
Octahedral	6	12	24	124	9	2	2	3.32	98.32	16.32
Pappus	18	27	54	370	15	6	5	9.81	85.08	0.12
Petersen	10	15	30	161	11	3	3	4.96	79.28	1.08
Poussin	15	39	60	585	27	3	3	7.87	98.04	0.04
Q3	8	12	24	97	8	2	2	6.39	14.40	0.52
Q4	16	32	64	578	26	6	4	9.68	96.80	0.32
Robertson	19	38	76	827	39	6	5	10.58	81.48	0.08
S2	3	2	7	13	3	1	1	1.52	95.28	47.16
S3	4	3	9	17	3	1	1	1.57	96.84	56.32
S4	5	4	12	28	4	1	1	2.31	81.84	11.60
S5	6	5	14	40	4	1	1	2.02	94.36	26.12
S6	7	6	16	49	5	1	1	3.16	98.56	4.80
S7	8	7	18	57	5	1	1	3.06	98.72	3.44
S8	9	8	21	80	8	1	1	3.78	53.64	0.76
S9	10	9	23	95	7	1	1	3.94	78.76	1.44
S10	11	10	25	107	10	1	1	4.22	81.36	0.44
Shrikhande	16	48	64	785	37	4	3	7.09	90.56	0.32
Sousselier	16	27	53	390	19	5	4	8.42	64.88	0.24
Tietze	12	18	36	191	10	4	3	5.74	87.84	1.96
Wagner	8	12	24	106	9	3	3	4.97	47.56	0.24

Table 3.6: Results for some small graphs for Edge Cover.

Graph	Order	Size	Logical Qubits	Physical Qubits	Embedding Max Chain	Best Answer	Optimal Answer	Average Valid Answer	Probability of Valid Answer	Probability of Best Answer
BidiakisCube	12	18	42	149	6	6	6	7.98	64.92	3.48
Bull	5	5	10	22	3	3	3	3.24	47.12	36.04
Butterfly	5	6	12	33	7	3	3	3.63	63.16	24.20
C10	10	10	20	55	5	5	5	5.42	89.24	53.88
C11	11	11	22	51	7	6	6	6.48	71.76	39.36
C12	12	12	24	53	5	6	6	6.98	77.52	12.00
C4	4	4	8	13	2	2	2	2.05	98.56	93.64
C5	5	5	10	16	2	3	3	3.01	98.92	97.88
C6	6	6	12	23	3	3	3	3.10	61.84	55.96
C7	7	7	14	31	4	4	4	4.03	70.36	68.16
C8	8	8	16	28	3	4	4	4.21	64.20	51.32
C9	9	9	18	34	3	5	5	5.12	74.40	66.16
Chvatal	12	24	48	227	7	7	6	11.05	69.60	0.96
Clebsch	16	40	88	653	22	13	8	19.02	78.48	0.20
Diamond	4	5	11	25	3	2	2	2.50	92.12	47.56
Dinneen	9	21	42	283	12	5	5	8.77	71.04	0.08
Dodecahedral	20	30	70	245	6	11	10	13.63	47.84	1.64
Durer	12	18	42	143	6	6	6	8.66	74.76	0.60
Errera	17	45	96	633	15	13	9	21.28	80.64	0.04
Frucht	12	18	42	139	6	6	6	8.09	39.12	0.56
GoldnerHarary	11	27	54	411	17	8	6	12.81	46.12	0.04
Grid2x3	6	7	15	33	3	3	3	3.42	90.64	56.68
Grid3x3	9	12	26	73	5	5	5	5.75	74.40	32.48
Grid3x4	12	17	37	113	6	6	6	7.36	58.20	8.52
Grid4x4	16	24	52	191	8	8	8	11.28	69.80	0.44
Grid4x5	20	31	67	265	13	10	10	14.02	47.64	0.08
Grotzsch	11	20	43	207	10	7	6	10.12	53.28	0.76
Heawood	14	21	49	179	8	7	7	9.41	64.88	0.64
Herschel	11	18	40	144	7	6	6	7.75	60.48	6.40
Hexahedral	8	12	28	114	10	4	4	5.74	66.52	4.56
Hoffman	16	32	64	325	11	9	8	12.90	46.52	0.08
House	5	6	13	28	3	3	3	3.15	82.68	70.44
Icosahedral	12	30	66	508	22	8	6	13.53	78.96	0.08
K2,3	5	6	13	28	3	3	3	3.14	77.72	67.24
K3,3	6	9	21	70	5	3	3	4.39	89.92	12.84
K3,4	7	12	26	103	7	4	4	5.00	81.44	23.00
K3	3	3	6	10	2	2	2	2.00	83.20	83.08
K4,4	8	16	32	149	11	4	4	6.46	92.48	4.16
K4,5	9	20	42	251	12	5	5	8.69	62.04	0.04
K4	4	6	14	49	5	2	2	2.32	83.16	58.76
K5,5	10	25	55	417	18	7	5	11.87	83.88	0.20
K5,6	11	30	63	576	22	8	6	12.15	73.36	0.80
K5	5	10	20	87	6	3	3	4.17	86.24	2.32
K6,6	12	36	72	719	31	9	6	15.18	87.12	0.04
K6	6	15	33	187	11	3	3	6.61	86.12	0.08
K7	7	21	42	338	14	4	4	8.65	91.16	0.04
K8	8	28	52	546	19	6	4	11.15	91.72	0.24
K9	9	36	63	801	27	7	5	13.30	95.36	0.08
Krackhardt	10	18	38	172	9	6	5	9.16	46.36	0.44
Octahedral	6	12	24	89	6	3	3	5.20	83.12	1.52
Pappus	18	27	63	224	8	9	9	12.60	50.72	0.08
Petersen	10	15	35	129	7	5	5	6.96	66.88	1.00
Poussin	15	39	82	692	21	12	8	18.42	70.00	0.08
Q3	8	12	28	95	5	4	4	5.42	77.92	8.00
Q4	16	32	64	431	17	8	8	12.81	71.24	0.04
Robertson	19	38	76	443	13	10	10	15.43	52.12	0.04
S2	3	2	3	4	2	2	2	2.00	97.68	97.68
S3	4	3	5	8	2	3	3	3.00	65.12	65.12
S4	5	4	6	14	3	4	4	4.00	10.32	10.32
S5	6	5	8	22	3	5	5	5.00	1.52	1.52
S6	7	6	9	30	5	6	6	6.00	0.84	0.84
S7	8	7	10	39	4	-	7	-	-	-
S8	9	8	11	44	5	8	8	8.00	0.04	0.04
S9	10	9	13	64	6	-	9	-	-	-
S10	11	10	14	77	7	-	10	-	-	-
Shrikhande	16	48	96	864	26	14	8	20.90	88.20	0.04
Sousselier	16	27	60	256	11	10	8	13.93	48.40	0.24
Tietze	12	18	42	146	6	6	6	8.39	50.44	0.20
Wagner	8	12	28	97	7	4	4	5.44	54.60	5.44

Chapter 4

The Graph Isomorphism Problem

The Graph Isomorphism Problem is the computational problem of determining whether two finite graphs are isomorphic. The problem is one of very few problems in NP that is neither known to be solvable in polynomial time nor NP-complete. Moreover, it is the only problem listed in [25] which remains still unsolved.

The problem can be solved in polynomial time for many special classes of graphs and in practice the Graph Isomorphism Problem can often be solved efficiently, see [40]. L. Babai posted a paper [4] showing that the Graph Isomorphism Problem can be solved in quasi-polynomial $\exp((\lg n)^{O(1)})$ time. These mathematical facts suggest that the Graph Isomorphism Problem has an intermediate complexity, hence a good chance to be solved efficiently using D-Wave.

If the graphs have different sizes or orders, then they cannot be isomorphic and these cases can be decided quickly. So in what follows we will assume that the input to the Graph Isomorphism Problem are two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with the same order and the same size. If the graphs are isomorphic, then the output is a bijective edge-invariant vertex mapping $f : V_1 \to V_2$.

Formally the problem can be stated as follows:

Graph Isomorphism Problem:

Instance:	Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $\lvert V_1\rvert = \lvert V_2\rvert$ and $\lvert E_1\rvert = \lvert E_2\rvert$.
Question:	Determine whether there exists a bijective edge-invariant vertex mapping (isomorphism) $f : V_1 \to V_2$.

The required mapping f is a permutation of vertices in V_1. To represent any of the $n!$ permutations we only need $\min\{k \mid 2^k \geq n!\} = \lceil \lg(n!) \rceil$ bits, that is about $n\lceil \lg n \rceil$ bits. A QUBO formulation of the problem with this theoretical lower bound seems difficult to be realized as mentioned in Section 2.3.

We will provide several different QUBO formulations of the Graph Isomorphism Problem obtained through different methods and compare their efficiency in terms of the number of variables required and the densities of the QUBO graphs. We will also provide formulation for two variations of the Graph Isomorphism Problem.

4.1 Simple approach using Integer Programming

In this section we present a simple formulation (i.e. a polynomial-time reduction) of the Graph Isomorphism Problem to an IP Problem.

Recall the input consists of two graphs G_1 and G_2 with both being of order n and size m. We use the following $n + 2m$ integer variables:

- v_i, $0 \leq i < n$, denotes the permutation from vertices of G_1 to G_2,
- x_k, $0 \leq k < 2m$, denotes the bijection from edges of G_1 to G_2.

For both graphs, we rank the m edges with two different values each from 0 to $2m-1$, by considering the pairs of integers (i, j) and (j, i) as two equivalent representations of a possible edge ij. We say that $\text{rank}(a, b) < \text{rank}(c, d)$ if $na + b < nc + d$. That is, the edges are ranked by considering their index within the (row-wise flattened) adjacency matrix representation of a graph. Let E^* denote this double set of $2m$ ordered pairs obtained from a set of unordered edges E.

A dummy objective function for our optimization problem is $\min v_0$, which indicates that the first vertex of G_1 is mapped to the smallest indexed vertex of G_2, if at least one isomorphism exits.

The integer programming constraints given below justify the conditions of an isomorphism between G_1 and G_2.

First, every vertex of G_1 is mapped to a vertex of G_2, zero indexed:

$$0 \leq v_i < n, \text{ for all } 0 \leq i < n. \tag{4.1}$$

Next, every vertex of G_1 is mapped to a different vertex of G_2:

$$(v_i - v_j)^2 > 0, \text{ for all } 0 \leq i < j < n. \tag{4.2}$$

Each edge ij of E_1 needs to be mapped to the correct two indices in E_2^* with respect to the given v_i variables:

$$nv_i + v_j = x_k, \text{ for } i \neq j, (i, j) \in E_1^* \text{ and } \text{rank}(i, j) = k. \tag{4.3}$$

Note that constraints (4.1)–(4.3) ensure that $1 \leq x_k \leq n^2 - 2$, which are the possible indices into the flattened adjacency matrix of G_2. These three sets of constraints also imply that $x_k \neq x_{k'}$ for all $k \neq k'$.

Next we check that the bijection, given by the map $i \mapsto v_i$, is edge-invariant. Let the pre-computed integer constant y_l, $0 \leq l < 2m$, be the edge encoding $y_l = na + b$ for $(a, b) \in E_2^*$ with $\mathrm{rank}(a, b) = l$:

$$\Pi_{y_l \in E_2^*}(x_k - y_l) = 0, \text{ for all } x_k. \tag{4.4}$$

The constraints given in (4.4) ensure that each edge of G_1 is mapped to an edge in G_2. Since x_k acts as an injective function and both input graphs have the same size m, the function is also surjective, so we do not need to explicitly check that non-edges map to non-edges.

To convert the IP to one with only linear binary constraints, we use standard conversion techniques (see [10]): a) $\lceil \lg n \rceil$ binary variables to represent each variable v_i and $\lceil \lg(n^2 - 2) \rceil$ binary variables to represent each variable x_k. b) each product xy of binary variables is replaced with a new binary variable z and two linear constraints involving x, y and z. Lastly, we need to convert the final binary linear IP to a standard form (equality constraints only) by introducing slack variables.

The final step is to build an equivalent QUBO matrix Q from the IP formulation. Here consider each linear equation constraint C_k of the form $\sum_{i=1}^{n} c_{(k,i)} x_i = d_k$ for $x_i \in \{0, 1\}$ with fixed integer constants $c_{(k,i)}$ and d_k. This equation is satisfied if and only if $\sum_{i=1}^{n} c_{(k,i)} x_i - d_k = 0$, or equivalently, the optimal solution of $x^* = \min_{\mathbf{x}} C'_k(\mathbf{x})$ is 0 where $C'_k(\mathbf{x}) = (\sum_{i=1}^{n} c_{(k,i)} x_i - d_k)^2$. We (symbolically) add all these quadratic expressions $C'_k(\mathbf{x})$ together, combining coefficients for any same terms $x_i x_j$, to get a final QUBO objective function $\mathbf{x}^T Q \mathbf{x}$. Note that the coefficients of the linear terms $x_i = x_i x_i$ correspond to the diagonal entries of Q and we can safely ignore any constant terms, which have no impact on the selection of the best assignment of the binary variables $\mathbf{x} = (x_0, x_2, \ldots, x_{n-1})$ as explained in Section 2.4.

Theorem 5. *The IP approach for generating a QUBO formulation for the Graph Isomorphism Problem described above requires $O(m^3 \lg n)$ qubits.*

Proof. We make a tally of the number of integer variables and their integer range to determine the number of binary variables needed. For the n variables v_i we need $n \lg n$ binary variables. For the possible n^2 products $v_i v_j$ we need $2n^2 \lg n$ binary variables. For the $2m$ variables x_k we need $4m \lg n$ binary variables. Further, for the products of constraints (4.4), we need to represent all powers x_k^j for $1 \leq j \leq 2m$, which increases the number of binary variables is slightly over $8m^3 \lg n$. This is because we have to choose all combinations of selecting products of all the $\lg n$ binary variables for each x_k, which is approximately $\sum_{i=2}^{2m} (\sqrt{2} \lg n + i)^2$. Following the standard reduction, see [9], we need at most $\lg n$ binary slack variables for constraints of type (4.1) and at most $2 \lg n$ binary slack variables for constraints of type (4.2). The other constraints are already in equality form. Thus the total number of binary variables is $O(m^3 \lg n)$. $\square$

4.1.1 An example: the graph P_3

Consider the path graph P_3 of order 3 and two copies represented as G_1 with edges $E_1 = \{\{0,1\},\{1,2\}\}$ and G_2 with edges $E_2 = \{\{0,1\},\{0,2\}\}$. It is easy to see there are two possible isomorphisms between G_1 and G_2, where we require vertex 1 of G_1 to be mapped to vertex 0 of G_2. With variables x_1, x_2, x_3, and x_4 from the ranked edges in E_1^* and constants $y_1 = 1$, $y_2 = 2$, $y_3 = 3$ and $y_4 = 6$ from E_2^* we have the following integer programming constraints.

$$0 \le v_0 \le 2, \quad 0 \le v_1 \le 2, \quad 0 \le v_2 \le 2,$$

$$1 \le (v_0 - v_1)^2 \le 4, \quad 1 \le (v_0 - v_2)^2 \le 4, \quad 1 \le (v_1 - v_2)^2 \le 4,$$

$$3v_0 + v_1 - x_0 = 0, \quad 3v_1 + v_0 - x_1 = 0, \quad 3v_1 + v_2 - x_2 = 0, \quad 3v_2 + v_1 - x_3 = 0,$$

$$\begin{aligned}(x_0-1)(x_0-2)(x_0-3)(x_0-6) = 0, \quad (x_1-1)(x_1-2)(x_1-3)(x_1-6) = 0,\\ (x_2-1)(x_2-2)(x_2-3)(x_2-6) = 0, \quad (x_3-1)(x_3-2)(x_3-3)(x_3-6) = 0.\end{aligned}$$

Converting to binary variables and adding slack variables we have the following constraints:

$$\begin{gathered}
2v_{0,1} + v_{0,0} + 2s_{0,1} + s_{0,0} = 2,\\
2v_{1,1} + v_{1,0} + 2s_{1,1} + s_{1,0} = 2,\\
2v_{2,1} + v_{2,0} + 2s_{2,1} + s_{2,0} = 2,\\
-(2v_{0,1} + v_{0,0} - 2v_{1,1} - v_{1,0})^2 + 2s_{3,1} + s_{3,0} = -1,\\
-(2v_{0,1} + v_{0,0} - 2v_{2,1} - v_{2,0})^2 + 2s_{4,1} + s_{4,0} = -1,\\
-(2v_{1,1} + v_{1,0} - 2v_{2,1} - v_{2,0})^2 + 2s_{5,1} + s_{5,0} = -1,\\
6v_{0,1} + 3v_{0,0} + 2v_{1,1} + v_{1,0} - 4x_{0,2} - 2x_{0,1} - x_{0,0} = 0,\\
6v_{1,1} + 3v_{1,0} + 2v_{0,1} + v_{0,0} - 4x_{1,2} - 2x_{1,1} - x_{1,0} = 0,\\
6v_{1,1} + 3v_{1,0} + 2v_{2,1} + v_{2,0} - 4x_{2,2} - 2x_{2,1} - x_{2,0} = 0,\\
6v_{2,1} + 3v_{2,0} + 2v_{1,1} + v_{1,0} - 4x_{3,2} - 2x_{3,1} - x_{3,0} = 0,\\
(4x_{0,2}+2x_{0,1}+x_{0,0}-1)(4x_{0,2}+2x_{0,1}+x_{0,0}-2)(4x_{0,2}+2x_{0,1}+x_{0,0}-3)(4x_{0,2}+2x_{0,1}+x_{0,0}-6)=0,\\
(4x_{1,2}+2x_{1,1}+x_{1,0}-1)(4x_{1,2}+2x_{1,1}+x_{1,0}-2)(4x_{1,2}+2x_{1,1}+x_{1,0}-3)(4x_{1,2}+2x_{1,1}+x_{1,0}-6)=0,\\
(4x_{2,2}+2x_{2,1}+x_{2,0}-1)(4x_{2,2}+2x_{2,1}+x_{2,0}-2)(4x_{2,2}+2x_{2,1}+x_{2,0}-3)(4x_{2,2}+2x_{2,1}+x_{2,0}-6)=0,\\
(4x_{3,2}+2x_{3,1}+x_{3,0}-1)(4x_{3,2}+2x_{3,1}+x_{3,0}-2)(4x_{3,2}+2x_{3,1}+x_{3,0}-3)(4x_{3,2}+2x_{3,1}+x_{3,0}-6)=0.
\end{gathered}$$

Here we added 6 additional binary slack variables (labeled $s_{i,j}$) for constraints of type (4.1) and 6 additional for constraints of type (4.2). Finally, to convert to linear constraints we need to add $(n\lceil \lg(n-1)\rceil)^2 = 6^2 = 36$ additional binary variables for those of type (4.2) and $2m\sum_{i=2}^{2m}\binom{\lceil \lg(n^2-2)+i-1\rceil}{i} = 4(6+10+15) = 124$ additional binary variables for those of type (4.4). Thus, 6+12+6+6+36+124=190 total qubits required for the final QUBO matrix for this input.

4.2 A direct QUBO formulation

We present an improved QUBO objective function F for the Graph Isomorphism Problem in this subsection. After we have developed this formulation, we noted that A. Lucas in his paper [37] had an Ising formulation that is similar to what we have developed. However, the Graph Isomorphism Problem studied in [37] does not make the explicit assumption that G_1 and G_2 have the same number of edges. As a result, the direct formulation we provide here is of a simpler form and has a lower time complexity (if we consider the formulation a reduction to QUBO) in some cases.

The formulation requires only n^2 binary variables represented by a binary vector $\mathbf{x} \in \mathbb{Z}_2^{n^2}$:

$$\mathbf{x} = (x_{0,0}, x_{0,1}, \dots, x_{0,n-1}, x_{1,0}, x_{1,1}, \dots, x_{1,n-1}, \dots, x_{n-1,0}, \dots, x_{n-1,n-1}).$$

The equality $x_{i,i'} = 1$ encodes the property that the function f maps the vertex v_i in G_1 to the vertex $v_{i'}$ in G_2:$f(v_i) = v_{i'}$. For this mapping we need to pre-compute n^2 binary constants $e_{i,j}$ for $0 \leq i < n$ and $0 \leq j < n$ such that

$$e_{i,j} = \begin{cases} 1 & \text{if } ij \in E_2 \\ 0 & \text{otherwise.} \end{cases}$$

The function F consists of two parts, $H(\mathbf{x})$ and $\sum_{ij \in E_1} P_{i,j}(\mathbf{x})$. Each part serves as a penalty for the case when the function f is not an isomorphism. The first part H ensures that f is a bijective function: $H = 0$ if and only if the function f encoded by the vector $\mathbf{x}$ is a bijection. The second term ensures that f is edge-invariant: $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) > 0$ if and only if there exists an edge $uv \in E_1$ such that $f(u)f(v) \notin E_2$.

The objective function $F(\mathbf{x})$ has the following form:

$$F(\mathbf{x}) = H(\mathbf{x}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}), \tag{4.5}$$

where

$$H(\mathbf{x}) = \sum_{0 \leq i < n} \left(1 - \sum_{0 \leq i' < n} x_{i,i'}\right)^2 + \sum_{0 \leq i' < n} \left(1 - \sum_{0 \leq i < n} x_{i,i'}\right)^2, \tag{4.6}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{0 \leq i' < n} \left(x_{i,i'} \sum_{0 \leq j' < n} x_{j,j'}(1 - e_{i',j'}) \right). \tag{4.7}$$

Assume that $x^* = \min_{\mathbf{x}} F(\mathbf{x})$. Decoder function D will be used to obtain the vertex mapping f. Let $\mathcal{F}$ be the set of all bijections between V_1 and V_2. Then $D : \mathbb{Z}_2^{n^2} \to \mathcal{F}$ is a partial function that re-constructs the vertex mapping f from the vector $\mathbf{x}$, if such f exists. The domain of D contains all vectors $\mathbf{x} \in \mathbb{Z}_2^{n^2}$ that can be decoded into a bijective function f:

$$\operatorname{dom}(D) = \left\{ \mathbf{x} \in \mathbb{Z}_2^{n^2} \;\middle|\; \sum_{0 \le i' < n} x_{i,i'} = 1, \text{ for all } 0 \le i < n \right. \\ \left. \text{and} \sum_{0 \le i < n} x_{i,i'} = 1, \text{ for all } 0 \le i' < n \right\},$$

and

$$D(\mathbf{x}) = \begin{cases} f, & \text{if } \mathbf{x} \in \operatorname{dom}(D), \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

where $f : V_1 \to V_2$ is a bijection such that $f(v_i) = v_{i'}$ if and only if $x_{i,i'} = 1$.

Recall that $I(v)$ denote the set of edges incident to the vertex v. The term $x_{i,i'}x_{j,j'}$ in the right-hand side of (4.7) has a positive coefficient if and only if $i'j' \notin I(v_{i'})$, hence an equivalent, simpler definition of $P_{i,j}(\mathbf{x})$ in (4.7) can be given without $e_{i',j'}$ as follows:

$$P_{i,j}(\mathbf{x}) = \sum_{0 \le i' < n} \left(x_{i,i'} \sum_{i'j' \notin I(v_{i'})} x_{j,j'} \right). \tag{4.8}$$

The following two lemmata will be used to prove correctness of the objective function F in (4.5).

Lemma 6. *For every $\mathbf{x} \in \mathbb{Z}_2^{n^2}$, $H(\mathbf{x}) = 0$ if and only if $D(\mathbf{x})$ is defined (in this case $D(\mathbf{x})$ is a bijection).*

Proof. Fix $\mathbf{x} \in \mathbb{Z}_2^{n^2}$. The term $H(\mathbf{x})$ has two components,

$$\sum_{0 \le i < n} \left(1 - \sum_{0 \le i' < n} x_{i,i'} \right)^2 \text{ and } \sum_{0 \le i' < n} \left(1 - \sum_{0 \le i < n} x_{i,i'} \right)^2.$$

Since both components consist of only quadratic terms, we have $H(\mathbf{x}) = 0$ if and only if both components are equal to 0.

First,

$$\sum_{0 \le i < n} \left(1 - \sum_{0 \le i' < n} x_{i,i'} \right)^2 = 0 \tag{4.9}$$

if and only if for each $0 \le i < n$, exactly one variable in the set $\{x_{i,i'} \mid 0 \le i' < n\}$ has value 1, that is, every vertex $v \in V_1$ has an image.

Second, with the same argument,

$$\sum_{0\le i'<n}\left(1-\sum_{0\le i<n} x_{i,i'}\right)^2 = 0 \tag{4.10}$$

if and only if for each $0 \le i' < n$, exactly one variable in the set $\{x_{i,i'} \mid 0 \le i < n\}$ has value 1, hence the function $v_i \mapsto v_{i'}$ is surjective.

Together the conditions (4.9) and (4.10) are equivalent with the property that every vertex $v_i \in V_1$ is mapped to a unique vertex $v_{i'} \in V_2$, and since the orders of G_1 and G_2 are same, the mapping $v_i \mapsto v_{i'}$ is bijective. □

The second lemma stated below ensures that the mapping f, if bijective, is also edge-invariant.

Lemma 7. *Let $\mathbf{x} \in \mathbb{Z}_2^{n^2}$ and assume that $D(\mathbf{x})$ is a bijective function. Then, $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) = 0$ if and only if the mapping $f = D(\mathbf{x})$ is edge-invariant.*

Proof. Fix $\mathbf{x} \in \mathbb{Z}_2^{n^2}$. Note that $P_{i,j}(\mathbf{x})$ from (4.7) does not contain cubic terms, so, as all $e_{i',j'}$ are constants, $P_{i,j}(\mathbf{x})$ contains only quadratic terms (see also (4.8)); consequently, $P_{i,j}(\mathbf{x}) \ge 0$, for all $ij \in E_1$. Furthermore, $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) = 0$ if and only if $P_{i,j}(\mathbf{x}) = 0$, for all $ij \in E_1$.

After expanding the left hand side of equation (4.7), we get

$$P_{i,j}(\mathbf{x}) = \sum_{0\le i'<n} x_{i,i'}\left(x_{j,0}(1-e_{i',0}) + x_{j,1}(1-e_{i',1}) + \cdots + x_{j,n-1}(1-e_{i',n-1})\right).$$

Since f is a bijection, for every edge $ij \in E_1$, in the set $\{x_{i,i'} \mid 0 \le i' < n\}$ there is a unique variable, denoted by $x^*_{i,i'}$, with value 1, and in the set $\{x_{j,j'} \mid 0 \le j' < n\}$ there is exactly one variable, denoted by $x^*_{j,j'}$, with value 1.

Assume that $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) \ne 0$, i.e. for some $ij \in E_1$ we have $P_{i,j}(\mathbf{x}) \ne 0$. It is easy to see that $P_{i,j}(\mathbf{x}) \ne 0$ if and only if $x^*_{i,i'}x^*_{j,j'}(1-e_{i',j'}) \ne 0$, or equivalently, $e_{i',j'} = 0$. The last equality violates the condition of an edge-invariant mapping as $e_{i',j'} = 0$ implies that there is no edge between the vertices $v_{i'}$ and $v_{j'}$ in G_2.

Conversely, if $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) = 0$, then $P_{i,j}(\mathbf{x}) = 0$ for all $ij \in E_1$, hence $x^*_{i,i'}x^*_{j,j'}(1-e_{i',j'}) = 0$ which implies $e_{i',j'} = 1$. This means that for all $ij \in E_1$, $f(i)f(j) \in E_2$. Since f is bijective and $|E_1| = |E_2|$, every edge $ij \in E_2$ must also have an corresponding edge $f^{-1}(i)f^{-1}(j) \in E_1$, so f is edge-invariant. □

Using Lemmata 6 and 7 we now prove the main result of the section.

Theorem 8. *For every $\mathbf{x} \in \mathbb{Z}_2^{n^2}$, $F(\mathbf{x}) = 0$ if and only if the mapping $f : V_1 \to V_2$ defined by $f = D(\mathbf{x})$ is an isomorphism.*

Proof. Since both $H(\mathbf{x})$ and $\sum_{ij\in E_1} P_{i,j}(\mathbf{x})$ contain only quadratic terms, we have $F(\mathbf{x}) = 0$ if and only if both $H(\mathbf{x}) = 0$ and $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) = 0$.

Assume $F(\mathbf{x}) = 0$. Then by Lemmas 6 and 7, f must be bijective and edge-invariant. On the other hand, if $F(\mathbf{x}) \neq 0$, then we have either $H(\mathbf{x}) \neq 0$ or $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) \neq 0$. If $H(\mathbf{x}) \neq 0$, then f is not bijective by Lemma 6. If $H(\mathbf{x}) = 0$ and $\sum_{ij\in E_1} P_{i,j}(\mathbf{x}) \neq 0$, then by Lemma 7 the mapping is not edge-invariant. □

4.2.1 An example: the graph P_3 revisited

We use the same instances of G_1 and G_2 as described in Section 4.1.1. The direct QUBO formulation requires $3^2 = 9$ variables and the binary variable vector $\mathbf{x} \in \mathbb{Z}_2^9$ is:

$$\mathbf{x} = (x_{0,0}, x_{0,1}, x_{0,2}, x_{1,0}, x_{1,1}, x_{1,2}, x_{2,0}, x_{2,1}, x_{2,2}).$$

By expanding equation (4.6), we have the following penalty terms:

$$\begin{aligned} H(\mathbf{x}) = &\left(1 - (x_{0,0} + x_{0,1} + x_{0,2})\right)^2 + \left(1 - (x_{1,0} + x_{1,1} + x_{1,2})\right)^2 \\ &+ \left(1 - (x_{2,0} + x_{2,1} + x_{2,2})\right)^2 + \left(1 - (x_{0,0} + x_{1,0} + x_{2,0})\right)^2 \\ &+ \left(1 - (x_{0,1} + x_{1,1} + x_{2,1})\right)^2 + \left(1 - (x_{0,2} + x_{1,2} + x_{2,2})\right)^2. \end{aligned}$$

Using definition of $P_{i,j}$ given by equation (4.7) we need to pre-compute the following binary constants $e_{i,j}$: $e_{0,0} = 0, e_{0,1} = 1, e_{0,2} = 1, e_{1,0} = 1, e_{1,1} = 0, e_{1,2} = 0, e_{2,0} = 1, e_{2,1} = 0, e_{2,2} = 0$. By substituting the variables and the constants $e_{i,j}$ in equation (4.7), we obtain the following penalty terms:

$$\begin{aligned} P_{0,1} &= x_{0,0}x_{1,0} + x_{0,1}(x_{1,1} + x_{1,2}) + x_{0,2}(x_{1,1} + x_{1,2}), \\ P_{1,2} &= x_{1,0}x_{2,0} + x_{1,1}(x_{2,1} + x_{2,2}) + x_{1,2}(x_{2,1} + x_{2,2}). \end{aligned}$$

Once again, we need to process some of the penalty terms before we can encode them in a QUBO instance. Take the first penalty term $(1-(x_{0,0}+x_{0,1}+x_{0,2}))^2$ for example. After expanding the brackets, we get

$$\begin{aligned} (1 - (x_{0,0} + x_{0,1} + x_{0,2}))^2 = 1 &- (2x_{0,0} + 2x_{0,1} + 2x_{0,2}) + x_{0,0}^2 + x_{0,0}x_{0,1} \\ &+ x_{0,0}x_{0,2} + x_{0,1}x_{0,0} + x_{0,1}^2 + x_{0,1}x_{0,2} + x_{0,2}x_{0,0} + x_{0,2}x_{0,1} + x_{0,2}^2. \end{aligned}$$

We have one constant term as well as three linear terms in the penalty term above. As described in Section 2.4, first the constant term 1 can be ignored. Second, all variables $x_{i,i'}$ will be replaced by $x_{i,i'}^2$. After summing up the new terms, we get the final penalty term that will be encoded into the QUBO instance:

$$-x_{0,0}^2 - x_{0,1}^2 - x_{0,2}^2 + 2x_{0,0}x_{0,1} + 2x_{0,0}x_{0,2} + 2x_{0,1}x_{0,2}.$$

The process has to be applied to all penalty terms in $F(\mathbf{x})$ to finally obtain an objective function $F(\mathbf{x})$ that has quadratic only terms, so it can be encoded into a QUBO instance. In our example we obtain a 9×9 QUBO matrix Q.

For all elements $x_{i,i'}$ in $\mathbf{x}$ we map the variable $x_{i,i'}$ to the index $d(x_{i,i'}) = 3i + i' + 1$ (between 1 and 9) and then the entry $Q_{(d(x_{i,i'}),d(x_{j,j'}))}$ is assigned the coefficient of the term $x_{i,i'}x_{j,j'}$ in $F(\mathbf{x})$. As for each pair $x_{i,i'}$ and $x_{j,j'}$ there are two possible equivalent terms, $x_{i,i'}x_{j,j'}$ and $x_{j,j'}x_{i,i'}$, as a convention, we will use the term that is be mapped to the upper-triangle part of Q. That is, we use $x_{i,i'}x_{j,j'}$ if $d(x_{i,i'}) \leq d(x_{j,j'})$ and $x_{j,j'}x_{i,i'}$ otherwise. The upper-triangular matrix representation of Q is shown in Table 4.1.

Table 4.1: QUBO matrix for P_3

variables	$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$
$x_{0,0}$	-2	2	2	3	0	0	2	0	0
$x_{0,1}$		-2	2	0	3	1	0	2	0
$x_{0,2}$			-2	0	1	3	0	0	2
$x_{1,0}$				-2	2	2	3	0	0
$x_{1,1}$					-2	2	0	3	1
$x_{1,2}$						-2	0	1	3
$x_{2,0}$							-2	2	2
$x_{2,1}$								-2	2
$x_{2,2}$									-2

Note that the removal of any constants will introduce an offset to the value of the objective function as mentioned in Section 2.4. In our case, we have removed a constant value of 6 from $F(\mathbf{x})$ when encoding it into Q. As a result, the optimal solution of $f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x}$ will now have a value of -6.

As mentioned in Section 4.1, the vertex 1 in G_1 is mapped to vertex 0 in G_2, hence we have the following two optimal solutions:

$$\mathbf{x}_1 = (0, 1, 0, 1, 0, 0, 0, 0, 1) \text{ and } \mathbf{x}_2 = (0, 0, 1, 1, 0, 0, 0, 1, 0).$$

An example: the graph C_4

In this section we present a different example. A cycle graph C_n of order n is a graph that consists of a single cycle of length n.

Definition 22. *A* ***cycle graph*** *with n vertices, denoted by C_n, has $V = \{0, 1, \cdots, n-1\}$ and $E = \{\{i, i+1\} \mid 0 \leq i < n-1\} \cup \{\{0, n-1\}\}$.*

Let $G_1 = G_2 = C_4$. After obtaining the objective function F and applying the procedure described in Section 2.4 again, we get 16 variables. Each variable $x_{i,i'}$ will be mapped to the index $4i + i' + 1$. The coefficients of each quadratic term $x_{i,i'}x_{j,j'}$ in $F(\mathbf{x})$ are then computed and the entry $Q_{(4i+i'+1,4j+j'+1)}$ is set to that coefficient. The complete QUBO matrix is shown in Table 4.2.

If we consider the vertex mapping f as a permutation of the vertices in V_1 and the sequence $0, 1, 2, 3$ as a cycle in G_1, then the sequence of vertices in the permutation

Table 4.2: QUBO matrix for C_4

variables	$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{3,0}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$
$x_{0,0}$	-2	2	2	2	3	0	1	0	2	0	0	0	3	0	1	0
$x_{0,1}$		-2	2	2	0	3	0	1	0	2	0	0	0	3	0	1
$x_{0,2}$			-2	2	1	0	3	0	0	0	2	0	1	0	3	0
$x_{0,3}$				-2	0	1	0	3	0	0	0	2	0	1	0	3
$x_{1,0}$					-2	2	2	2	3	0	1	0	2	0	0	0
$x_{1,1}$						-2	2	2	0	3	0	1	0	2	0	0
$x_{1,2}$							-2	2	1	0	3	0	0	0	2	0
$x_{1,3}$								-2	0	1	0	3	0	0	0	2
$x_{2,0}$									-2	2	2	2	3	0	1	0
$x_{2,1}$										-2	2	2	0	3	0	1
$x_{2,2}$											-2	2	1	0	3	0
$x_{2,3}$												-2	0	1	0	3
$x_{3,0}$													-2	2	2	2
$x_{3,1}$														-2	2	2
$x_{3,2}$															-2	2
$x_{3,3}$																-2

corresponds to a cycle in G_2. There are eight different cycles in G_2. A cycle can start at any of the four vertices, and after fixing the starting vertex of the cycle, the path can take on two different orientations. Using this ideas, we find the eight optimal solutions of the equation $x^* = \min_{\mathbf{x}} F(\mathbf{x})$:

$$\mathbf{x}_1 = (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1), \quad \mathbf{x}_2 = (1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0),$$

$$\mathbf{x}_3 = (0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0), \quad \mathbf{x}_4 = (0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0),$$
$$\mathbf{x}_5 = (0,0,1,0,0,0,0,1,1,0,0,0,0,1,0,0), \quad \mathbf{x}_6 = (0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1),$$
$$\mathbf{x}_7 = (0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0), \quad \mathbf{x}_8 = (0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0),$$

each of which gives an isomorphism between G_1 and G_2.

4.3 Formulation via reduction to Clique

In this section we give an alternate QUBO formulation that requires n^2 binary variables for the Graph Isomorphism Problem. The construction is based on a known polynomial-time reduction from the Graph Isomorphism Problem to the Clique Problem [6]. Then later in this section we provide an optimal QUBO formulation of the Clique Problem to complete the formulation.

Definition 23. *Let $G = (V, E)$ be graph, a* ***clique*** *of G is a subgraph induced by a subset of vertices $V' \subseteq V$ such that for all $\{a, b\} \subseteq V'$ we have $ab \in E$.*

Clique Problem:

Instance: Graph $G = (V, E)$ and integer k.
Question: Is there a clique $V' \subseteq V$ of size k?

The Clique Problem is one of the original Karp's 21 NP-complete problems [33]. The maximum clique of size k of a graph is called the *clique number*, denoted by $\chi(G)$. For our construction we use some ideas developed in [45].

Definition 24. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The associated **graph product** $\Psi(G_1, G_2) = (V, E)$ is a graph vertices $V = V_1 \times V_2$ and edges $E = \{((a, b), (c, d)) \in V \times V \mid a \neq c, b \neq d$ and $ac \in E_1 \Leftrightarrow bd \in E_2\}$*

Theorem 9. *Two graphs G_1 and G_2 with $|V_1| = |V_2| = n$ are isomorphic if and only if $\chi(\Psi(G_1, G_2)) = n$.*

Proof. Let us first consider the case where G_1 and G_2 are isomorphic via a bijection $f : V_1 \to V_2$. Then we claim that the subset $V' = \{(i, f(i)) \mid i \in V_1\} \subseteq V$ is a clique in $\Psi(G_1, G_2)$. Since V' has n vertices, we just need to check there is an edge between any pair of vertices. Consider a pair (a, b) and (c, d) in V' with $a \neq c$. Since f is a bijection we have $b \neq d$ when $b = f(a)$ and $d = f(c)$. Since any isomorphism is edge-invariant, we also have $ac \in E_1 \Leftrightarrow bd \in E_2$, which is preserved by the definition of E for $\Psi(G_1, G_2)$.

For the other direction, we assume there is clique V' of order n in $\Psi(G_1, G_2)$ and we extract an isomorphism f from this set. First note that for any distinct pair of vertices (a, b) and (c, d) in V', we have $a \neq c$ and $b \neq d$ since otherwise, we would not have an edge between them in the clique. Thus, with exactly n pairs of vertices (u, v) with $u \in V_1$ and $v \in V_2$, we have a well-defined bijective function f from V_1 to V_2 defined by $b = f(a)$ for $(a, b) \in V'$. For f to be an isomorphism we need $ac \in E_1 \Leftrightarrow f(a)f(c) \in E_2$. Again, since V' was assumed to be a clique and, for $a \neq c$, there exists an edge between $(a, f(a))$ and $(c, f(c))$ so using the definition of E we must have $ac \in E_1 \Leftrightarrow f(a)f(c) \in E_2$. □

Next we give a simple construction of an optimal QUBO matrix for the Clique Problem. For a graph $G = (V, E)$ of order n we build an upper-triangle matrix Q of dimension n where

$$Q_{(i,j)} = \begin{cases} -1, & \text{if } i = j, \\ 0, & \text{if } i < j \text{ and } ij \in E, \\ 2, & \text{if } i < j \text{ and } ij \notin E. \end{cases}$$

Theorem 10. *For every graph G, the minimum value of the QUBO objective function $f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$ is $-\chi(G)$; in this case the set of variables of $\mathbf{x}$ with value 1 correspond to a maximum clique.*

Proof. First, let V' be a maximum clique of G and set $x_i = 1$ if $i \in V'$, otherwise $x_i = 0$. The sum $\sum_{i \leq j} x_i Q_{(i,j)} x_j$ has $\chi(G)$ terms with value -1 whenever $i = j$ and

$x_i = 1$. All other terms will be 0 since, by assumption, if both x_i and x_j are 1 then there is an edge between i and j in V' and the corresponding entries $Q_{(i,j)}$ are defined to be 0. In other cases, one of x_i or x_j is 0, so it does not matter what value is set for $Q_{(i,j)}$. Hence the sum $f(\mathbf{x})$ totals to $-\chi(G)$ for any maximum clique.

Now let us assume x^* is an optimal minimum value of the objective $f(\mathbf{x})$ for some assignment of $\mathbf{x}$ that does not correspond to a clique V' of G. Let i and j be two vertices such that $ij \notin E$ but $x_i = x_j = 1$. The sum $\sum_{i \le j} x_i Q_{(i,j)} x_j$ has at least one term with value 2. If we slightly change $\mathbf{x}$, say setting $x_i = 0$, the sum will decrease by 2 for that term (and possibly more for other non-edges involving i) and increase by 1 for the diagonal term $x_i Q_{(i,i)} x_i$. This global decrease with at least 1 which contradicts the minimality of x^*. Finally, if the clique V' is not as large as possible, then x^* is also not optimal, so the theorem is proved. □

4.3.1 Example: the graph P_3 revisited

We use the same instances of G_1 and G_2 as described in Section 4.1.1. The associated product graph with nine vertices is given in Figure 4.1. We can see there are two cliques of size 3, which correspond to the two possible isomorphisms of G_1 and G_2. The shared vertex '1,0' (of the two 3-cliques) indicates that both of these two isomorphisms require vertex 1 to be mapped to vertex 0.

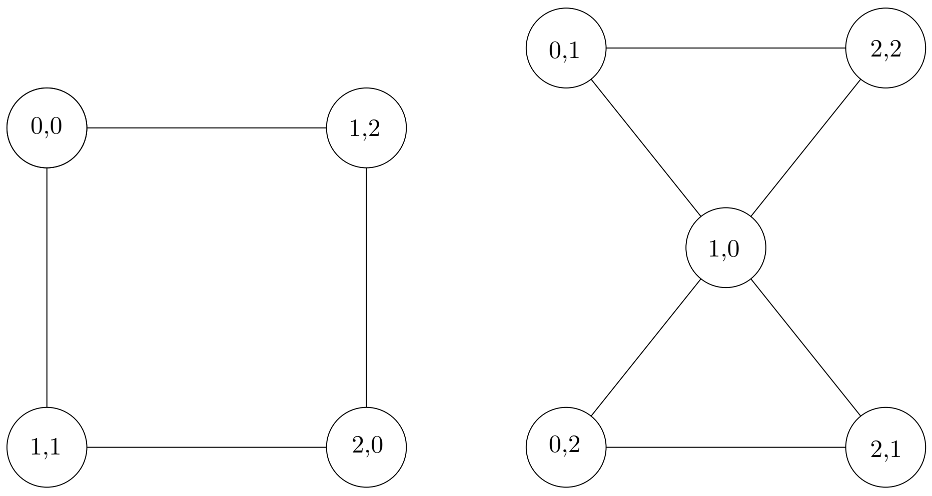

Figure 4.1: The graph $\Psi(G_1, G_2)$.

The QUBO matrix for the Clique Problem applied to this graph $\Psi(G_1, G_2)$ is given in Table 4.3.

Note that for this particular input P_3, this clique-based n^2 formulation has slightly more non-zero entries than the direct formulation given in Table 4.1. Thus, embedding the QUBO instance on the D-Wave architecture may require more physical qubits even though the number of logical qubits are the same. As we mentioned in

Table 4.3: The alternative QUBO matrix for P_3

vertices	0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2
$0,0$	-1	2	2	2	0	0	2	2	2
$0,1$		-1	2	0	2	2	2	2	0
$0,2$			-1	0	2	2	2	0	2
$1,0$				-1	2	2	2	0	0
$1,1$					-1	2	0	2	2
$1,2$						-1	0	2	2
$2,0$							-1	2	2
$2,1$								-1	2
$2,2$									-1

Section 2.3, we will always be restricted by the number of physical qubits in the hardware, hence a formulation that requires less physical qubits to embed is advantageous here from a practical point of view.

4.4 Minor embedding comparison

Motivated by the observation we made on the P_3 example in Section 4.2.1 and 4.3.1, and the limitation on the number of available physical qubits, we ran several experiments to investigate the difference on hardware embeddings between the clique and direct approach. Most of the relevant test cases are too big to be embedded on the actual D-Wave 2X chipset; since D-Wave is planning on upscaling their current hardware design in the future [14], we decided to pick two large Chimera graphs to test the embeddings on, hoping the results will provide some insight on how well the different formulations will perform with future machines.

The first host graph is a Chimera graph with 7200 vertices and 21360 edges ($M = N = 30, L = 4$). The second is a Chimera graph with 6800 vertices and 31680 edges ($M = N = 20, L = 4$). For convenience, we will call them $host_1$ and $host_2$ respectively. Standard adjacency list for both hosts can be generated using script H in the appendix. The actual hardware structure of the D-Wave quantum computers always had blocks of $K_{4,4}$, $host_2$ has blocks of $K_{8,8}$ which doubles the connectivity (number of edges) inside each block. As a result, $host_2$ has about 50% more edges than $host_1$. We purposely pick $K_{8,8}$ to see whether a large increase in the connectivity inside each block (which seems a more challenging task from an engineering point of view) would lead to better embeddings.

We ran the same minor embedding algorithm used in Section 3.3 on a number of different test cases. We also used the same graphs from Section 3.3 as the input graph G_1 to the Graph Isomorphism Problem. A random permutation of the vertices was generated to obtain G_2 (i.e. we relabeled the vertices of G_1 to get G_2), hence the two graphs will always have the same number of vertices and edges, and they are always isomorphic to one another. The QUBO instances for the clique and direct approach can be generated using Python scripts E, F and G. This particular minor embedding algorithm is very time consuming, and therefore we only did two trials on each test cases and only the better result (one with less number of physical qubits) is shown here. We also set a time out value of about 10 minutes for each trial, that is, the algorithm is terminated after 10 minutes even if a minor embedding was not found.

4.4.1 Presentation of results

In Table 4.4, the second and third column contain the *order* and *size* of the input graphs G_1 and G_2. The next three columns contains the number of variables of the QUBO instances obtained through the three different approaches described in Section 4.1, 4.2 and 4.3. The last two columns contain the densities of the QUBO instances for the direct and clique formulation. The density of a QUBO instance is defined as the number of non-zero entries in $Q_{(i,j)}$ where $i < j$ divided by the total number of entries in the same part of the matrix. Note that the main diagonal was excluded as the connection of a logical qubit to itself does not affect the minor embedding of the guest graph.

In Table 4.5, we provide the embedding result for both $host_1$ and $host_2$. It contains the same standard information about the minor embedding as explained in Section 3.3.

4.4.2 Discussion

The IP formulation does not seem very useful here from a practical point of view, the number of variables required quickly scales out of control and results in large number of logical qubits. An interesting thing to note here is that the IP formulation of the Broadcast Time Problem studies in [10] also require a huge number of binary variables when converted to QUBO. In all test cases, the density of the QUBO instances from the direct formulation is always smaller or equal to the one obtained by the clique approach. In theory, it means that the QUBO instance of the direct formulation should be easier to embed on the host graph. However, the minor embedding algorithm we used is a heuristic algorithm that heavily relies on a randomized initialization procedure [7]. As a result, the minor embedding found on the same set of guest and host graphs may vary quite a lot on two different trials. Hence there are several test cases where the direct QUBO instance requires more physical qubits.

In general, the numbers fit our expectation, that is, the direct QUBO formulation requires less number of physical qubits to embed on the host than the clique approach and the embedding max chain length is shorter for the direct method as well. This pattern becomes clearer as the scale of the QUBO instance gets larger. We suspect that the difference in the densities is related to the fact that the Clique approach does not make any explicit assumptions on the two input graphs G_1 and G_2 having the same order and size. In other words, the direct formulation uses more information of the input. Together with the absurdly large number of variables required from the IP approach, it shows how a directly encoded QUBO instance for the problem could outperform one that is obtained through a more generic approach (reduction from IP and Clique here) in practice.

The entries in the tables marked by a '-' correspond to the cases where the algorithm was not able to find a minor embedding. The large number of such cases is likely due to the big scaling of the number of physical qubits required. As the order of input graphs increase by 1, the number of physical qubits approximately doubles. Even with the large increase of connectivity in $host_2$, the algorithm was still not able to embed any test cases with order (of G_1 and G_2) bigger than 12. With $host_1$, the largest embeddable case had an order of 11. So it does not seems like upscaling the internal connectivity inside each block made too much difference here. In summary, the overall result is not very satisfactory. As we explained in Section 3.3.3, we suspect that longer embedding chains may lead to less accurate results. Hence the huge chains we have in both test cases could prevent us from obtaining any meaningful solution from the machine even if the scale of the actual hardware is up to the size of $host_1$ and $host_2$.

Table 4.4: Number of variables and QUBO densities for different formulations

			Logical Qubits			Density	
Graph	Order	Size	IP	Clique	Direct	Clique	Direct
BidiakisCube	12	18	6380377398	144	144	0.4895	0.3217
Bull	5	5	30315	25	25	0.6667	0.5000
Butterfly	5	6	74539	25	25	0.6533	0.4933
C10	10	10	17762575	100	100	0.4646	0.3232
C11	11	11	34339547	121	121	0.4333	0.3000
C12	12	12	252442038	144	144	0.4056	0.2797
C4	4	4	4056	16	16	0.6667	0.5333
C5	5	5	30315	25	25	0.6667	0.5000
C6	6	6	223191	36	36	0.6286	0.4571
C7	7	7	543235	49	49	0.5833	0.4167
C8	8	8	1194584	64	64	0.5397	0.3810
C9	9	9	8654166	81	81	0.5000	0.3500
Chvatal	12	24	68183718414	144	144	0.5455	0.3497
Clebsch	16	40	5142153247264	256	256	0.5098	0.3137
Diamond	4	5	10104	16	16	0.5667	0.4833
Dinneen	9	21	3607826070	81	81	0.5889	0.3944
Dodecahedral	20	30	3400324505130	400	400	0.3358	0.2155
Durer	12	18	6380377398	144	144	0.4895	0.3217
Errera	17	45	155792785116353	289	289	0.5047	0.3079
Frucht	12	18	6380377398	144	144	0.4895	0.3217
GoldnerHarary	11	27	23558624475	121	121	0.5832	0.3749
Grid2x3	6	7	543061	36	36	0.6413	0.4635
Grid3x3	9	12	63111360	81	81	0.5556	0.3778
Grid3x4	12	17	4013029118	144	144	0.4775	0.3157
Grid4x4	16	24	68183720736	256	256	0.4000	0.2588
Grid4x5	20	31	4617380539608	400	400	0.3423	0.2188
Grotzsch	11	20	2515662329	121	121	0.5523	0.3595
Heawood	14	21	22548907234	196	196	0.4410	0.2872
Herschel	11	18	1160070477	121	121	0.5336	0.3501
Hexahedral	8	12	14251368	64	64	0.6032	0.4127
Hoffman	16	32	766017051200	256	256	0.4627	0.2902
House	5	6	74539	25	25	0.6533	0.4933
Icosahedral	12	30	443520588882	144	144	0.5734	0.3636
K10	10	45	1156161672465	100	100	0.1818	0.1818
K2,3	5	6	74539	25	25	0.6533	0.4933
K2	2	1	12	4	4	0.6667	0.6667
K3,3	6	9	2423145	36	36	0.6286	0.4571
K3,4	7	12	14251185	49	49	0.6173	0.4337
K3	3	3	552	9	9	0.5000	0.5000
K4,4	8	16	88342552	64	64	0.6032	0.4127
K4,5	9	20	2515661504	81	81	0.5951	0.3975
K4	4	6	21932	16	16	0.4000	0.4000
K5,5	10	25	13219293745	100	100	0.5859	0.3838
K5,6	11	30	52178894829	121	121	0.5799	0.3733
K5	5	10	1062875	25	25	0.3333	0.3333
K6,6	12	36	2087102677590	144	144	0.5734	0.3636
K6	6	15	58434165	36	36	0.2857	0.2857
K7	7	21	515404071	49	49	0.2500	0.2500
K8	8	28	3442573800	64	64	0.2222	0.2222
K9	9	36	208710268992	81	81	0.2000	0.2000
Krackhardt	10	18	1160070063	100	100	0.5745	0.3782
Octahedral	6	12	14251011	36	36	0.5143	0.4000
Pappus	18	27	1278055259613	324	324	0.3653	0.2353
Petersen	10	15	308866125	100	100	0.5455	0.3636
Poussin	15	39	4138707982302	225	225	0.5336	0.3293
Q3	8	12	14251368	64	64	0.6032	0.4127
Q4	16	32	766017051200	256	256	0.4627	0.2902
Robertson	19	38	31291626737038	361	361	0.4111	0.2556
S10	11	10	17762989	121	121	0.4146	0.2906
S2	3	2	190	9	9	0.7222	0.6111
S3	4	3	1358	16	16	0.7000	0.5500
S4	5	4	10583	25	25	0.6533	0.4933
S5	6	5	80505	36	36	0.6032	0.4444
S6	7	6	223365	49	49	0.5561	0.4031
S7	8	7	543418	64	64	0.5139	0.3681
S8	9	8	3924080	81	81	0.4765	0.3383
S9	10	9	8654577	100	100	0.4436	0.3127
Shrikhande	16	48	24727250232768	256	256	0.5412	0.3294
Sousselier	16	27	182579326560	256	256	0.4254	0.2715
Tietze	12	18	6380377398	144	144	0.4895	0.3217
Wagner	8	12	14251368	64	64	0.6032	0.4127

Table 4.5: Minor embedding results

	Physical Qubits				Embedding Max Chain			
	$host_1$		$host_2$		$host_1$		$host_2$	
Graph	Clique	Direct	Clique	Direct	Clique	Direct	Clique	Direct
BidiakisCube	-	-	4916	4221	-	-	54	54
Bull	220	183	114	101	10	10	6	6
Butterfly	210	160	100	101	11	9	6	6
C10	4216	3402	1997	1857	66	51	31	28
C11	5996	5685	3534	2699	79	76	44	30
C12	-	-	5579	4735	-	-	66	47
C4	65	70	40	40	5	5	3	3
C5	192	171	107	98	10	8	6	5
C6	516	406	218	206	17	14	8	8
C7	963	765	464	409	27	18	12	11
C8	1448	1345	897	702	31	31	19	15
C9	2956	2362	1343	1203	48	39	25	24
Chvatal	-	-	5440	4710	-	-	55	53
Clebsch	-	-	-	-	-	-	-	-
Diamond	70	65	44	41	6	5	3	4
Dinneen	3021	2232	1283	1363	64	37	21	24
Dodecahedral	-	-	-	-	-	-	-	-
Durer	-	-	4833	4961	-	-	54	51
Errera	-	-	-	-	-	-	-	-
Frucht	-	-	4546	5373	-	-	55	55
GoldnerHarary	6457	6027	2920	3342	88	73	35	44
Grid2x3	445	348	233	252	17	14	9	9
Grid3x3	2893	2523	1508	1367	54	46	27	22
Grid3x4	-	-	5050	5169	-	-	54	52
Grid4x4	-	-	-	-	-	-	-	-
Grid4x5	-	-	-	-	-	-	-	-
Grotzsch	6204	5990	3225	3404	83	95	47	43
Heawood	-	-	-	-	-	-	-	-
Herschel	6059	5986	3181	3043	75	80	41	40
Hexahedral	1754	1385	812	857	38	31	19	17
Hoffman	-	-	-	-	-	-	-	-
House	210	178	117	110	10	8	6	7
Icosahedral	-	-	4738	4775	-	-	53	46
K10	2513	2381	1361	1233	54	43	21	18
K2,3	192	179	96	92	12	8	5	6
K2	4	4	4	4	1	1	1	1
K3,3	312	380	173	187	11	14	7	7
K3,4	765	698	319	329	21	19	9	10
K3	20	20	12	12	3	3	2	2
K4,4	1792	1205	558	715	41	26	10	17
K4,5	1622	2019	1373	954	26	34	28	19
K4	58	60	44	43	4	5	3	3
K5,5	5178	3342	2178	1651	67	41	28	20
K5,6	6385	5553	3216	2498	83	78	42	32
K5	148	150	95	94	7	8	5	5
K6,6	-	-	4856	4182	-	-	50	50
K6	291	313	167	179	11	11	6	6
K7	558	577	345	320	19	16	10	9
K8	964	925	513	481	24	22	13	10
K9	1663	1574	825	937	32	33	15	17
Krackhardt	4132	4168	2133	1871	67	55	30	28
Octahedral	434	306	212	196	15	12	8	7
Pappus	-	-	-	-	-	-	-	-
Petersen	5030	4044	2299	2444	78	60	33	33
Poussin	-	-	-	-	-	-	-	-
Q3	1953	1671	854	807	41	37	19	17
Q4	-	-	-	-	-	-	-	-
Robertson	-	-	-	-	-	-	-	-
S10	5935	4956	2801	2267	79	70	35	31
S2	24	24	13	13	4	3	2	2
S3	77	67	47	43	6	6	4	4
S4	199	161	111	106	10	9	6	5
S5	441	402	214	214	16	14	8	8
S6	805	713	380	385	25	20	10	11
S7	1503	1180	644	652	37	28	14	16
S8	2554	1872	1114	1064	67	49	20	25
S9	5031	3205	2398	1421	97	52	38	25
Shrikhande	-	-	-	-	-	-	-	-
Sousselier	-	-	-	-	-	-	-	-
Tietze	-	-	4648	3892	-	-	50	55
Wagner	1728	1408	875	718	43	31	21	18

4.5 The Subgraph Isomorphism Problem

The Subgraph Isomorphism Problem – a generalization of the Graph Isomorphism Problem – is the NP-hard problem of determining whether a graph G_1 is a subgraph of another graph G_2.

Definition 25. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. We say that G_1 is a* ***subgraph*** *of G_2 if if there exist an* ***edge-preserving*** *injective function $f : V_1 \to V_2$ such that if $\{u, v\} \in E_1$, then $\{f(u), f(v)\} \in E_2$.*

Subgraph Isomorphism Problem:

Instance:	Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $n_1 = \lvert V_1\rvert \le \lvert V_2\rvert = n_2$ and $\lvert E_1\rvert \le \lvert E_2\rvert$.
Question:	Find an edge-preserving injective function $f : V_1 \to V_2$ if such f exists.

Note that the problem is trivial if G_1 is a graph of order 1, as it will always be a subgraph of any other non-empty graphs. So we will assume that all input graphs are of order at least 2.

4.5.1 A direct QUBO formulation

The objective function (4.5) can be modified to solve the Subgraph Isomorphism Problem using the same method as in Section 4.2. As the order of G_1 and G_2 can be different, all possible mappings could be represented by a vector $\mathbf{x} \in \mathbb{Z}_2^{n_1 n_2}$:

$$\mathbf{x} = (x_{0,0}, x_{0,1}, \ldots, x_{0,n_2-1}, x_{1,0}, x_{1,1}, \ldots, x_{1,n_2-1}, \ldots, x_{n_1-1,0}, \ldots, x_{n_1-1,n_2-1}).$$

We also need n_2 slack variables encoded in $\mathbf{y} = (y_0, y_1, \ldots, y_{n_2-1})$, which will be appended to $\mathbf{x} \in \mathbb{Z}_2^{n_1 n_2}$ to form the binary vector $\mathbf{z} \in \mathbb{Z}_2^{(n_1+1)n_2}$:

$$\mathbf{z} = \mathbf{x}\mathbf{y}.$$

Let

$$F(\mathbf{z}) = H(\mathbf{z}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}), \tag{4.11}$$

where

$$H(\mathbf{z}) = \sum_{0 \le i < n_1} \left(1 - \sum_{0 \le i' < n_2} x_{i,i'}\right)^2 + \sum_{0 \le i' < n_2} \left(1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'}\right)^2, \tag{4.12}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{0 \le i' < n_2} \left(x_{i,i'} \sum_{0 \le j' < n_2} x_{j,j'}(1 - e_{i',j'})\right). \tag{4.13}$$

The definition of the decoder function $D : \mathbb{Z}_2^{n_1 n_2} \to \mathcal{F}$ is:

$$\operatorname{dom}(D) = \left\{ \mathbf{x} \in \mathbb{Z}_2^{n_1 n_2} \;\middle|\; \sum_{0 \le i' < n_2} x_{i,i'} = 1, \text{ for all } 0 \le i < n_1 \right.$$
$$\left. \text{and} \sum_{0 \le i < n_1} x_{i,i'} = 1, \text{ for all } 0 \le i' < n_2 \right\},$$

and

$$D(\mathbf{x}) = \begin{cases} f, & \text{if } \mathbf{x} \in \operatorname{dom}(D), \\ \text{undefined}, & \text{otherwise}, \end{cases}$$

where $f : V_1 \to V_2$ is an injection such that $f(v_i) = v_{i'}$ where $x_{i,i'} = 1$.

Lemma 11. *For every* $\mathbf{z} \in \mathbb{Z}_2^{(n_1+1)n_2}$ *corresponding to the solution* $z^* = \min_{\mathbf{z}} F(\mathbf{z})$*,* $H(\mathbf{z}) = 0$ *if and only if* $D(\mathbf{x})$ *is defined (in this case* $D(\mathbf{x})$ *is an injection).*

Proof. Fix $\mathbf{z} \in \mathbb{Z}_2^{(n_1+1)n_2}$ where $\mathbf{z}$ corresponds to an optimal solution of $z^* = \min_{\mathbf{z}} F(\mathbf{z})$. The term $H(\mathbf{z})$ has two components,

$$\sum_{0 \le i < n_1} \left(1 - \sum_{0 \le i' < n_2} x_{i,i'} \right)^2 \text{ and } \sum_{0 \le i' < n_2} \left(1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'} \right)^2.$$

Since both components consist of only quadratic terms, we have $H(\mathbf{z}) = 0$ if and only if both components are equal to 0. First,

$$\sum_{0 \le i < n_1} \left(1 - \sum_{0 \le i' < n_2} x_{i,i'} \right)^2 = 0 \tag{4.14}$$

if and only if for each $0 \le i < n$, exactly one variable in the set $\{x_{i,i'} \mid 0 \le i' < n_2\}$ has value 1, hence every vertex $v \in V_1$ has exactly one image in V_2.

Second, with a similar argument,

$$\sum_{0 \le i' < n_2} \left(1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'} \right)^2 = 0 \tag{4.15}$$

if and only if for each $0 \le i' < n_2$, $1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'} = 0$. We have the following cases:

1. None of the variables in the set $\{x_{i,i'} \mid 0 \le i < n_1\}$ has value 1.
2. Exactly one variable in the set $\{x_{i,i'} \mid 0 \le i < n_1\}$ has value 1.

3. More than one variables in the set $\{x_{i,i'} \mid 0 \leq i < n_1\}$ have value of 1.

In the first case, we have $1 - \sum_{0 \leq i < n_1} x_{i,i'} = 1$, so setting $y_{i'} = 1$ will avoid the penalty. By assumption, $n_1 \leq n_2$, so if $n_1 < n_2$, then not all vertices in V_2 should have a pre-image and no penalty should be given in this case. If $n_1 = n_2$ however, the mapping from V_1 to V_2 should be a bijection. Since condition (4.14) enforces every vertex in V_1 to have an image in V_2, we will have either Case 2 or 3.

In the second case, exactly one variable in the set $\{x_{i,i'} \mid 0 \leq i < n_1\}$ has value 1 which means there is exactly one vertex in $v_i \in V_1$ that has been mapped to $v_{i'} \in V_2$. As a result, $1 - \sum_{0 \leq i < n_1} x_{i,i'} - y_{i'} = 0$ when $y_{i'}$ is assigned value 0.

In the last case, the mapping can not be injective, so no values for $y_{i'}$ can avoid the penalty.

Together conditions (4.14) and (4.15) are equivalent with the property that every vertex $v_i \in V_1$ is mapped to exactly one unique vertex $v_{i'} \in V_2$, that is, the map $v_i \mapsto v_{i'}$ is injective. $\square$

In the proof of Lemma 7, the bijectivity of the mapping f was essential to prove edge-invariance. With the same argument one can prove that an injective function f is edge-preserving. Indeed, for each edge $ij \in E_1$, the equality $P_{i,j} = 0$ ensures that there is an edge $f(i)f(j) \in E_2$, so we have the following result:

Lemma 12. *Let $\mathbf{x} \in \mathbb{Z}_2^{n_1 n_2}$ and assume that $D(\mathbf{x})$ is an injective function. Then, $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) = 0$ if and only if the mapping $f = D(\mathbf{x})$ is edge-preserving.*

Theorem 13. *For every $\mathbf{z} \in \mathbb{Z}_2^{n_1 n_2}$, $F(\mathbf{z}) = 0$ if and only if the mapping $f : V_1 \to V_2$ defined by $f = D(\mathbf{x})$ is a subgraph isomorphism.*

Proof. As in the proof of Theorem 8, the statement of the theorem is a direct consequence of Lemmata 11 and 12. $\square$

4.5.2 Another direct QUBO formulation

In this subsection, we present a different QUBO formulation of the Subgraph Isomorphism Problem that does not need any slack variables. This alternative formulation uses a vector $\mathbf{x} \in \mathbb{Z}_2^{n_1 n_2}$:

$$\mathbf{x} = (x_{0,0}, x_{0,1}, \ldots, x_{0,n_2-1}, x_{1,0}, x_{1,1}, \ldots, x_{1,n_2-1}, \ldots, x_{n_1-1,0}, \ldots, x_{n_1-1,n_2-1}).$$

The objective function this time is of the form:

$$F(\mathbf{x}) = H(\mathbf{x}) + b \sum_{ij \in E_1} P_{i,j}(\mathbf{x}), \ b \in \mathbb{R}^+ \tag{4.16}$$

where

$$H(\mathbf{x}) = a \sum_{1\le i\le n_1} (1 - \sum_{1\le i'\le n_2} x_{i,i'})^2 + \sum_{1\le i'\le n_2} (1 - \sum_{1\le i\le n_1} x_{i,i'})^2, \ a > 1, \tag{4.17}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{1\le i'\le n_2} \left(x_{i,i'} \sum_{1\le j'\le n_2} x_{j,j'}(1 - e_{i',j'}) \right). \tag{4.18}$$

Assuming we have the solution to $x^* = \min_{\mathbf{x}} F(\mathbf{x})$, the same decoder function D in Section 4.5.1 can be used again to obtain the vertex mapping f.

In order for the formulation to be correct, the constant b has to be sufficiently large than a, this fact will be mentioned in the proof of Lemma 16.

Theorem 14. *If G_1 is a subgraph of G_2, assuming $x^* = \min_{\mathbf{x}} F(\mathbf{x})$, then $D(\mathbf{x})$ is an edge-preserving vertex mapping.*

Before we can prove Theorem 14, we need two lemmata.

Lemma 15. *The term $H(\mathbf{x})$ is bounded.*

Proof. Since $H(\mathbf{x})$ contains two quadratic terms, we have $H(\mathbf{x}) \ge 0$. $H(\mathbf{x})$ is also bounded above as the largest penalty we can obtain from $H(\mathbf{x})$ is when all variables $x_{i,j}$ are set to 1, in this case we get $H(\mathbf{x}) = an_1(1 - n_2)^2 + n_2(1 - n_1)^2$ as the upper bound. □

Lemma 16. *Let $x^* = \min_{\mathbf{x}} F(\mathbf{x})$, then $P_{i,j}(\mathbf{x}) = 0$ for all $ij \in E_1$.*

Proof. Assume we have $P_{i,j}(\mathbf{x}) \ne 0$ for some $ij \in E_1$. This means that we have i and j mapped to i' and j' respectively and $i'j' \notin E_2$. If this is the case, we can reduce $F(\mathbf{x})$ further by setting $x_{i,i'}$ and $x_{j,j'}$ to 0.

Setting $x_{i,i'}$ and $x_{j,j'}$ to 0 may or may not add a penalty to $H(\mathbf{x})$, but since Lemma 15 states that $H(\mathbf{x})$ is bounded, the objective function $F(\mathbf{x})$ will increase by at most the upper bound of $H(\mathbf{x}) = an_1(1 - n_2)^2 + n_2(1 - n_1)^2$. Therefore with sufficiently large b, the overall value of $F(\mathbf{x})$ will decrease. A contradiction to $x^* = \min_{\mathbf{x}} F(\mathbf{x})$. □

Now we can begin the proof of Theorem 14.

Proof. Suppose we have two graphs G_1 and G_2 where G_1 is a subgraph of G_2. Assume we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$.

By Lemma 16, the term $\sum_{ij\in E_1} P_{i,j}$ in $F(\mathbf{x})$ must equal to 0. So the value of $F(\mathbf{x})$ solely depends on $H(\mathbf{x})$. Consider the assignment of variables that has a minimum penalty in H. Since $H(\mathbf{x})$ consists of two quadratic terms, we have $H(\mathbf{x}) \ge 0$. There are two cases here:

- There exists a bijection between V_1 and V_2 (e.g. $n_1 = n_2$).
- There is no bijection between V_1 and V_2 (e.g. $n_1 < n_2$).

In the former case, we must have $n_1 = n_2$. Using the same argument in the proof of Lemma 6, we can show that $H(\mathbf{x}) = 0$ is the minimum value of $H(\mathbf{x})$ and the decoded function f must be a bijection in this case. And since G_1 is a subgraph of G_2 by assumption, we know that an edge-preserving bijection from V_1 to V_2 exists. If f is not edge-preserving, then $P_{i,j} \neq 0$ for some $ij \in E_1$, Lemma 16 forbids this as it contradicts the assumption that $x^* = \min_{\mathbf{x}} F(\mathbf{x})$.

In the latter case, we have $n_1 < n_2$. By construction, the term $a \sum_{1 \leq i \leq n_1}(1 - \sum_{1 \leq i' \leq n_2} x_{i,i'})^2$ in $H(\mathbf{x})$ will be minimized with priority by the choice of a. So the function f decoded in this case has to be an injection. Once again, we know an edge-preserving function exists by assumption. If this function is not found by decoding x^*, it contradicts the assumption that $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ by Lemma 16 again. □

We also have the following corollary as a direct consequence (contrapositive) of Theorem 14.

Corollary 17. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs, assuming $x^* = \min_{\mathbf{x}} F(\mathbf{x})$. If the mapping $D(\mathbf{x})$ is not an edge-preserving injection from V_1 to V_2, then G_1 is not a subgraph of G_2.*

4.5.3 Post-processing verification

Theorem 14 is of a weaker form than Theorem 13. To demonstrate the difference, suppose we have $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Assume that whether G_1 is a subgraph of G_2 is unknown. Let $x^* = F(\mathbf{x}^*)$ and $y^* = F'(\mathbf{y}^*)$ be the optimal solutions and their corresponding variable assignment we obtain using objective function (4.11) and (4.16) respectively.

If $x^* = 0$, then we know by Theorem 13 that G_1 is a subgraph of G_2. We do not need to generate the actual mapping $f : V_1 \to V_2$ using the decoder function D to verify its correctness. In other words, the correctness of f is encoded in x^*. This is not the case with y^* and $\mathbf{y}^*$. Theorem 14 does not provide a constant value for y^* to distinguish whether G_1 is a subgraph of G_2. As a result, we need to verify the mapping f encoded in $\mathbf{y}^*$ to check if it is a valid edge-preserving mapping. There are two things to check:

- Is the function $f = D(\mathbf{y}^*)$ an injective function.
- Is the function $f = D(\mathbf{y}^*)$ edge-preserving.

Both steps can be done efficiently and the overall verification can be performed in polynomial time. Depending on whether f is an edge-preserving injective function or not, by Theorem 14 and Corollary 17, we know whether G_1 is a subgraph of G_2 depending on the verification result.

Note that it could be the case that with specific values of a and b, the minimum value of objective function (4.16) could provide extra information about the correctness of the mapping f. We did not try to proof this fact explicitly. The main point of the this alternative formulation is only to show that the redundancy in the number of variables could be reduced sometimes by manipulating the objective function in certain ways.

4.5.4 Formulation via reduction to Clique

In this section we give an alternate QUBO formulation that requires $n_1 n_2$ binary variables for checking if a graph of order n_1 is a subgraph of a graph of order n_2. We slightly modify the product graph construction given earlier in Section 4.3. The product graph $\Psi'(G_1, G_2)$ this time has the same vertex set V as in Definition 24, the edges are $E = \{((a,b),(c,d)) \in V \times V \mid a \neq c, b \neq d \text{ and } (ac \notin E_1 \text{ or } bd \in E_2)\}$.

Theorem 18. *The graph $G_1 = (V_1, E_1)$ is a subgraph of the graph $G_2 = (V_2, E_2)$ with $n_1 = |V_1| \leq |V_2| = n_2$ if and only if $\chi(\Psi'(G_1, G_2)) = n_1$.*

Proof. We first consider the case when G_1 is a subgraph of G_2 and $f : V_1 \to V_2$ is the edge-preserving injective mapping. We claim that the subset of vertices $V' = \{(i, f(i)) \mid i \in V_1\} \subseteq V$ is a clique in $\Psi'(G_1, G_2)$. By construction, V' has n_1 vertices, so we only need to check the existence of an edge between any pair in it. Let (a,b) and (c,d) be in V', with $a \neq c$. Since f is injective by assumption, if $b = f(a)$ and $d = f(c)$, then $b \neq d$. Furthermore, since f is edge-preserving, if $ac \in E_1$ then $bd \in E_2$. With the definition of E for $\Psi'(G_1, G_2)$, this means that $((a,b),(c,d))$ is an edge in E.

Conversely, suppose $V' \subseteq V$ is a clique of order n_1 in $\Psi'(G_1, G_2)$. From the definition of E, we know that for any pair of distinct vertices (a,b) and (c,d) in E, $a \neq c$ and $b \neq d$, so the same condition is true for all pairs of vertices (a,b) and (c,d) in V'. Accordingly, a well-defined injective function $f : V_1 \to V_2$ can be defined by setting $f(a) = b$, for all $(a,b) \in V'$. In order for f to be a subgraph isomorphism the following condition has to be satisfied: if $ac \in E_1$ then $f(a)f(c) \in E_2$. Since V' is a clique by assumption, if $a \neq c$, then $((a, f(a), (c, f(c)) \in E$. Therefore, in view of the definition of E, we must have either $ac \notin E_1$ or $f(a)f(c) \in E_2$, that is, f is edge-preserving. □

To solve the Subgraph Isomorphism Problem we just construct a QUBO instance for the Clique Problem as in Section 4.3.

4.6 The Induced Subgraph Isomorphism

The Induced Subgraph Isomorphism Problem is a NP-hard problem related to both the Graph and Subgraph Isomorphism Problems. The input are two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_1)$ and the goal is to find an injective edge-invariant vertex mapping $f : V_1 \to V_2$. We formally define the problem as follows:

Induced Subgraph Isomorphism Problem:

Instance: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $n_1 = |V_1| \leq |V_2| = n_2$ and $|E_1| \leq |E_2|$.

Question: Find an edge-invariant injective function $f : V_1 \to V_2$ if such f exists.

4.6.1 A direct formulation

We can extend the QUBO formulation given for the Subgraph Isomorphism Problem in Section 4.5.1 to solve the Induced Subgraph Isomorphism Problem. This formulation uses the same binary variable vector $\mathbf{z} \in \mathbb{Z}_2^{(n_1+1)n_2}$ which is the concatenation of two vectors $\mathbf{x} \in \mathbb{Z}_2^{n_1 n_2}$ and $\mathbf{y} \in \mathbb{Z}_2^{n_2}$, each serving the same purpose as in Section 4.5.1: $\mathbf{x}$ encodes the injective function f, $\mathbf{y}$ counter balances unnecessary penalties. At the end, the decoder function $D : \mathbb{Z}_2^{n_1 n_2} \to \mathcal{F}$ described in Section 4.5.1 can be used again to obtain the actual mapping.

The objective function $F(\mathbf{z})$ has the following form:

$$F(\mathbf{z}) = H(\mathbf{z}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}) + \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}), \tag{4.19}$$

$$H(\mathbf{z}) = \sum_{0 \leq i < n_1} (1 - \sum_{0 \leq i' < n_2} x_{i,i'})^2 + \sum_{0 \leq i' < n_2} (1 - \sum_{0 \leq i < n_1} x_{i,i'} - y_i)^2, \tag{4.20}$$

$$P_{i,j}(\mathbf{x}) = \sum_{0 \leq i' < n_2} \left(x_{i,i'} \sum_{0 \leq j' < n_2} x_{j,j'}(1 - e_{i',j'}) \right), \tag{4.21}$$

and

$$N_{i,j}(\mathbf{x}) = \sum_{0 \leq i' < n_2} \left(x_{i,i'} \sum_{0 \leq j' < n_2} x_{j,j'} e_{i',j'} \right). \tag{4.22}$$

The parts $H(\mathbf{z})$ and $\sum_{ij \in E_1} P_{i,j}(\mathbf{x})$ serve the same purpose as in Equation (4.11), that is, $H(\mathbf{z})$ ensures the mapping decoded by $D(\mathbf{x})$ is injective and $\sum_{ij \in E_1} P_{i,j}(\mathbf{x})$ guarantees the mapping is edge-preserving. Since both parts are identical as in Equation (4.11), Lemmata 11 and 12 hold and can be proved with the same argument as in Section 4.5.1.

The vertex mapping required for the Induced Subgraph Isomorphism Problem has to be edge-invariant instead of edge-preserving. This means we need one more condition, namely, for all $ij \notin E_1$ we have $f(i)f(j) \notin E_2$. This property is ensured by the new term $\sum_{ij \notin E_1} N_{i,j}(\mathbf{x})$ in $F(\mathbf{x})$.

Lemma 19. *Let $\mathbf{x} \in \mathbb{Z}_2^{n_1 n_2}$ and assume that $D(\mathbf{x})$ is a function (injective). Then $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) + \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}) = 0$ if and only if the mapping $f = D(\mathbf{x})$ is edge-invariant.*

Proof. Since both $P_{i,j}(\mathbf{x})$ and $N_{i,j}(\mathbf{x})$ contain quadratic terms only, we have $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) + \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}) = 0$ if and only if $P_{i,j}(\mathbf{x}) = N_{i,j}(\mathbf{x}) = 0$, for all i and j.

As Lemma 12 holds here, f has to be edge-preserving when $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) = 0$, hence we only need to show that the non-edges are preserved as well under f. This is indeed true as the term $(1 - e_{i',j'})$ in Equation (4.7) is replaced by $e_{i',j'}$ in $N_{i,j}$. If $\sum_{ij \notin E_1} N_{i,j}(\mathbf{x}) = 0$, then for all $ij \notin E_1$ we must have $f(i)f(j) \notin E_2$. Hence f is edge-invariant if $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) + \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}) = 0$.

Conversely, if $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) + \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}) \neq 0$, then either at least one term of the sum has to be non-zero. Therefore, either f is not edge-preserving by Lemma 12 or for some $ij \notin E_1$, $f(i)f(j) \in E_2$. In either case, f is not edge-invariant.

□

Next we show the correctness of our direct QUBO formulation.

Theorem 20. *For every $\mathbf{z} \in \mathbb{Z}_2^{n_1 n_2}$, $F(\mathbf{z}) = 0$ if and only if the mapping $f : V_1 \to V_2$ defined by $f = D(\mathbf{x})$ is an induced subgraph isomorphism.*

Proof. If $F(\mathbf{z}) = 0$, then $H(\mathbf{z}) = \sum_{ij \in E_1} P_{i,j}(\mathbf{x}) = \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}) = 0$. By Lemmata 11, 12 and 19, the mapping f must be injective and edge-invariant, therefore an induced subgraph isomorphism.

On the other hand, if $F(\mathbf{z}) \neq 0$, at least one of the terms $H(\mathbf{z})$, $\sum_{ij \in E_1} P_{i,j}(\mathbf{x})$ or $\sum_{ij \notin E_1} N_{i,j}(\mathbf{x})$ is not 0. By Lemmata 11, and 19, at least one of the requirement for an induced subgraph isomorphism is not met, hence f is not an induced subgraph isomorphism. □

4.6.2 Formulation via reduction to Clique

In this section we give an alternate QUBO formulation that uses only $n_1 n_2$ binary variables for checking if a graph of order n_1 is an induced subgraph of a graph of order n_2. It turns out that we can use the same product graph construction given earlier in Section 4.3. Note that in the proof of Theorem 9 we do not actually require the property that f is a bijection, we only require f to map different vertices in V_1 to different vertices in V_2 hence an injection f is sufficient. As a result, we have the following corollary.

Corollary 21. *A graph G_1 is an induced subgraph of a graph G_2 with $n_1 = |V_1| \leq |V_2| = n_2$ if and only if $\chi(\Psi(G_1, G_2)) = n_1$.*

We then construct a QUBO instance for the Clique Problem, as done earlier, so we can solve the Induced Subgraph Isomorphism Problem.

Chapter 5

Conclusions

We will provide a summary of the studies we have conducted as well as some possible future works in this chapter.

We did not provide any benchmarking on the total computation time required to solve any of the problems studied in this thesis between the method of using the D-Wave quantum computer and any classical algorithms. Such benchmarking would require a set of carefully formulated criteria in order to be able to provide any meaningful insights, and current theory studies and engineering obstacles are preventing us from determining them. For example, consider the experimental result we provided on the Dominating Set problem in Chapter 3, and suppose we want to solve an instance of the problem P using its QUBO formulation and the quantum annealer. Using the method we described in Section 2.3, the total time T required to solve the problem is

$$T = Translate + Embed + n \times (Query + Decode)$$

where $Translate$ and $Embed$ are the time required to obtain and embed the QUBO instance on the hardware and $n \times (Query + Decode)$ is the time needed to query the quantum annealer and decoding the solution n times. Ideally, we want the machine to have probability 1 of finding the optimal solution in one query so we have probability 1 of solving P with $n = 1$. But since the probability of obtaining the optimal solution using current hardware is relatively low (based on the data in Table 3.5 and 3.6), we have to query the machine many times to have an acceptable probability of finding the optimal solution of P in practice. The low probability could be related to a number of engineering issues, and it reasonable to assume that through time, the hardware precision will increase. Together with the development of tools which can reduce error rates furthermore ([46] for example), hopefully we will see machines which are more consistent and accurate in the future.

Note that there is one other big overhead in T, namely, the term $Embed$ which corresponds to the time needed to solve the NP-hard Minor Containment Problem, this is closely related to the next two points.

5.1 Hardware design

The host structure of all generations (up to this point) of D-Wave quantum computers are all Chimera graphs. The embedding result we had in Section 4.4 shows that it might be the case that the 'scalability' of Chimera graphs is not good in terms of the Minor Containment Problem. Even after a 50% increase in connectivity (number of edges) in the host graphs, the order of the QUBO instance that could be embedded only went from 121 to 144. Note that the maximum minor embedding chain length did decrease by a considerable amount for the larger host though. The exact cause of such low scalability is unknown, but we suspect that it might be related to some unknown properties of the Chimera graphs.

Similar to classical computers, building machines of larger scales is an important part of hardware development for quantum computers as well. As we have mentioned in Section 2.3, we can embed larger QUBO instances with a bigger host and the length of the minor embedding chains will also be shorter in general. Thus making it possible to solve bigger problems and to obtain more accurate answers. D-Wave has been successfully doubling the number of qubits on their hardware each year for the last 8 years now and is planning on doing so by constantly upscaling the current design in the future [14]. If the poor scalability is due to some inherent features of the Chimera structure, then maybe a hardware redesign could enable more potential of the machine. We are planning on verifying this concept experimentally (and/or theoretically) in the future.

5.2 Hybrid model

Let us put all the practical difficulties of engineering and manufacturing aside for a moment and assume we live in a world where quantum computers with perfected hardware exist. Would quantum computers replace classical ones by this point? No doubt there are problems only a quantum computer can solve [23], but it does not mean a classical computer would be of no value here. In fact, it has been shown in studies that sometimes, a combination of quantum and classical computing can be combined to achieve what each model can not perform alone [9, 35].

The concept of this quantum-classical computation, or *quassical* (see [1] for a more detailed discussion) computation for short, can also be adapted here to solve some of the problems studied in this thesis. Consider the weighted version of the graph covering problems presented in Section 3.2. For the Weighted Dominating Set Problem, suppose we have an instance P of the problem. If we change the weight function $W : V \to \mathbb{R}^+$ and obtain a new instance P', the minor embedding will obviously remain the same for P' but with reconfigured weights on each of the couplers. Hence it effectively reduces the *Embed* time for P' to 0 (assuming the embedding of P was previously computed). With a classical algorithm, it may need to treat P' as an entirely new instance where any information from its previous computation on P will be of little use here.

As mentioned in Section 2.3, the Minor Containment Problem is one of the main issues we need to consider if we want to achieve any kind of speedup using this general approach to problems. We believe that the ideas from the previous paragraph can be used effectively to provide a way in which the result of previous computations can be reused and therefore reduce the overall overhead. Again, the viability of this method has to be verified both theoretically and experimentally. We plan to study this concept of quassical computing further in the future.

5.3 Final comments

Although we did not show that the current hardware can be of any practical use in this thesis, the concepts we provided in Section 2.3 and 5.2 may still be applicable as a general framework to how a quantum computing device can be used in practice. Our study on the Graph Isomorphism Problem and its related variations in Chapter 4 also shows that there is a huge difference on the number of logical (physical) qubits required between a directly formulated QUBO and one obtained through a reduction. The impracticality of these generic approaches motivates the development of more efficient QUBO formulations.

The US White House has published a report recently (see [44]) identifying quantum computing (as a subtopic of Quantum Information Science) as a rapidly evolving field. It has also showed that there has been a steadily increase in the amount of quantum computing related research over the last 15 years. With attentions from both the government and private sectors, it will be interesting to see where the future of quantum computing will lead us.

Bibliography

[1] Alastair A. Abbott, Cristian S. Calude, Michael J. Dinneen, and Richard Hua. Quassical computing with D-Wave, 2016. In preparation.

[2] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM J. Comput.*, 37(1):166–194, 2007.

[3] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, May 2004.

[4] László Babai. Graph isomorphism in quasipolynomial time. Accessed 22 March 2016, 11 December/19 January 2015/2016, http://arxiv.org/abs/1512.03547; see also http://goo.gl/QPTXXY.

[5] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[6] H. G. Barrow and R. M. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4(4):83–84, 1976.

[7] Jun Cai, William G. Macready, and Aidan Roy. A practical heuristic for finding graph minors. *ArXiv e-prints*, June 2014. 2014arXiv1406.2741C.

[8] Cristian S. Calude. Unconventional computing: A brief subjective history. Technical report, Department of Computer Science, The University of Auckland, New Zealand, 2015.

[9] Cristian S. Calude, Elena Calude, and Michael J. Dinneen. Adiabatic quantum computing challenges. *ACM SIGACT News*, 46(1):40–61, March 2015.

[10] Cristian S. Calude and Michael J. Dinneen. Solving the Broadcast Time Problem using a D-Wave quantum computer. In Andy Adamatzky, editor, *Advances in Unconventional Computing*, volume 22 of *Emergence, Complexity and Computation*, chapter 17, pages 439–453. Springer International Publishing Switzerland, 2016.

[11] Cristian S. Calude, Michael J. Dinneen, and Richard Hua. QUBO formulations for the Graph Isomprhism Problem, 2016. In preparation.

[12] D-Wave. Programming with QUBOs. Technical Report 09-1002A-B, D-Wave Systems, Inc., 2013. Python Release 1.5.1-beta4 (for Mac/Linux).

[13] D-Wave. Developer guide for Python. Technical Report 09-1024A-A, D-Wave Systems, Inc., 2016. Python Release 2.3.1 (for Mac/Linux).

[14] D-Wave. Introduction to the D-Wave quantum hardware. http://www.dwavesys.com/tutorials/background-reading-series/introduction-d-wave-quantum-hardware, 2016. D-Wave Systems, Inc. Accessed: 2016-08-30.

[15] D-Wave. The D-Wave 2X$^{\text{TM}}$ System, 2016. http://www.dwavesys.com/d-wave-two-system.

[16] IMB Big Data and Analytics Hub. The four V's of Big Data. http://www.ibmbigdatahub.com/infographic/four-vs-big-data, 2013. Accessed: 2016-08-30.

[17] Vasil S. Denchev, Sergio Boixo, Sergei V. Isakov, Nan Ding, Ryan Babbush, Vadim Smelyanskiy, John Martinis, and Hartmut Neven. What is the computational value of finite range tunneling?, 2015. arXive 1512.02206.

[18] Michael J. Dinneen and Richard Hua. Formulating graph covering problems for Adiabatic Quatumn Computers. Technical Report CDMTCS-495, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, 2016.

[19] Manek Dubash. Moore's Law is dead, says Gordon Moore. http://www.techworld.com/news/operating-systems/moores-law-is-dead-says-gordon-moore-3576581, 2010. Techworld. Accessed: 2016-08-30.

[20] Shimon Even and Oded Kariv. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 100–112. IEEE, 1975.

[21] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*, 292(5516):472–475, 2001.

[22] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. arXiv:quant-ph/0001106, January 2000.

[23] Richard P. Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[24] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)*, 56(5):25, 2009.

[25] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[26] 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.

[27] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[28] Frank Harary et al. Graph theory. *Reading: Addison-Wesley*, 1969.

[29] John Michael Harris, Jeffry L. Hirst, and Michael J. Mossinghoff. *Combinatorics and graph theory*, volume 2. Springer, 2008.

[30] Stephen T. Hedetniemi and Renu C. Laskar. Bibliography on domination in graphs and some basic definitions of domination parameters. *Discrete Mathematics*, 86(1-3):257–277, 1990.

[31] Robert D. Hof. Deep Learning. https://www.technologyreview.com/s/513696/deep-learning, 2013. MIT Technology Review. Accessed: 2016-08-30.

[32] Lei Jin, Zhaokang Wang, Rong Gu, Chunfeng Yuan, and Yihua Huang. Training large scale deep neural networks on the Intel Xeon Phi many-core coprocessor. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 1622–1630. IEEE, 2014.

[33] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[34] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2002.

[35] Marco Lanzagorta and Jeffrey K. Uhlmann. Hybrid quantum-classical computing with applications to computer graphics. In *ACM SIGGRAPH 2005 Courses*, page 2. ACM, 2005.

[36] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, Inc., 1976.

[37] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2(5), 2014.

[38] Enrique Martín-López, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773–776, 2012.

[39] Catherine McGeoch. *Adiabatic Quantum Computation and Quantum Annealing. Theory and Practice,*. Morgan & Claypool Publishers, 2014.

[40] Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[41] Marc Mezard, Giorgio Parisi, Miguel Angel Virasoro, and David J. Thouless. Spin glass theory and beyond. *Physics Today*, 41:109, 1988.

[42] Maryam M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

[43] Robert Z. Norman and Michael O. Rabin. An algorithm for a minimum cover of a graph. *Proceedings of the American Mathematical Society*, 10(2):315–319, 1959.

[44] Interagency Working Group on Quantum Information Science of the Subcommittee on Physical Sciences. Advancing quantum information science: National challenges and opportunities. Technical report, Committee on Science and Committee on Homeland and National Security of the National Science and Technology Council, July 2016.

[45] Marcello Pelillo. Replicator equations, maximal cliques, and graph isomorphism. *Neural Computation*, 11:1933–1955, 1999.

[46] Kristen L. Pudenz, Tameem Albash, and Daniel A. Lidar. Error-corrected quantum annealing with hundreds of qubits. *Nature communications*, 5, 2014.

[47] Geordie Rose and William G. Macready. An introduction to quantum annealing. Technical Report Document 0712, D-Wave Systems, Inc., 2007.

[48] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer Science & Business Media, 2003.

[49] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[50] David Silver and Demis Hassabis. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 2016.

[51] Tom Simonite. Moores Law is dead. Now what? https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what, 2016. MIT Technology Review. Accessed: 2016-08-30.

[52] The Sage Developers. *Sage Mathematics Software (Version 6.5)*, 2015. http://www.sagemath.org.

[53] André Viebke and Sabri Pllana. The potential of the Intel Xeon Phi for supervised deep learning. In *17th IEEE International Conference on High Performance Computing and Communications (HPCC 2015)*, 2015.

[54] M Mitchell Waldrop. The chips are down for Moores law. *Nature News*, 530(7589):144, 2016.

[55] Di Wang and Robert Kleinberg. Analyzing quadratic unconstrained binary optimization problems via multicommodity flows. *Discrete Applied Mathematics*, 157(18):3746–3753, 2009.

[56] Wikipedia. Microprocessor transistor counts 1971 to 2011. https://en.wikipedia.org/wiki/Transistor_count, 2013. Accessed: 2016-08-30.

[57] Wikipedia. Integer Programming. http://en.wikipedia.org/wiki/Integer_programming, 2015. Accessed: 2016-04-07.

Appendix A

Python Program to Generate QUBO Formulation of the Dominating Set Problem

```
import sys, math, networkx as nx

def read_graph():
    n=int(sys.stdin.readline().strip())
    G=nx.empty_graph(n,create_using=nx.Graph())
    for u in range(n):
            neighbors=sys.stdin.readline().split()
        for v in neighbors:
                G.add_edge(u,int(v))
    return G

def generateQUBO(G):
    Q = {}
    numOfRedVars = 0
    # stores the number of redundant variables each vertex has
    redVarsDict = {}
    order = G.order()

    for v in G:
        redVars = int(math.log(nx.degree(G,v),2))+1
        numOfRedVars += redVars
        redVarsDict[v] = redVars

    numOfRedVars = int(numOfRedVars)
    totalNumOfVars = G.order() + numOfRedVars
    redVarsIndexDict = {}

    # compute index of y_i,k in Q
    for v in G:
        temp = 0
        for i in range(v):
            temp += redVarsDict[i]
        redVarsIndexDict[v] = order + temp

    # initialize Q
    for i in range(totalNumOfVars):
        for j in range(totalNumOfVars):
            Q[i,j] = 0

    # pick constant A > 1
    A = 2

    for v in G:
        # (1-A)x_i
        Q[v, v] -= 1

        # -2A sum x_j
        for u in G.neighbors(v):
            Q[u,u] -= 2*A
        # starting index of redundant variables of vertex v in Q
        index = redVarsIndexDict[v]
        # num is the number of redundant variables vertex v has
        num = redVarsDict[v]

        # 2A sum 2^ky_i,k
        for i in range(num):
            temp = int(2*A*math.pow(2,i))
            Q[index+i,index+i] += temp
        # 2A x_i sum x_j
        for u in G.neighbors(v):
            Q[v, u] += 2*A
        # -2A x_i sum 2^ky_i,k
        for i in range(num):
            Q[v,index+i] -= int(2*A*math.pow(2,i))
        # A sum x_j sum x_j
        for u in G.neighbors(v):
            for w in G.neighbors(v):
                Q[u, w] += A
        # -2A sum x_j sum 2^ky_i,k
        for u in G.neighbors(v):
            for i in range(num):
                Q[u, index+i] -= int(2*A*math.pow(2,i))
        # A sum 2^ky_i,k sum 2^ky_i,k
        for i in range(num):
            for j in range(num):
                Q[index+i,index+j] += int(A*math.pow(2,i)*math.pow
(2,j))

    # move all entries to the upper triangle of the matrix
    for i in range(totalNumOfVars):
        for j in range(totalNumOfVars):
            if j > i:
                Q[i,j] += Q[j,i]
                Q[j,i] = 0

    # print a symmetric form of Q
    for i in range(totalNumOfVars):
        for j in range(totalNumOfVars):
            if i<= j:
                print Q[i,j],
            else:
                print Q[j,i],
        print

# main program
G=read_graph()
generateQUBO(G)
```

listings/DS_generate_QUBO.py

Appendix B

Python Program to Generate QUBO Formulation of the Edge Cover Problem

```python
import sys, math, networkx as nx

def generateQUBO(G):

    # map each edge of G to some index in Q
    Q,edgeDict = {},{}
    index = 0
    for (u,v) in G.edges():
        if (u,v) in edgeDict:
            edgeDict[(v,u)] = edgeDict[(u,v)]
        elif (v,u) in edgeDict:
            edgeDict[(u,v)] = edgeDict[(v,u)]
        else:
            edgeDict[(u,v)] = index
            edgeDict[(v,u)] = index
            index+=1

    # compute the index of redundant variables in Q
    size = G.size()
    numOfRedVars = 0
    redVarsDict = {}

    for v in G:
        if nx.degree(G,v) != 1:
            redVars = int(math.log(nx.degree(G,v)-1,2))+1
        else:
            redVars = 0
        numOfRedVars += redVars
        redVarsDict[v] = redVars

    numOfRedVars = int(numOfRedVars)
```

```
totalNumOfVars = G.size() + numOfRedVars
redVarsIndexDict = {}

for v in G:
    temp = 0
    for i in range(v):
        temp += redVarsDict[i]
    redVarsIndexDict[v] = size + temp

# initializing Q
for i in range(totalNumOfVars):
    for j in range(totalNumOfVars):
        Q[i,j] = 0

# pick constant A > 1
A = 2

# sum x_i,j
for e in G.edges():
    Q[edgeDict[e],edgeDict[e]] = 1

# sum P_i
for v in G.nodes():

    # I is the set of edges incident to v
    I = G.edges(v)

    # -2A sum x_i,j
    for e in I:
        Q[edgeDict[e],edgeDict[e]] -= 2*A

    # starting index of redundant variable corresoponding to v
in Q
    index = redVarsIndexDict[v]
    # num is the number of redundant variables vertex v has
    num = redVarsDict[v]

    # 2A sum 2^k y_i,k
    for k in range(num):
        Q[index+k, index+k] += int(2*A*math.pow(2,k))
    # A sum x_i,j sum x_i,j
    for e1 in I:
        for e2 in I:  # -2A sum x_i,j sum 2^k y_i,k
            Q[edgeDict[e1],edgeDict[e2]] += A
    for e in I:
        for k in range(num):
            Q[edgeDict[e],index+k] -= int(2*A*math.pow(2,k))
    # A sum 2^k y_i,k sum 2^k y_i,k
    for k1 in range(num):
      for k2 in range(num):
          Q[index+k1,index+k2] += A*int(math.pow(2,k1)*math.pow
(2,k2))
```

```
    # move all entries to the upper triangle of the matrix
    for i in range(totalNumOfVars):
        for j in range(totalNumOfVars):
            if j > i:
                Q[i,j] += Q[j,i]
                Q[j,i] = 0

    # print a symmetric form of Q
    for i in range(totalNumOfVars):
        for j in range(totalNumOfVars):
            if i<= j:
                print Q[i,j],
            else:
                print Q[j,i],
        print

G=read_graph()
result = generateQUBO(G)
```

listings/EC_generate_QUBO.py

Appendix C

Sage Math Program to Compute the Exact Solution to the Dominating Set Problem

```
import sys, networkx as nx

def read_graph():
    n=int(sys.stdin.readline().strip())
    G=nx.empty_graph(n,create_using=nx.Graph())
    for u in range(n):
        neighbors=sys.stdin.readline().split()
        for v in neighbors: G.add_edge(u,int(v))
    return G

G=read_graph()
n=G.order()
p=MixedIntegerLinearProgram(solver="GLPK", maximization=False)
x=p.new_variable(binary=True)

for v in G.nodes():
    c = x[v]
    for u in G.neighbors(v):
        c = c + x[u]
    p.add_constraint(c >= 1)
p.set_objective(sum(x[j] for j in range(n)))
try:
    sz=p.solve()
except sage.numerical.mip.MIPSolverException as e:
    pass
else:
    pass
    print "Minimum dominating set is", int(sz)
    for i in p.get_values(x).items():
        print i
```

listings/DS_sage.py

Appendix D

Python Program to Scale the Ising Instances

```
# QUBO (with embedding) -> Scaled Ising -> DWave

import sys, time, math, traceback

from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.util import get_hardware_adjacency
from dwave_sapi2.embedding import embed_problem, unembed_answer
from dwave_sapi2.util import qubo_to_ising
from dwave_sapi2.core import solve_ising

# coupler strength for embedded qubits of same variable
s,s2=0.9,1.0
if (len(sys.argv)==2): s = float(sys.argv[1])
if (len(sys.argv)==3): s,s2 = float(sys.argv[1]),float(sys.argv[2])
print 'Embed scale:',s,s2
assert 0 <= s <= 1

# read input QUBO
line=sys.stdin.readline().strip().split()
n=int(line[0])
print 'Logical qubits used=', n

Q = {}
for i in range(n):
    line=sys.stdin.readline().strip().split()
    for j in range(n):
        t = float(line[j])
        if j>=i and t!=0: Q[(i,j)]=t

# convert to Ising
(H,J,ising_offset) = qubo_to_ising(Q)
print 'orig H=',H
print 'orig J=',J
print 'ising_offset=',ising_offset
```

```
# scale by maxV and s2
if len(H): maxH=max(abs(min(H)),abs(max(H)))
else:      maxH=0.0
maxJ=max(abs(min(J.values())),abs(max(J.values())))
maxV=max(maxH,maxJ)

for i in range(n):
    if len(H)>i: H[i]=s2*H[i]/maxV
    for j in range(n):
        if j>=i and (i,j) in J:
            J[(i,j)]=s2*J[(i,j)]/maxV
print 'scaled H=',H
print 'scaled J=',J

# read minor embedding
embedding=eval(sys.stdin.readline())
print 'embedding=', embedding
print 'Physical qubits used= %s' % sum(len(embed) for embed in
   embedding)

# create a remote connection using url and token and connect to
   solver
print 'Attempting to connect to network...'
remote_connection = RemoteConnection(url, token)
solver = remote_connection.get_solver(solver_name)
#print 'Solver properties:\n%s\n' % solver.properties
A = get_hardware_adjacency(solver)

# Embed problem into hardware and scale non-minor couplers by s
(h0, j0, jc, new_emb) = embed_problem(H, J, embedding, A)
h1= [val*s for val in h0]
j1 = {}
for (key, val) in j0.iteritems():
    j1[key]=val*s
j1.update(jc)
print 'h1=',h1
print 'j1=',j1

# call the dwave solver
annealT=20 # annealing_time_range = [20, 2000]
progT=500  # programming_thermalization_range = [0,10000]
readT=10   # readout_thermalization_range = [0,10000]

print 'annealT=',annealT,'progT=',progT,'readT=',readT
result = solve_ising(solver, h1, j1, num_reads=100, annealing_time=
   annealT, programming_thermalization=progT, readout_thermalization
   =readT)
print 'result:', result

newresult = unembed_answer(result['solutions'], new_emb,
   broken_chains='discard', h=H, j=J)
```

```
#newresult = unembed_answer(result['solutions'], new_emb,
   broken_chains='vote', h=H, j=J)
#newresult = unembed_answer(result['solutions'], new_emb,
   broken_chains='minimize_energy', h=H, j=J)
print 'newresult:', newresult

# print unembed solutions in QUBO format
for i, (embsol, sol) in enumerate(zip(result['solutions'], newresult
   )):
    print "solution %d:" % i,
    for j, emb in enumerate(embedding):
        if sol[j]==-1: print "0",
        if sol[j]==1: print "1",
    print
```

listings/scale_Ising.py

Appendix E

Python Program to Generate the Direct QUBO Formulation of the Graph Isomorphism Problem

```python
# direct QUBO formulation for graph isomorphism problem

import networkx as nx
import sys, math

# G1 and G2 are of type graph defined by the networkX library
def generateQUBO(G1, G2):
    n = G1.order()
    varsDict = {}
    edgeDict = {}

    # compute constants e_i,j
    for i in range(n):
        for j in range(n):
            if ((i,j) in G2.edges()) or ((j,i) in G2.edges()):
                edgeDict[i,j] = 1
                edgeDict[j,i] = 1
            else:
                edgeDict[i,j] = 0
                edgeDict[j,i] = 0

    # map each variable to an index in Q
    index = 0
    for i in range(n):
        for j in range(n):
            varsDict[(i,j)] = index
            index += 1

    # initialize Q
    Q = {}
    for i in range(n*n):
        for j in range(n*n):
            Q[i,j] = 0

    # HA part 1
    for i in range(n):
        for iprime in range(n):
            index = varsDict[(i,iprime)]
            Q[index,index] -= 2

        for iprime1 in range(n):
            for iprime2 in range(n):
                index1 = varsDict[(i,iprime1)]
                index2 = varsDict[(i,iprime2)]
                Q[index1, index2] += 1
    # HA part 2
    for iprime in range(n):
        for i in range(n):
            index = varsDict[(i,iprime)]
            Q[index,index] -= 2

        for i1 in range(n):
            for i2 in range(n):
                index1 = varsDict[(i1,iprime)]
                index2 = varsDict[(i2,iprime)]
                Q[index1, index2] += 1

    # Pij
    for (i,j) in G1.edges():
        for iprime in range(n):
            xiiprime = varsDict[(i,iprime)]
            for jprime in range(n):
                xjjprime = varsDict[(j,jprime)]
                Q[xiiprime,xjjprime] += (1-edgeDict[iprime,jprime])

    # Making Q uppertriangular
    for i in range(n*n):
        for j in range(n*n):
            if (i > j) and (not(Q[i,j]==0)):
                Q[j,i] += Q[i,j]
                Q[i,j] = 0


    return Q
```

listings/generateISOQUBO.py

Appendix F

Python Program to Generate Graph Product

```python
# create Psi product graph (iso -> clique construction)
# input graphs as two command-line argument filenames or 'cat' two
   graphs from stdin

import sys, networkx as nx

def read_graph(infile=sys.stdin):
    n=int(infile.readline().strip())
    G=nx.empty_graph(n,create_using=nx.Graph())
    for u in range(n):
        neighbors=infile.readline().split()
        for v in neighbors: G.add_edge(u,int(v))
    return G

#print(sys.argv)

if len(sys.argv)==3:
    gfile=open(sys.argv[1],'r')
    G1=read_graph(gfile)
    gfile.close()
    gfile=open(sys.argv[2],'r')
    G2=read_graph(gfile)
    gfile.close()
elif len(sys.argv)==2:
    gfile=open(sys.argv[1],'r')
    G1=read_graph(gfile)
    gfile.close()
    G2=read_graph()
else:
    G1=read_graph()
    G2=read_graph()

n1=G1.order()
n2=G2.order()
```

```
n=n1*n2
P=nx.empty_graph(n,create_using=nx.Graph())

for a in range(n1):
    for b in range(n2):
        for c in range(n1):
            if a==c: continue
            for d in range(n2):
                if b==d: continue
                if (a in G1[c])==(b in G2[d]): P.add_edge(a*n2+b,c*
   n2+d)

print(n)
for u in range(n):
    for v in P[u]:
        print(v,end=' ')
    print()
```

listings/psiproduct.py

Appendix G

Python Program to Generate QUBO Formulation of the Clique Problem

```
# Max Clique graph to QUBO Hamiltonian objective

import sys, networkx as nx

def read_graph(infile=sys.stdin):
    n=int(infile.readline().strip())
    G=nx.empty_graph(n,create_using=nx.Graph())
    for u in range(n):
        neighbors=infile.readline().split()
        for v in neighbors: G.add_edge(u,int(v))
    return G

if len(sys.argv)==2:
    gfile=open(sys.argv[1],'r')
    G=read_graph(gfile)
    gfile.close()
else:
    G=read_graph()

n=G.order()

Q = {}
print(n)
for u in range(n):
    for v in range(n):
        if u > v:
            Q[u,v] = 0
        elif u==v:
            Q[u,v] = -1
        elif u in G[v]:
            Q[u,v] = 0
        else:
            Q[u,v] = 2

for u in range(n):
    for v in range(n):
        print(Q[u,v], end=' ')
    print()
```

listings/Clique2QUBO.py

Appendix H

Python Program to Generate Chimera Graphs

```
import sys
from dwave_sapi import get_chimera_adjacency
import networkx as nx

[m,n,l]=[int(sys.argv[i]) for i in [1,2,3]]

A=get_chimera_adjacency(m,n,l)
#print A

order=n*m*l*2
G = nx.empty_graph(order)
G.add_edges_from(A)

print order
for i in range(order):
    for j in G[i]: print j,
    print
```

listings/chimera_graph.py