CDMTCS
Research
Report
Series

Improved QUBO Formulation of the Graph Isomorphism Problem

Richard Hua
Michael J. Dinneen
Schoolof Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-539
June 2019

Centre for Discrete Mathematics and
Theoretical Computer Science

Improved QUBO Formulation of the Graph Isomorphism Problem

Richard Hua and Michael J. Dinneen
School of Computer Science, University of Auckland, Auckland, New Zealand

June 10, 2019

Abstract

In this paper we provide a practically efficient QUBO formulation for the Graph Isomorphism Problem that is suitable for quantum annealers, such as those produced by D-Wave. After proving correctness of our new method, based on exploiting vertex degree classes, we do some experimental work on a D-Wave 2X computer. We observe that for all "hard" graphs of 6 vertices, we save around 50% to 95% of the number of required qubits over the standard QUBO formulation that was given earlier by Calude *et al* [13]. We also provide some theoretical analysis showing that for two random graphs with the same degree sequence our new method substantially improves in qubit savings as the number of vertices increases beyond 6.

1 Introduction

The Graph Isomorphism Problem is the computational problem of determining whether two graphs are isomorphic. Given 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-invariant vertex bijection $f : V_1 \to V_2$ such that for every pair of vertices $\{u, v\}$, we have $uv \in E_1$ if and only if $f(u)f(v) \in E_2$. Formally, we define the problem below:

Graph Isomorphism Problem:

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

We assume that the input graphs G_1 and G_2 are both connected graphs; if they have multiple connected components then we can always reduce to smaller instances of the Graph Isomorphism Problem. We will also assume G_1 and G_2 have the same degree sequence, they can not be isomorphic otherwise and these cases can be decided quickly.

The Graph Isomorphism Problem has numerous applications in a variety of fields ranging from the study of chemical compound structure [19] and computational biology [34] to security in social networks [35]. The exact complexity of the problem has eluded researchers ever since its conception for more than five decades. On one hand, the Graph Isomorphism Problem is clearly in NP; given a vertex mapping $f : V_1 \to V_2$, one can easily check in polynomial time if f is bijective and edge-invariant. On the other hand, the Graph Isomorphism Problem seems to be neither in P nor NP-complete. The current best known classical algorithm for the problem by [4] has a time complexity of $\exp\left((\log n)^{O(1)}\right)$ which is quasipolynomial. If the Graph Isomorphism Problem is NP-complete, it would have huge implication on complexity theory. It has been proven that if the Graph Isomorphism Problem is NP-complete, then the polynomial-time hierarchy collapses [3] which is strong evidence that P $\neq$ NP. However, despite the similarities between the Graph Isomorphism Problem and many NP-complete problems [27], no one has been able to prove that the Graph Isomorphism Problem is NP-complete. This makes the Graph Isomorphism Problem one of the most interesting problems in complexity theory since not many other problems have the aforementioned properties.

The concept of adiabatic quantum computing is based on the Adiabatic Theorem in physics [21, 29] which involves the process of evolving the ground state (state of minimum energy) of a physical system [20, 21]. It has attracted much attention from both researchers and private sectors over the last few years. One advantage of adiabatic quantum computing over the more traditional quantum gate model is fact that a particular type of physical device that can be used for adiabatic quantum computing, known as quantum annealer, is relatively easier to build compared to quantum gate machines. The Canadian company D-Wave Systems was able to manufacture the D-Wave 2000Q$^{\text{TM}}$ with more than 2000 physical qubits [14] which is a huge improvement when compared with state-of-the-art quantum gate model device such as the IBM Q 20 [24], which only has 20 physical qubits. Note that physical qubits in an adiabatic quantum computing framework are not equivalent to qubits in the quantum gate framework. To simulate an arbitrary quantum gate computation, the adiabatic framework requires more qubits in general [2]. Furthermore, D-Wave quantum annealers can only support a specific type of problem structure. This means that an algorithm developed for a given problem using a set number of logical of physical qubits has to be 'embedded' in the hardware first, which normally cause the number of physical qubits required to increase dramatically.

Motivated by the limited number of physical qubits in current hardware, we present an improved version of the QUBO formulation given in [13] and empirically compare their performance. Our experimental results indicate that the improved version is much more suitable for current hardware not only in terms of better embedding but also has a higher probability of obtaining the correct answer when run on a quantum annealer.

The rest of the paper is organized as follows. In Section 2, we will provide the necessary mathematical background knowledge required. In Section 3, we present the improved QUBO formulation along with a proof of correctness. In Section 4, we will first provide an overview of experiment conducted followed by the embedding experiment on an active D-Wave 2X hardware. We also give a brief explanation on how an D-Wave quantum annealer can be used to solved QUBO problems followed by some comments on how to configure some of the parameters and options supported by D-Wave 2X. A discussion of the experiment results are provided in Section 5.

2 Mathematical Prerequisites

In this section we introduce the mathematical and graph theory prerequisites that are useful for this paper.

The cardinality of a set X is denoted by $|X|$. By lg we denote the logarithm in base 2 and $Z_2 = \{0, 1\}$. The power set of X will be denoted by 2^X. A partial function $f : X \to Y$ is a function which can be undefined for some values $x \in X$. The domain of f, denoted by $dom(f)$ is the set of all $x \in X$ for which $f(x)$ is defined. A graph $G = (V, E)$ consists of a finite non-empty set of vertices V together with a set of edges E. The order of G, denoted by n, is the number of vertices in V. In our representation vertices are labeled by $V = \{v_i \mid 0 \leq i < n\}$. The set (of edges) E consists of unordered pairs of vertices $u, v \in V$; we denote an edge by $e = uv$. Note that since we only consider undirected graphs in this paper, uv and vu represent the same edge. The number of edges, denoted by m, is called the size of G. For a vertex v in a graph, the degree of v, denoted by $deg(v)$ is the number of neighbors v has. That is, $deg(v) = |\{u \mid uv \in E\}|$. A regular graph is a graph $G = (V, E)$ such that $deg(u) = deg(v)$ for all $u, v \in V$. The degree sequence of a graph $G = (V, E)$ is the monotonic nonincreasing sequence of the vertex degrees of G.

In this paper we will only consider simple graphs, that is, graphs with no multi-edges nor self-loops. The first condition means that for all pairs of vertices u and v, there is at most one edge between u and v; the second condition states that for every vertex $v \in V$ we have $vv \notin E$.

The following well-known theorem is also useful.

Theorem 1. *Given two graphs G_1 and G_2. G_1 and G_2 are isomorphic if and only if the complement of G_1 and G_2 are isomorphic.*

Quadratic Unconstrained Binary Optimization (QUBO) is an NP-hard optimization problem involving the minimization of a quadratic objective function $F : Z_2^n \to \mathbb{R}$ that is defined by an $n \times n$ upper-triangular matrix Q. Let $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ be a vector of n binary variables, the objective function is of the following form $F(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$. Formally the QUBO problem is defined as follows:

$$x^* = \min_{\mathbf{x}} \sum_{i \leq j} x_i Q_{(i,j)} x_j, \text{ where } x_i \in \mathbb{Z}_2. \tag{1}$$

Note that for notational convenience, the entries of Q are indexed from 0 (i.e. $0 \leq i, j < n$). The goal is to find a binary value assignment of variables $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ such that the value of $F(\mathbf{x})$ is minimum. We use x^* to denote the minimum value of $F(\mathbf{x})$ and $\mathbf{x}^* = (x_0^*, x_1^*, \ldots, x_{n-1}^*)$ to denote the set of value assignments of the n variables that yield x^*.

Although this paper focuses on the QUBO model, some relevant knowledge of the Ising spin glass model will also be useful since the D-Wave quantum annealers implement the Ising model. The Ising model is a problem in physics that involves the mechanics of ferromagnetism [10], we will not be overly concerned with the physical nature of the problem and will only provide an abstract mathematical description of the model. Let $G = (V, E)$ be a graph where each vertex $i \in V$ and edge $ij \in E$ are associated with real values h_i and $J_{(i,j)}$

respectively. An Ising Minimization Problem with $n = |V|$ variables has a variable vector $\mathbf{x} = (x_0, x_1, \dots, x_{n-1})$ and can be defined as follows:

$$x^* = \min_{\mathbf{x}} \sum_{ij \in E} x_i J_{(i,j)} x_j + \sum_{i \in V} h_i x_i, \text{ where } x_i \in \{-1, 1\}. \tag{2}$$

The QUBO and Ising problems are equivalent, they are both NP-hard [5, 33] and one can easily transform one to the other with a relatively simple transformation function [16]. This paper will mainly focus on the QUBO model since binary values are more natural and interpretive than the ± 1 model.

3 QUBO Formulation for the Graph Isomorphism Problem and Improvements

For the sake of completeness, we will first introduce the QUBO formulation for the Graph Isomorphism Problem developed in [13]. Note that we assume that the two input graphs G_1 and G_2 have the same order and size. Now suppose $G_1 = (V_1, E_2)$ and $G_2 = (V_2, E_2)$ are two graphs both of order n and size m. The objective function of [13] requires a total of n^2 logical qubits, one variable $x_{i,j}$ for each integer pair i and j where $0 \le i < n$ and $0 \le j < n$. If $x_{i,j} = 1$ then the vertex mapping maps v_i in G_1 to v_j in G_2. We label the collection of n^2 variables by a binary vector $\mathbf{x} \in \mathbb{Z}_2^{n^2}$:

$$\mathbf{x} = [x_{0,0}, x_{0,1}, \dots x_{0,n-1}, x_{1,0}, x_{1,1}, \dots, x_{1,n-1}, \dots x_{n-1,0}, \dots x_{n-1,n-1}].$$

We also pre-compute n^2 binary constants $e_{i,j}$ for $0 \le i < n$ and $0 \le j < n$ where $e_{i,j} = 1$ if $ij \in E_2$ and $e_{i,j} = 0$ if $ij \notin E_2$.

The objective function in [13] is of the following form:

$$F(\mathbf{x}) = H(\mathbf{x}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}), \tag{3}$$

where

$$H(\mathbf{x}) = \sum_{0 \le i < n} \left(1 - \sum_{0 \le i' < n} x_{i,i'}\right)^2 + \sum_{0 \le i' < n} \left(1 - \sum_{0 \le i < n} x_{i,i'}\right)^2, \tag{4}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{0 \le i' < n} \left(x_{i,i'} \sum_{0 \le j' < n} x_{j,j'} (1 - e_{i',j'}) \right). \tag{5}$$

Now suppose x^* and $\mathbf{x}^*$ are the optimal value and its corresponding binary variable assignment of Equation (3), one can decide if whether G_1 and G_2 are isomorphic and compute the edge-invariant, if it exists, efficiently. See [13] for the proof of correctness and more details.

Clearly, if G_1 and G_2 are indeed isomorphic then an edge-invariant vertex mappings can only map vertices in G_1 to vertices of the same degree in G_2 (i.e. if $u = f(v)$ then $deg(u) = deg(v)$). Based on this fact, we can quickly eliminate unnecessary logical variables which can not have value 1. This is important from a practical point of view since the current D-Wave hardware only have a limited number of physical qubits and couplers available, hence reducing the number of required logical qubits means that we can solve larger instances of the Graph Isomorphism Problem. This also makes the problem easier to embed in the hardware.

The improved QUBO formulation still requires $O(n^2)$ qubits in the worst case but substantially less in practice; the exact number depends on the degree sequence of the input graphs. Formally, for each vertex pair (v_i, v_j) where $v_i \in V_1$ and $v_j \in V_2$, we define a binary variable $x_{i,j}$ if $deg(v_i) = deg(v_j)$. For readability, we define a set $S = \{(i,j) \mid deg(v_i) = deg(v_j) \text{ for } v_i \in V_1 \text{ and } v_j \in V_2\}$ and the collection of all binary variables can be indexed by a binary vector $\mathbf{x} = [x_{i,j} \mid (i,j) \in S]$. Formally, the objective function is of the following form:

$$F(\mathbf{x}) = H(\mathbf{x}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}), \tag{6}$$

where

$$H(\mathbf{x}) = \sum_{0 \le i < n} \left(1 - \sum_{(i,i') \in S} x_{i,i'}\right)^2 + \sum_{0 \le i' < n} \left(1 - \sum_{(i,i') \in S} x_{i,i'}\right)^2, \tag{7}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{(i,i') \in S} \left(x_{i,i'} \sum_{(j,j') \in S} x_{j,j'}(1 - e_{i',j'})\right). \tag{8}$$

As in [13], assume that $x^* = \min_{\mathbf{x}} F(\mathbf{x})$. Then, the mapping f can be 'decoded' from the values of the variables $x_{i,i'}$ using an additional partial function D. Let $\mathcal{F}$ be the set of all bijections between V_1 and V_2. Then $D : \mathbb{Z}_2^{|\mathbf{x}|} \to \mathcal{F}$ is a partial 'decoder' function that re-constructs the vertex mapping f from the vector $\mathbf{x}$, if such f exists. The domain of D contains all vectors $\mathbf{x} \in \mathbb{Z}_2^{|\mathbf{x}|}$ that can be 'decoded' into a bijective function f:

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

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 $x_{i,i'} = 1$.

Although Equation (6) is very similar to the objective function presented in [13], we present a proof of correctness here for the sake of completeness.

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

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

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

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,

$$\sum_{0 \le i < n} \left(1 - \sum_{(i,i') \in S} x_{i,i'}\right)^2 = 0 \tag{9}$$

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

Second, with the same argument,

$$\sum_{0 \le i' < n} \left(1 - \sum_{(i,i') \in S} x_{i,i'}\right)^2 = 0 \tag{10}$$

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

Together the conditions (9) and (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 3. *Let $\mathbf{x} \in \mathbb{Z}_2^{|\mathbf{x}|}$ 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 $\mathbf{x} \in \mathbb{Z}_2^{|\mathbf{x}|}$. Note that $P_{i,j}(\mathbf{x})$ from Equation (6) does not contain cubic terms, so, as all $e_{i',j'}$ are constants, $P_{i,j}(\mathbf{x})$ contains only quadratic terms; 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 Equation (8), we get

$$P_{i,j}(\mathbf{x}) = \sum_{(i,i') \in S} x_{i,i'} \left(x_{j,j'_0}(1 - e_{i',j'_0}) + x_{j,j'_1}(1 - e_{i',j'_1}) + \cdots + x_{j,j'_k}(1 - e_{i',j'_k})\right) \text{ for each } (j,j') \in S.$$

Since f is a bijection, for every edge $ij \in E_1$, in the set $\{x_{i,i'} \mid (i,i') \in S\}$ there is a unique variable, denoted by $x^*_{i,i'}$, with value 1, and in the set $\{x_{j,j'} \mid (j,j') \in S\}$ 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 a corresponding edge $f^{-1}(i)f^{-1}(j) \in E_1$, so f is edge-invariant. $\square$

Theorem 4. *For all $\mathbf{x} \in \mathbb{Z}_2^{|\mathbf{x}|}$, $F(\mathbf{x}) = 0$ 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 2 and 3, f must be bijective and edge-invariant.

On the other hand, if $F(\mathbf{x}) \neq 0$, then 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 2. If $H(\mathbf{x}) = 0$ and $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) \neq 0$, then by Lemma 3 the mapping is not edge-invariant. $\square$

3.1 An example: the graph P_3

Recall the example given in [13]. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs of order 3, where $V_1 = V_2 = \{0, 1, 2\}$ and $E_1 = \{\{0, 1\}, \{1, 2\}\}$ and $E_2 = \{\{0, 1\}, \{0, 2\}\}$. The standard QUBO from [13] has nine variables and the binary variable vector is

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

The complete QUBO matrix is given in Table 1. Note that we always ignore constant terms in Equation (3) and (6) when using the matrix based representation, removing constant values will not affect the optimality of $\mathbf{x}^*$ and x^*. Now, there is only one vertex of degree 2 in either graph, vertex 1 in G_1 has to be mapped to vertex 0 in G_2 and it can be verified that the only two optimal solutions are $\mathbf{x}_1 = (0, 1, 0, 1, 0, 0, 0, 0, 1)$ and $\mathbf{x}_2 = (0, 0, 1, 1, 0, 0, 0, 1, 0)$. See [13] for the complete set of constraints for this particular example.

Table 1: Standard 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

The objective function (6) takes advantage of the fact that vertex 1 in G_1 has to be mapped to vertex 2 in G_2 by an edge-invariant vertex mapping. See Table 2 for the improved

QUBO matrix. According to the QUBO formulation given in Equation 6, the set S consists of the following 2-tuples of five binary variables:

$$S = \{(0,1), (0,2), (1,0), (2,1), (2,2)\}$$

and so the binary variable vector is now

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

After expanding Equation (7), we get

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

Using the definition of Equation (6), 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 these constants into Equation (8), we obtain the following two constraints:

$$P_{0,1}(\mathbf{x}) = (x_{0,1}x_{1,0}(1-e_{1,0})) + (x_{0,2}x_{1,0}(1-e_{2,0})) ,$$

$$P_{1,2}(\mathbf{x}) = x_{1,0}\left(x_{2,1}(1-e_{0,1}) + x_{2,2}(1-e_{0,2})\right) .$$

Note that both penalties terms completely vanish if we substitute the corresponding value of $e_{i,j}$ into them.

Once again, it is fairly easy to verify that the two optimal solutions are now $\mathbf{x}_1 = (0,1,1,1,0)$ and $\mathbf{x}_2 = (1,0,1,0,1)$.

Table 2: Degree sequence mapped QUBO matrix for P_3

Variables	$x_{0,1}$	$x_{0,2}$	$x_{1,0}$	$x_{2,1}$	$x_{2,2}$
$x_{0,1}$	-2	2	0	2	0
$x_{0,2}$		-2	0	0	2
$x_{1,0}$			-2	0	0
$x_{2,1}$				-2	2
$x_{2,2}$					-2

4 Experiments

We conducted numerous experiments on a D-Wave 2X quantum annealer. The hardware structure of the D-Wave computers are called Chimera graphs, indexed by three integers (M, N, L), they consist of M by N blocks of interconnected $K_{L,L}$ (complete bipartite graphs of order $2L$). A more detailed specification of the Chimera graph can be found in [1]. We experimentally compare the efficiency of the QUBO formulation in Section 3 and of [13] by embedding the QUBO instances of both formulation on actual D-Wave hardware and experimentally test the solvability of both formulation on the quantum annealer. In this section, we will first provide an overview followed by a summary of the experiments.

4.1 Test cases and minor embeddings

We experimentally measured the improvement of the QUBO formulation given in Section 3 on graphs of small order. It is common knowledge that the number of labeled graphs of order n is $2^{\binom{n}{2}}$ which becomes intractable very quickly when $n > 7$. For example, see `http://oeis.org/A000088` for the number of graphs with order up to 19. However, not all graphs are difficult; for example, consider an instance of the Graph Isomorphism Problem when the input graphs have multiple connected components then we can reduce it to several smaller instances of the Graph Isomorphism Problem and combine their solutions to solve the original instance. Furthermore, the Graph Isomorphism Problem is in P if G_1 and G_2 are trees (since the Tree Isomorphism Problem is in logspace [25]). Together with Theorem 1, it means that the Graph Isomorphism Problem can be efficiently solved if either G_1 and G_2 or their complements are trees. As a result, we tested all the graphs G with the following properties: 1) both G and its complement are connected; 2) neither G nor its complement is a tree. We also filtered out degree sequences that only corresponds to one unique graph under these two conditions.

Our Script A is a Sage [32] program that enumerates all degree sequences of graphs of order 6, it then filters out graphs as described in the previous paragraph and outputs all selected graphs. Now, say a given degree sequence has k graphs selected $G_0, G_1, \ldots, G_{k-1}$. For each $0 \leq i < k$, we generate several instances of the Graph Isomorphism Problem by doing the following: first, we select G_i as the input graph G_1 of the Graph Isomorphism Problem and then, for each j in range $i \leq j < k$, we then randomly permute the vertex labels of G_j and use it as input graph G_2 (see Script B).

In order to compare the improvement of the QUBO instances, we generate two QUBOs for each graph pair, one based on the QUBO formulation given in [13] and one based on Equation (6). The Python Script C contains the implementation of both QUBO formulations. We then tested the embeddings of the QUBOs on a D-Wave 2X quantum computer. The hardware structure of this particular model is a $12 \times 12 \times 4$ Chimera graph which consists of 12 by 12 grids of interconnected complete bipartite graphs $K_{4,4}$. Note that the actual device we used has 54 inactive qubits. The minor embedding algorithm is provided by the software package developed by D-Wave [15], the same algorithm used in [13]. More details about the embedding algorithm can be found in [11]. Note that some of our Python programs use NetworkX [22] as well.

Since the embedding algorithm used here is probabilistic and heavily relies on an initial random vertex selection, it is fairly difficult to predict its behavior. Hence each test case is embedded five times and Tables 3 and 4 contain these embeddings results. Each test case is indexed by three integers i, j and k. The first integer is the degree sequence index generated using the Sage degree sequence generator (see Script A). The other two integers j and k correspond to the indices of selected graphs in the degree sequence (see Script B). For example, *seq_0_0_1* corresponds to the first degree sequence where G_1 and G_2 were chosen to be the first and the second graph outputed by Script A, corresponding to that degree sequence. The column labeled 'Logical Qubits' is the number of logical variables required by the QUBO formulation, the QUBOs generated by the standard QUBO formulation [13] (Table 3) all require $n^2 = 6^2 = 36$ logical qubits whereas the improved degree sequence mapping QUBOs (Table 4) each requires a different number depending on the degree sequence. With respect to the minor embedding, the two most important factors are the total number of physical qubits used and the map size (number of physical qubits each logical qubit is mapped to), so the average map size $\left(\frac{\text{no. physical qubits}}{\text{no. logical qubits}}\right)$ is also computed and presented in Tables 3 and 4.

4.2 D-Wave experiments

Several experiments were ran on an actual D-Wave 2X hardware to test the quality of the QUBO instances generated. Since we only have limited access to the quantum computer, it is not possible to test the viability of all QUBO instances in combination with all different embeddings. Previous experimental works such as [1, 18] and [30][1] suggest that large map size in general will lead to a low probability of solving the QUBO instance with a quantum annealer. Therefore each QUBO instance was executed with an embedding that has the lowest average map size. Note that due to the random nature of the embedding algorithm, the embeddings selected are not likely to be the best embeddings (i.e. other embeddings with a smaller average map size may exist) and different embeddings could affect the result drastically.

Recall the definition of objective function (6) and Theorem 4. If G_1 and G_2 are not isomorphic, then the value of the optimal solution of $F(\mathbf{x})$ is meaningless. As a result, only isomorphic pairs of graphs are used in the experiment (these are test cases labeled by seq_i_j_k where $j = k$). The correctness of the D-Wave answers can be verified efficiently for these test cases as well, one only need to compute $F(\mathbf{x})$ and if $F(\mathbf{x}) = 0$ then $\mathbf{x}$ is the correct optimal solution.

The D-Wave quantum annealing hardware uses the Ising model so the QUBO instances has to be converted to their corresponding Ising form. The transformation operation is relatively straightforward so we will omit the details here (see [16] for the details). This function is implemented in the D-Wave software package [15] and so all QUBO instances are converted to Ising form before D-Wave hardware is used to solve the Ising instance. See Script D for the complete implementation, the Python program reads a QUBO in the format outputed by Script B, convert the QUBO instance to an Ising. It then embeds the Ising instance on the hardware structure of the D-Wave 2X, note that the actual embeddings used in the experiment are not included in this paper since these precomputed embeddings are unlikely to be reusable as each current D-Wave computer has a different set of faulty physical qubits.

[1]Note that this publication refers to map size as 'chain length' which could be somewhat misleading since the set of physical qubits is not necessarily a path.

Table 3: Standard QUBO embedding result (1 of 2).

Graph name	Logical Qubits	Physical 1	Average 1	Physical 2	Average 2	Physical 3	Average 3	Physical 4	Average 4	Physical 5	Average 5
seq_0_0_0	36	516	14.33	482	13.39	682	18.94	399	11.08	515	14.31
seq_0_0_1	36	532	14.78	480	13.33	501	13.92	467	12.97	514	14.28
seq_0_1_1	36	506	14.06	538	14.94	453	12.58	480	13.33	424	11.78
seq_1_0_0	36	587	16.31	464	12.89	462	12.83	463	12.86	506	14.06
seq_1_0_1	36	511	14.19	454	12.61	539	14.97	483	13.42	472	13.11
seq_1_0_2	36	432	12.0	483	13.42	442	12.28	525	14.58	480	13.33
seq_1_1_1	36	399	11.08	544	15.11	478	13.28	502	13.94	461	12.81
seq_1_1_2	36	496	13.78	460	12.78	502	13.94	450	12.5	464	12.89
seq_1_2_2	36	531	14.75	546	15.17	441	12.25	504	14.0	559	15.53
seq_2_0_0	36	471	13.08	514	14.28	478	13.28	483	13.42	506	14.06
seq_2_0_1	36	472	13.11	482	13.39	490	13.61	554	15.39	484	13.44
seq_2_0_2	36	574	15.94	557	15.47	479	13.31	533	14.81	476	13.22
seq_2_0_3	36	609	16.92	472	13.11	487	13.53	465	12.92	461	12.81
seq_2_1_1	36	486	13.5	490	13.61	498	13.83	444	12.33	383	10.64
seq_2_1_2	36	454	12.61	442	12.28	475	13.19	395	10.97	472	13.11
seq_2_1_3	36	557	15.47	576	16.0	411	11.42	531	14.75	502	13.94
seq_2_2_2	36	495	13.75	499	13.86	422	11.72	472	13.11	423	11.75
seq_2_2_3	36	440	12.22	497	13.81	517	14.36	521	14.47	498	13.83
seq_2_3_3	36	459	12.75	451	12.53	535	14.86	496	13.78	498	13.83
seq_3_0_0	36	574	15.94	495	13.75	445	12.36	542	15.06	447	12.42
seq_3_0_1	36	512	14.22	493	13.69	459	12.75	518	14.39	501	13.92
seq_3_0_2	36	536	14.89	571	15.86	454	12.61	465	12.92	493	13.69
seq_3_0_3	36	436	12.11	515	14.31	566	15.72	425	11.81	505	14.03
seq_3_1_1	36	512	14.22	472	13.11	485	13.47	539	14.97	472	13.11
seq_3_1_2	36	517	14.36	417	11.58	460	12.78	521	14.47	425	11.81
seq_3_1_3	36	420	11.67	580	16.11	485	13.47	531	14.75	485	13.47
seq_3_2_2	36	475	13.19	457	12.69	429	11.92	492	13.67	568	15.78
seq_3_2_3	36	372	10.33	560	15.56	613	17.03	412	11.44	505	14.03
seq_3_3_3	36	513	14.25	449	12.47	452	12.56	567	15.75	482	13.39
seq_4_0_0	36	481	13.36	573	15.92	573	15.92	477	13.25	448	12.44
seq_4_0_1	36	495	13.75	549	15.25	486	13.5	440	12.22	460	12.78
seq_4_1_1	36	415	11.53	508	14.11	535	14.86	527	14.64	362	10.06
seq_5_0_0	36	373	10.36	473	13.14	566	15.72	462	12.83	571	15.86
seq_5_0_1	36	426	11.83	433	12.03	477	13.25	520	14.44	510	14.17
seq_5_1_1	36	434	12.06	476	13.22	513	14.25	509	14.14	541	15.03
seq_6_0_0	36	484	13.44	418	11.61	401	11.14	402	11.17	527	14.64
seq_6_0_1	36	469	13.03	604	16.78	507	14.08	471	13.08	611	16.97
seq_6_0_2	36	492	13.67	528	14.67	486	13.5	477	13.25	490	13.61
seq_6_0_3	36	474	13.17	482	13.39	547	15.19	446	12.39	437	12.14
seq_6_1_1	36	499	13.86	547	15.19	490	13.61	499	13.86	595	16.53
seq_6_1_2	36	464	12.89	581	16.14	518	14.39	557	15.47	492	13.67
seq_6_1_3	36	472	13.11	493	13.69	495	13.75	599	16.64	528	14.67
seq_6_2_2	36	563	15.64	484	13.44	498	13.83	471	13.08	550	15.28
seq_6_2_3	36	507	14.08	481	13.36	474	13.17	460	12.78	473	13.14
seq_6_3_3	36	532	14.78	526	14.61	521	14.47	487	13.53	508	14.11
seq_7_0_0	36	512	14.22	550	15.28	505	14.03	632	17.56	491	13.64
seq_7_0_1	36	547	15.19	539	14.97	467	12.97	453	12.58	403	11.19
seq_7_0_2	36	478	13.28	467	12.97	504	14.0	538	14.94	490	13.61
seq_7_0_3	36	518	14.39	474	13.17	513	14.25	540	15.0	485	13.47
seq_7_1_1	36	509	14.14	479	13.31	475	13.19	480	13.33	498	13.83
seq_7_1_2	36	463	12.86	383	10.64	504	14.0	516	14.33	517	14.36
seq_7_1_3	36	452	12.56	475	13.19	567	15.75	543	15.08	478	13.28
seq_7_2_2	36	532	14.78	492	13.67	467	12.97	593	16.47	448	12.44
seq_7_2_3	36	581	16.14	468	13.0	574	15.94	456	12.67	532	14.78
seq_7_3_3	36	452	12.56	537	14.92	511	14.19	456	12.67	549	15.25

Table 3: Standard QUBO embedding result (2 of 2).

Graph name	Logical Qubits	Physical 1	Average 1	Physical 2	Average 2	Physical 3	Average 3	Physical 4	Average 4	Physical 5	Average 5
seq_8_0_0	36	519	14.42	546	15.17	547	15.19	602	16.72	462	12.83
seq_8_0_1	36	556	15.44	428	11.89	508	14.11	447	12.42	502	13.94
seq_8_0_2	36	641	17.81	516	14.33	459	12.75	500	13.89	531	14.75
seq_8_0_3	36	456	12.67	514	14.28	469	13.03	514	14.28	446	12.39
seq_8_1_1	36	582	16.17	596	16.56	511	14.19	537	14.92	512	14.22
seq_8_1_2	36	509	14.14	487	13.53	453	12.58	546	15.17	530	14.72
seq_8_1_3	36	577	16.03	516	14.33	521	14.47	539	14.97	504	14.0
seq_8_2_2	36	522	14.5	560	15.56	534	14.83	513	14.25	469	13.03
seq_8_2_3	36	399	11.08	528	14.67	531	14.75	524	14.56	483	13.42
seq_8_3_3	36	460	12.78	553	15.36	429	11.92	500	13.89	532	14.78
seq_9_0_0	36	492	13.67	507	14.08	428	11.89	483	13.42	468	13.0
seq_9_0_1	36	511	14.19	471	13.08	522	14.5	499	13.86	495	13.75
seq_9_0_2	36	528	14.67	524	14.56	495	13.75	426	11.83	545	15.14
seq_9_0_3	36	513	14.25	444	12.33	466	12.94	464	12.89	439	12.19
seq_9_1_1	36	416	11.56	446	12.39	445	12.36	392	10.89	564	15.67
seq_9_1_2	36	486	13.5	492	13.67	498	13.83	549	15.25	483	13.42
seq_9_1_3	36	570	15.83	516	14.33	418	11.61	440	12.22	502	13.94
seq_9_2_2	36	477	13.25	508	14.11	496	13.78	467	12.97	471	13.08
seq_9_2_3	36	550	15.28	563	15.64	508	14.11	508	14.11	446	12.39
seq_9_3_3	36	454	12.61	457	12.69	493	13.69	505	14.03	472	13.11
seq_10_0_0	36	416	11.56	629	17.47	513	14.25	473	13.14	456	12.67
seq_10_0_1	36	471	13.08	517	14.36	479	13.31	493	13.69	454	12.61
seq_10_0_2	36	424	11.78	468	13.0	513	14.25	517	14.36	453	12.58
seq_10_0_3	36	495	13.75	401	11.14	430	11.94	513	14.25	498	13.83
seq_10_1_1	36	464	12.89	338	9.39	431	11.97	455	12.64	516	14.33
seq_10_1_2	36	442	12.28	482	13.39	488	13.56	484	13.44	528	14.67
seq_10_1_3	36	572	15.89	445	12.36	466	12.94	446	12.39	543	15.08
seq_10_2_2	36	508	14.11	564	15.67	481	13.36	539	14.97	491	13.64
seq_10_2_3	36	467	12.97	525	14.58	465	12.92	502	13.94	525	14.58
seq_10_3_3	36	477	13.25	479	13.31	518	14.39	463	12.86	443	12.31
seq_11_0_0	36	475	13.19	536	14.89	486	13.5	493	13.69	464	12.89
seq_11_0_1	36	495	13.75	446	12.39	472	13.11	463	12.86	405	11.25
seq_11_1_1	36	417	11.58	418	11.61	458	12.72	506	14.06	455	12.64
seq_12_0_0	36	595	16.53	529	14.69	484	13.44	472	13.11	493	13.69
seq_12_0_1	36	478	13.28	469	13.03	510	14.17	467	12.97	552	15.33
seq_12_0_2	36	476	13.22	459	12.75	546	15.17	472	13.11	486	13.5
seq_12_0_3	36	465	12.92	520	14.44	421	11.69	452	12.56	511	14.19
seq_12_1_1	36	548	15.22	540	15.0	488	13.56	560	15.56	482	13.39
seq_12_1_2	36	585	16.25	459	12.75	539	14.97	381	10.58	487	13.53
seq_12_1_3	36	423	11.75	481	13.36	493	13.69	454	12.61	470	13.06
seq_12_2_2	36	487	13.53	527	14.64	507	14.08	474	13.17	465	12.92
seq_12_2_3	36	494	13.72	433	12.03	609	16.92	466	12.94	432	12.0
seq_12_3_3	36	435	12.08	527	14.64	579	16.08	411	11.42	374	10.39
seq_13_0_0	36	524	14.56	483	13.42	558	15.5	536	14.89	500	13.89
seq_13_0_1	36	465	12.92	488	13.56	392	10.89	459	12.75	497	13.81
seq_13_0_2	36	369	10.25	469	13.03	480	13.33	439	12.19	468	13.0
seq_13_1_1	36	484	13.44	444	12.33	584	16.22	478	13.28	578	16.06
seq_13_1_2	36	456	12.67	538	14.94	464	12.89	438	12.17	440	12.22
seq_13_2_2	36	477	13.25	494	13.72	524	14.56	498	13.83	541	15.03

Table 4: Improved (degree mapping) QUBO embedding result (1 of 2).

Graph name	Logical Qubits	Physical 1	Average 1	Physical 2	Average 2	Physical 3	Average 3	Physical 4	Average 4	Physical 5	Average 5
seq_0_0_0	14	29	2.07	33	2.36	28	2.0	30	2.14	35	2.5
seq_0_0_1	14	35	2.5	31	2.21	35	2.5	31	2.21	32	2.29
seq_0_1_1	14	43	3.07	35	2.5	32	2.29	57	4.07	33	2.36
seq_1_0_0	18	78	4.33	85	4.72	89	4.94	87	4.83	88	4.89
seq_1_0_1	18	83	4.61	91	5.06	73	4.06	85	4.72	99	5.5
seq_1_0_2	18	86	4.78	101	5.61	83	4.61	75	4.17	86	4.78
seq_1_1_1	18	87	4.83	81	4.5	86	4.78	80	4.44	89	4.94
seq_1_1_2	18	80	4.44	84	4.67	91	5.06	86	4.78	94	5.22
seq_1_2_2	18	87	4.83	95	5.28	98	5.44	77	4.28	91	5.06
seq_2_0_0	12	16	1.33	18	1.5	16	1.33	15	1.25	22	1.83
seq_2_0_1	12	36	3.0	40	3.33	20	1.67	41	3.42	50	4.17
seq_2_0_2	12	30	2.5	27	2.25	29	2.42	46	3.83	27	2.25
seq_2_0_3	12	26	2.17	27	2.25	34	2.83	24	2.0	22	1.83
seq_2_1_1	12	19	1.58	23	1.92	25	2.08	22	1.83	27	2.25
seq_2_1_2	12	32	2.67	20	1.67	23	1.92	29	2.42	31	2.58
seq_2_1_3	12	32	2.67	29	2.42	29	2.42	30	2.5	22	1.83
seq_2_2_2	12	25	2.08	24	2.0	32	2.67	24	2.0	22	1.83
seq_2_2_3	12	28	2.33	32	2.67	25	2.08	35	2.92	26	2.17
seq_2_3_3	12	24	2.0	24	2.0	20	1.67	23	1.92	21	1.75
seq_3_0_0	12	24	2.0	26	2.17	25	2.08	25	2.08	25	2.08
seq_3_0_1	12	38	3.17	27	2.25	35	2.92	30	2.5	27	2.25
seq_3_0_2	12	29	2.42	27	2.25	35	2.92	27	2.25	27	2.25
seq_3_0_3	12	27	2.25	31	2.58	29	2.42	31	2.58	28	2.33
seq_3_1_1	12	29	2.42	25	2.08	27	2.25	30	2.5	25	2.08
seq_3_1_2	12	29	2.42	28	2.33	30	2.5	25	2.08	32	2.67
seq_3_1_3	12	27	2.25	32	2.67	30	2.5	28	2.33	31	2.58
seq_3_2_2	12	26	2.17	28	2.33	29	2.42	30	2.5	29	2.42
seq_3_2_3	12	28	2.33	27	2.25	27	2.25	37	3.08	36	3.0
seq_3_3_3	12	30	2.5	28	2.33	29	2.42	33	2.75	30	2.5
seq_4_0_0	10	14	1.4	12	1.2	12	1.2	10	1.0	14	1.4
seq_4_0_1	10	13	1.3	13	1.3	14	1.4	14	1.4	13	1.3
seq_4_1_1	10	17	1.7	15	1.5	15	1.5	15	1.5	16	1.6
seq_5_0_0	10	12	1.2	16	1.6	18	1.8	17	1.7	13	1.3
seq_5_0_1	10	13	1.3	13	1.3	12	1.2	13	1.3	13	1.3
seq_5_1_1	10	12	1.2	14	1.4	10	1.0	12	1.2	12	1.2
seq_6_0_0	20	94	4.7	91	4.55	93	4.65	88	4.4	98	4.9
seq_6_0_1	20	112	5.6	103	5.15	107	5.35	100	5.0	97	4.85
seq_6_0_2	20	116	5.8	107	5.35	115	5.75	89	4.45	89	4.45
seq_6_0_3	20	101	5.05	112	5.6	109	5.45	110	5.5	108	5.4
seq_6_1_1	20	99	4.95	100	5.0	99	4.95	90	4.5	108	5.4
seq_6_1_2	20	112	5.6	125	6.25	109	5.45	125	6.25	112	5.6
seq_6_1_3	20	105	5.25	116	5.8	104	5.2	115	5.75	93	4.65
seq_6_2_2	20	106	5.3	128	6.4	108	5.4	105	5.25	118	5.9
seq_6_2_3	20	108	5.4	109	5.45	116	5.8	125	6.25	94	4.7
seq_6_3_3	20	106	5.3	113	5.65	115	5.75	108	5.4	109	5.45
seq_7_0_0	14	39	2.79	41	2.93	44	3.14	45	3.21	32	2.29
seq_7_0_1	14	47	3.36	44	3.14	48	3.43	48	3.43	39	2.79
seq_7_0_2	14	48	3.43	44	3.14	47	3.36	47	3.36	54	3.86
seq_7_0_3	14	43	3.07	51	3.64	39	2.79	35	2.5	36	2.57
seq_7_1_1	14	32	2.29	44	3.14	34	2.43	37	2.64	43	3.07
seq_7_1_2	14	44	3.14	39	2.79	45	3.21	45	3.21	42	3.0
seq_7_1_3	14	36	2.57	35	2.5	41	2.93	40	2.86	44	3.14
seq_7_2_2	14	52	3.71	34	2.43	35	2.5	34	2.43	43	3.07
seq_7_2_3	14	40	2.86	33	2.36	45	3.21	39	2.79	39	2.79
seq_7_3_3	14	45	3.21	36	2.57	67	4.79	42	3.0	36	2.57

Table 4: Improved (degree mapping) QUBO embedding result (2 of 2).

Graph name	Logical Qubits	Physical 1	Average 1	Physical 2	Average 2	Physical 3	Average 3	Physical 4	Average 4	Physical 5	Average 5
seq_8_0_0	14	41	2.93	30	2.14	43	3.07	45	3.21	42	3.0
seq_8_0_1	14	39	2.79	58	4.14	42	3.0	49	3.5	43	3.07
seq_8_0_2	14	35	2.5	59	4.21	37	2.64	40	2.86	40	2.86
seq_8_0_3	14	43	3.07	60	4.29	37	2.64	62	4.43	64	4.57
seq_8_1_1	14	40	2.86	33	2.36	38	2.71	38	2.71	40	2.86
seq_8_1_2	14	38	2.71	39	2.79	37	2.64	40	2.86	42	3.0
seq_8_1_3	14	44	3.14	47	3.36	43	3.07	45	3.21	48	3.43
seq_8_2_2	14	33	2.36	37	2.64	44	3.14	42	3.0	33	2.36
seq_8_2_3	14	40	2.86	55	3.93	51	3.64	49	3.5	50	3.57
seq_8_3_3	14	48	3.43	37	2.64	39	2.79	42	3.0	46	3.29
seq_9_0_0	12	26	2.17	30	2.5	25	2.08	29	2.42	30	2.5
seq_9_0_1	12	25	2.08	26	2.17	25	2.08	26	2.17	34	2.83
seq_9_0_2	12	28	2.33	32	2.67	27	2.25	31	2.58	28	2.33
seq_9_0_3	12	25	2.08	34	2.83	27	2.25	32	2.67	29	2.42
seq_9_1_1	12	28	2.33	26	2.17	25	2.08	23	1.92	26	2.17
seq_9_1_2	12	25	2.08	27	2.25	25	2.08	28	2.33	27	2.25
seq_9_1_3	12	27	2.25	29	2.42	28	2.33	28	2.33	29	2.42
seq_9_2_2	12	30	2.5	30	2.5	38	3.17	28	2.33	25	2.08
seq_9_2_3	12	29	2.42	29	2.42	28	2.33	28	2.33	26	2.17
seq_9_3_3	12	30	2.5	29	2.42	26	2.17	29	2.42	28	2.33
seq_10_0_0	12	22	1.83	25	2.08	20	1.67	21	1.75	22	1.83
seq_10_0_1	12	33	2.75	25	2.08	30	2.5	26	2.17	30	2.5
seq_10_0_2	12	27	2.25	38	3.17	38	3.17	22	1.83	25	2.08
seq_10_0_3	12	22	1.83	22	1.83	20	1.67	22	1.83	23	1.92
seq_10_1_1	12	26	2.17	30	2.5	25	2.08	28	2.33	28	2.33
seq_10_1_2	12	37	3.08	34	2.83	30	2.5	43	3.58	29	2.42
seq_10_1_3	12	30	2.5	31	2.58	25	2.08	21	1.75	19	1.58
seq_10_2_2	12	26	2.17	28	2.33	19	1.58	41	3.42	24	2.0
seq_10_2_3	12	22	1.83	34	2.83	20	1.67	24	2.0	27	2.25
seq_10_3_3	12	23	1.92	31	2.58	33	2.75	24	2.0	26	2.17
seq_11_0_0	14	30	2.14	34	2.43	30	2.14	30	2.14	29	2.07
seq_11_0_1	14	38	2.71	40	2.86	43	3.07	39	2.79	42	3.0
seq_11_1_1	14	32	2.29	36	2.57	41	2.93	34	2.43	34	2.43
seq_12_0_0	20	88	4.4	95	4.75	96	4.8	109	5.45	93	4.65
seq_12_0_1	20	104	5.2	105	5.25	89	4.45	92	4.6	92	4.6
seq_12_0_2	20	96	4.8	86	4.3	107	5.35	98	4.9	99	4.95
seq_12_0_3	20	113	5.65	108	5.4	97	4.85	95	4.75	95	4.75
seq_12_1_1	20	115	5.75	103	5.15	105	5.25	110	5.5	111	5.55
seq_12_1_2	20	107	5.35	111	5.55	128	6.4	114	5.7	105	5.25
seq_12_1_3	20	111	5.55	105	5.25	107	5.35	102	5.1	116	5.8
seq_12_2_2	20	101	5.05	114	5.7	101	5.05	116	5.8	134	6.7
seq_12_2_3	20	102	5.1	96	4.8	90	4.5	100	5.0	87	4.35
seq_12_3_3	20	104	5.2	99	4.95	102	5.1	99	4.95	101	5.05
seq_13_0_0	18	81	4.5	87	4.83	80	4.44	92	5.11	82	4.56
seq_13_0_1	18	82	4.56	93	5.17	95	5.28	79	4.39	98	5.44
seq_13_0_2	18	93	5.17	109	6.06	90	5.0	93	5.17	93	5.17
seq_13_1_1	18	81	4.5	80	4.44	91	5.06	72	4.0	81	4.5
seq_13_1_2	18	78	4.33	90	5.0	93	5.17	91	5.06	74	4.11
seq_13_2_2	18	77	4.28	71	3.94	91	5.06	80	4.44	91	5.06

Table 5: Probability of obtaining optimal solution with 5000 samples

Graph name	Solution probability Improved QUBO
seq_0_0_0	0.9986
seq_0_1_1	0.999
seq_1_0_0	0.3016
seq_1_1_1	0.0102
seq_1_2_2	0.9724
seq_2_0_0	1.0
seq_2_1_1	1.0
seq_2_2_2	1.0
seq_2_3_3	1.0
seq_3_0_0	1.0
seq_3_1_1	0.9988
seq_3_2_2	1.0
seq_3_3_3	0.9964
seq_4_0_0	1.0
seq_4_1_1	1.0
seq_5_0_0	1.0
seq_5_1_1	1.0
seq_6_0_0	0.9892
seq_6_1_1	0.9512
seq_6_2_2	0.3754
seq_6_3_3	0.684
seq_7_0_0	0.9968
seq_7_1_1	0.8734

Graph name	Solution probability Improved QUBO
seq_7_2_2	0.9962
seq_7_3_3	1.0
seq_8_0_0	1.0
seq_8_1_1	0.9982
seq_8_2_2	0.9998
seq_8_3_3	0.9952
seq_9_0_0	0.9992
seq_9_1_1	1.0
seq_9_2_2	0.9792
seq_9_3_3	0.851
seq_10_0_0	1.0
seq_10_1_1	0.9968
seq_10_2_2	1.0
seq_10_3_3	1.0
seq_11_0_0	0.9976
seq_11_1_1	0.9998
seq_12_0_0	0.7818
seq_12_1_1	0.5208
seq_12_2_2	0.8864
seq_12_3_3	0.1164
seq_13_0_0	0.0
seq_13_1_1	0.0448
seq_13_2_2	0.2154

4.2.1 Parameter and option settings

Each different model of D-Wave quantum computers supports a different range of h and J values of Equation (2) for the Ising problem submitted. The particular model used for this experiment (D-Wave 2X) have a h value range of $[-2, 2]$ and a J value range of $[-1, 1]$. All instances submitted to the hardware will have all entries scaled to the full available value range that is supported by the hardware if the 'auto-scale' parameter is set to true (default). The D-Wave user manual [17] states that one should avoid having $J < -0.8$ for some types of problems. Although the exact reason for this advice is not given in [17], we suspect that this effect is mainly caused by Integrated Control Errors (ICEs) in the D-Wave quantum annealers which are systematic errors in the hardware (see [17] for details on the different types of ICEs D-Wave hardware may have). Since these ICEs are difficult to approximate and unavoidable in general, the 'auto-scale' option is turned off and all entries of the Ising instance are scaled by a factor s to ensure that all J entries are greater than -0.8. See Python Script D for the implementation.

Another setting the D-Wave software package [15] provides to mitigate the effects of ICEs is the spin reversal setting. The physical process of adiabatic quantum computing can be described by the Adiabatic Theorem [21], which states that the spins (± 1) of all qubits will converge to the minimum energy state (ground state) of the system with high probability if the system is let to evolve slow enough. The ground state of the system in this case is defined by the Ising objective function (2) which is dictated by h and J values. Due to systematic errors in the hardware, when physical qubits and couplers in the D-Wave are being programmed, the actual value set for each qubit (h) and coupler (J) may have some positive or negative bias. This essentially means that the Ising instance being solved by the hardware is not a hundred percent accurately described by the Ising model (2). This behavior can be modeled by the following Ising problem:

$$x^* = \min_{\mathbf{x}} \sum_{ij \in E} x_i (J_{(i,j)} + \delta J_{(i,j)}) x_j + \sum_{i \in V} (h_i + \delta h_i) x_i. \tag{11}$$

In Equation (11), δh and δJ are the offset bias values. Depending on how big these biases are, they will affect the probability of getting the optimal solution of the original Ising instance to a different degree. What makes this issue difficult to deal with is the fact that these bias values are not constant but rather depend on h and J values [17]. Once again, there is no definitive way of calculating these biases in theory and we do not have enough resource to empirically measure them for all the different test cases we have. The spin reversal setting provides an alternative. It takes a random subset of all qubits say $S = \{x_0, x_1, \ldots x_n\}$ used and set:

$$\begin{gathered} x_i \to x_i' = -x_i \\ h_i \to h_i' = -h_i \\ J_{(x,y)} \to J_{(x,y)}' = J_{(x,y)} \text{ if } x = i \text{ or } y = i \end{gathered}$$

For each $x_i \in S$. This operation does not change the energy level of any state of the system and only interprets $+1$ spins as -1 spins and vice versa. The basic idea is that by doing this, we would essentially be introducing a different set of bias values which may have a less effect on the entire system as a whole then the original errors. Note that this operation does not guarantee better solution quality (one may get a new set of biases that are even worse), and once again, we do not have the resource to determine the best spin reversal setting for each test case. And therefore the recommended value of 2 [17](default value is 0) is used in Script D. This means rather than submitting one Ising instance to the hardware, two different instances are being created each with a different set of biases. And then the two new instances are executed on the D-Wave quantum annealer separately. The manual [17] does not contain any details on how the random subset S is determined. The spin reversal transformation also avoids (at least to a higher probability) potential programming errors. The D-Wave hardware have a less than 10% probability of not having physical qubits programmed correctly [17]. By setting spin reversal number to 2, the chance of error occurring in both trials are reduced to less than 1%.

Before other parameter settings can be discussed, one needs to understand the basic procedure of how D-Wave quantum annealers generate samples. After the Ising instance is programmed into the hardware, there will be a short wait time known as the post-programming thermalization time. Its purpose is to let the chipset cool down as much as it can since the smallest amount of heat could affect the system. The default value of 1000 microseconds is used in this experiment, the user manual [15] states that smaller values will potentially lower the quality of samples. Then the sampling cycle begins. The system will evolve for some time, this is known as the annealing time. During annealing, the states of all physical qubits will converge towards the ground state of the system. The default time of 20 microseconds is used here. In theory, the longer the annealing time (the longer the system is let to evolve), the higher the probability it will end up in the ground state of the system [6]. However, in practice, the longer annealing takes place, the more susceptible the system becomes to noise due to heat leakage. As a result, a short annealing time is much suitable for current hardware and so we used the minimum annealing time of 20 microseconds (default) as suggested by previous experimental work such as [1, 18, 28, 31]. The state of each qubits will be read at the end of annealing and this counts as one sample. The system will also take some extra time after reading one sample to properly cool down again and this is known as the post-readout thermalization time. Once again, the user manual [15] mentions that lower values of this parameter will reduce solution quality so a value of 100 microseconds is used instead of the default value which is 0 microseconds. Note that, this default time is not referenced in the user manual [15], since this number seems to be different for each model [15, 17]. This default value can be found by using an API call to the actual D-Wave 2X hardware. The annealing cycle is only done once by default, this makes estimating the probability of obtaining the optimal solution impossible to calculate. Therefore in Script D, the process is set to repeat 5000 times, generating 5000 samples for each test case. Note that since the number of spin reversal is set to 2, $5000/2 = 2500$ samples are taken for each of the two transformed instances.

After obtaining the samples for an Ising instance, it is then mapped back into the corresponding solutions to the original QUBO instance. This function used to do so is the one implemented in the D-Wave API [15]. Recall that the embedding maps each logical qubit of the QUBO problem to a set of connected physical qubits in the hardware. Sometimes, the set of physical qubits for a single logical qubit does not have a consistent answer (i.e. a combination of $+1$ and -1 spins). In this case, a majority vote is taken among the physical qubits to determine the final value for the logical qubit. There is also a post-processing optimization option available, it will attempt to optimize the sample obtained using certain classical algorithm, the manual [15, 16] does not have much detail on exactly what type of algorithm is used and it does not seem to affect the overall probability of getting the optimal solution much based on previous experimental work [1, 18, 28].

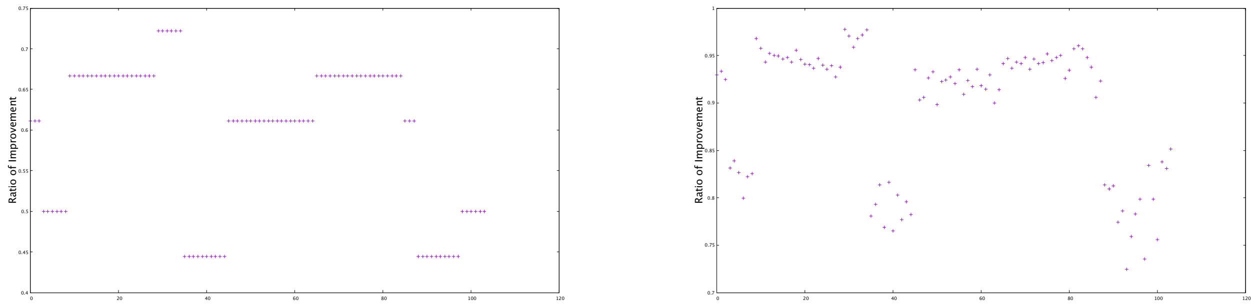
5 Discussion and Conclusions

As mentioned in Section 4.2.1, 5000 samples are generated for each instance of our tests. The number of correct optimal solution for each instance is then calculated and the probabilities are given in Table 5. Note that only the probabilities for the improved QUBOs are given. For all the standard QUBOs, the D-Wave 2X we used failed to find any optimal solutions with 5000 samples hence making the estimating the probabilities impossible. Note we did verify the correctness of the QUBOs via a classical algorithm using the CPLEX library [23] (see [26] for the implementation).

As can be seen in the Table 5, the improved QUBOs have consistently high probabilities of obtaining the optimal solution for most test cases. There are also a few exceptions, see the highlighted entries in Table 5. The embeddings of these highlighted test cases are at least on par with the rest, but the probability of obtaining the optimal solution is significantly lower the rest of the test cases. While we do not have a definitive answer that explains this result, we suspect it may have been caused by some unexpected sudden background noise at the time of the experiment. The test cases were executed on the hardware in batches over several days so it is very difficult to pinpoint exactly what was causing the problem (see [17] for a summary of potential sources).

Based on the embedding results in Table 3 and 4, we have calculated the ratio of improvement in terms of the number of qubits required. Let $\text{qubits}_{\text{imp}}$ and $\text{qubits}_{\text{std}}$ be the number of qubits required by Equation (6) and (3) respectively, we define the ratio of improvement as $1 - \dfrac{\text{qubits}_{\text{imp}}}{\text{qubits}_{\text{std}}}$. The ratio of improvement of both logical and physical qubits for each test cases in these tables are shown in Figure 1 in the same order as they are listed in Table 3 and 4. Note that we only used the best embedding (one with minimum average map size) when calculating the ratio of improvements for physical qubits. As can be seen, there are very clear clusterings in both figures. Test cases with the same degree sequence will have the exactly same number of logical qubits and since there are only fourteen different degree sequences, the clusterings are expected. Another thing to note is that the improvement ratio post-embedding is in general a lot higher. It has been shown in [9] that it requires $O(n^2)$ physical qubits to embed a complete graph of order n in the Chimera architecture. This means that any small improvement in the number of logical qubits should be magnified after embedding hence the increased ratio of improvement, as we see here.

It is interesting to see how the ratio of improvement scales with much larger graphs so we also computed the expected ratio of improvement for random graphs. There are several different models to generate random graphs, we consider the case where the edges in the graph are chosen independently each with probability p, that is, the probability of an edge existing between two vertices v_i and v_j is p for some $0 < p \leq 1$. See [7, 8] for more details



(a) Pre-embedding (Logical Qubits) (b) Post-embedding (Physical Qubits)

Figure 1: Ratio of improvement in QUBO variables.

about this model. The graphs we consider here will be too large to be embedded in current D-Wave computer. So only the expected improvement ratio for logical qubits are computed. Given a random graph $G = (V, E)$ with order n and a vertex $v \in V$, the probability of $\deg(v) = k$ can be calculated by the following formula from [7]

$$f(n,k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}.$$

Intuitively, vertex v has k neighbors, chosen from $n-1$ vertices, each with independent probability p and hence the formula follows. See [7] for a more detailed discussion. Note that to randomly choose k neighbors each with probability p is the same as to randomly choose $n-k-1$ non-neighbors each with probability $1-p$, so the value of $f(n,k)$ is identical for p_1 and p_2 if $p_1 = 1 - p_2$.

Since the degree of a vertex can range from 0 to $n-1$, if u and v are two vertices, then the probability of $\deg(u) = \deg(v)$ can be expressed as $\sum_{k=0}^{n-1} f(n,k)^2$. If $\deg(u) = \deg(v) \neq 0$ then we would need one binary variable $x_{u,v}$ and so the expected number of variables we need is $n^2 \sum_{k=1}^{n-1} f(n,k)^2$. Formula (3) always require n^2 binary variables, so the expected ratio of improvement is exactly $1 - \sum_{k=1}^{n-1} f(n,k)^2$. We calculated the expected ratio of improvement for n up to 1000 along with different values of p, see Figure 2. The degree of a vertex can range from 0 to $n-1$ and the distribution of the degrees should be the most uniform at $p = 0.5$. A more uniform distribution of degrees suits Formula (6) better since there will be less vertices pairs of the same degree and hence the highest ratio of improvement we see in Figure 2. Overall, the result is very positive for large random graphs. The ratio of improvement increases as n goes up and exceeds 95% very quickly at around $n = 130, 160$ and 360 for $p = 0.5, 0.3$ and 0.1 respectively. And it is reasonable to assume it will be even better post-embedding (recall that the number of qubits scales quadratically post-embedding).

In summary, compared to the standard QUBO formulation in [13], the improved version given in Section 3 requires only a small amount of classical computation overhead. See Script C and the appendix of [12] for a comparison of the implementation of the two QUBO formulations. The payoff of this classical preprocessing overhead is paramount. Not only does the improved version works a lot better with current hardware, the scaling behavior shown in Figure 2 also demonstrates that this approach is very suitable for hardware of much larger scale. Furthermore, there are other similar approach to boost the performance

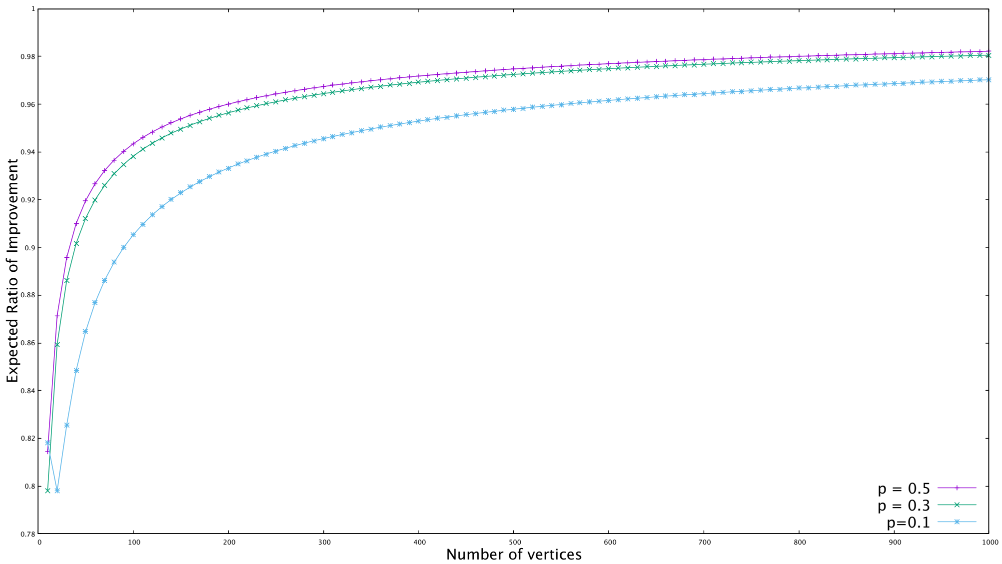


Figure 2: Expected ratio of improvement for graphs of order n up to 1000.

even more. For example, by Theorem 1, to check if G_1 and G_2 are isomorphic, we can check whether the complement of G_1 and G_2 are isomorphic instead. So it is relatively straightforward to see that we can generate two different QUBO instances for each test case, and use whichever gives a better embedding in practice. We plan to study some of these properties further in the future.

Acknowledgement

This project was supported in part by the Quantum Computing Research Initiatives at Lockheed Martin.

References

[1] Alastair A Abbott, Cristian S Calude, Michael J Dinneen, and Richard Hua. A hybrid quantum-classical paradigm to mitigate embedding costs in quantum annealing. *arXiv preprint arXiv:1803.04340*, 2018.

[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 review*, 50(4):755–787, 2008.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[4] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM Symposium on Theory of Computing*, pages 684–697. ACM, 2016.

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

[6] Sergio Boixo, Troels F Rønnow, Sergei V Isakov, Zhihui Wang, David Wecker, Daniel A Lidar, John M Martinis, and Matthias Troyer. Evidence for quantum annealing with more than one hundred qubits. *Nature Physics*, 10(3):218, 2014.

[7] Béla Bollobás. Degree sequences of random graphs. *Discrete Mathematics*, 33(1):1–19, 1981.

[8] Béla Bollobás. Vertices of given degree in a random graph. *Journal of Graph Theory*, 6(2):147–155, 1982.

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

[10] Stephen G Brush. History of the Lenz-Ising model. *Reviews of modern physics*, 39(4):883, 1967.

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

[12] Cristian S Calude, Michael J Dinneen, and Richard Hua. QUBO formulations for the graph isomorphism problem and related problems. Report CDMTCS-499, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, October 2016.

[13] Cristian S Calude, Michael J Dinneen, and Richard Hua. QUBO formulations for the graph isomorphism problem and related problems. *Theoretical Computer Science*, 701:54–69, 2017.

[14] D-Wave Systems. D-Wave 2000Q^{TM}. https://www.dwavesys.com/d-wave-two-system, 2017.

[15] D-Wave Systems, Inc. Developer guide for Python. *Technical Report Release 2.4 09-1024A-B*, 2017.

[16] D-Wave Systems, Inc. D-Wave problem-solving handbook. *D-Wave user manual 09-1171A-A*, 2018.

[17] D-Wave Systems, Inc. Technical description of the D-Wave quantum processing unit. *D-Wave User Manual 09-1109A-N*, 2019.

[18] Michael J Dinneen and Richard Hua. Formulating graph covering problems for adiabatic quantum computers. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '17, pages 18:1–18:10, New York, NY, USA, 2017. ACM.

[19] Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, 346:180–197, 2016.

[20] 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.

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

[22] Aric Hagberg, Dan Schult, Pieter Swart, D Conway, L Séguin-Charbonneau, C Ellison, B Edwards, and J Torrents. Networkx. high productivity software for complex networks. *Webová strá nka https://networkx. lanl. gov/wiki*, 2013.

[23] IBM. ILOG CPLEX optimization studio CPLEX user's manual, 2017. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer.

[24] IBM. IBM Q. https://www.research.ibm.com/ibm-q/technology/devices/, 2019.

[25] Steven Lindell. A logspace algorithm for tree canonization. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 400–404. ACM, 1992.

[26] Kai Liu. Solving optimization problems using adiabatic quantum computing. Master's thesis, University of Auckland, 2018.

[27] Anna Lubiw. Some NP-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981.

[28] Anuradha Mahasinghe, Richard Hua, Michael J Dinneen, and Rajni Goyal. Solving the Hamiltonian cycle problem using a quantum computer. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW 2019, pages 8:1–8:9, New York, NY, USA, 2019. ACM.

[29] Catherine Mcgeoch. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing*, 5:1–93, 07 2014.

[30] Kristen L Pudenz, Tameem Albash, and Daniel A Lidar. Error-corrected quantum annealing with hundreds of qubits. *Nature communications*, 5:3243, 2014.

[31] Troels F Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V Isakov, David Wecker, John M Martinis, Daniel A Lidar, and Matthias Troyer. Defining and detecting quantum speedup. *Science*, 345(6195):420–424, 2014.

[32] William Stein et al. Sage mathematics software (version 7.0), 2016.

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

[34] Elisabeth Wong, Brittany Baur, Saad Quader, and Chun-Hsi Huang. Biological network motif detection: principles and practice. *Briefings in bioinformatics*, 13(2):202–215, 2011.

[35] Bin Zhou and Jian Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, volume 8, pages 506–515. Citeseer, 2008.

A Sage Script to Generate All Graphs of Order 6

```
import sys
n = 6
seq_count = 0

# iterate all degree sequences of length 6
for seq in DegreeSequences(n):

    list_of_graphs = graphs(n, degree_sequence = seq)
    num_chosen_graphs = 0
    # count the number of graphs selected as test cases
    for graph in list_of_graphs:
        if graph.is_connected() and graph.complement().is_connected():
            if (not (graph.is_tree())) and (not graph.complement().is_tree()):
                num_chosen_graphs += 1
    if num_chosen_graphs > 1:
        list_of_graphs = graphs(n, degree_sequence = seq)
        print str(seq),"#",seq_count
        print str(num_chosen_graphs)

        for graph in list_of_graphs:

            if (not (graph.is_tree())) and (not graph.complement().is_tree()):
                if graph.is_connected() and graph.complement().is_connected():
                    print str(graph.order())
                    for i in graph.vertices():
                        for j in graph.neighbors(i): print j,
                        print

        seq_count += 1
```

listings/ds_sage.sage

B Python Script to Generate the QUBOs

```
# usage: this program takes the output generated by Script A for one degree
   sequence from the standard input stream

import sys, random
import networkx as nx
import graph_util as util
```

```
# get the degree sequence and the number of graphs that corresponds to it
seq = eval(sys.stdin.readline())
print seq
num_graphs = int(sys.stdin.readline().strip())
print num_graphs

# get all graphs correspond to the degree sequence
graph_list = []
for i in range(num_graphs):
    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))
    graph_list.append(G)

# generate all qubos per graph pair
for i in range(num_graphs):
    G_1 = graph_list[i]
    for j in range(i, num_graphs):
        G_2 = graph_list[j]

        # generate random vertex permutation
        perm = list(range(G_2.order()))
        random.shuffle(perm)

        # permute G_2
        G_2 = util.vertex_permutation(G_2, perm)

        # qubo generated using the standard formula
        (Q, n, vars_dict) = util.generate_standard_qubo(G_1, G_2)
        util.print_qubo(Q,n)
        print 'perm = ', perm
        print 'vars = ', vars_dict

        # qubo generated using the improved formula
        (Q, n, vars_dict) = util.generate_deg_map_qubo(G_1, G_2)
        util.print_qubo(Q,n)
        print 'perm = ', perm
        print 'vars = ', vars_dict
```

listings/generate_all_qubo.py

C Python Utility Script

```
import networkx as nx
import sys

def vertex_permutation(G,p):
    n = G.order()
```

```python
    GP = nx.empty_graph(n,create_using=nx.Graph())
    for (u,v) in G.edges():
        GP.add_edge(p[u],p[v])
    return GP

def print_qubo(Q,n):
    print n
    for i in range(n):
        for j in range(n):
            print Q[i,j],
        print

def generate_standard_qubo(G1, G2):
    n1 = G1.order()
    n2 = G2.order()
    if not(n1 == n2):
        print 'Order of graphs not the same'
        return

    varsDict,edgeDict = {}, {}
    for i in range(n2):
        for j in range(n2):
            if ((i,j) in G2.edges()) or ((j,i) in G2.edges()):
                edgeDict[i,j],edgeDict[j,i] = 1,1
            else:
                edgeDict[i,j],edgeDict[j,i] = 0,0
    index = 0
    for i in range(n1):
        for j in range(n2):
            varsDict[(i,j)] = index
            index += 1

    # initialize Q
    Q = {}
    for i in range(n1*n2):
        for j in range(n1*n2):
            Q[i,j] = 0

    a = 2
    b = 3
    # HA part 1
    # -2 sum xi,i'
    for i in range(n1):
        for iprime in range(n2):
            index = varsDict[(i,iprime)]
            Q[index,index] -= 2*a
        for iprime1 in range(n2):
            for iprime2 in range(n2):
                index1 = varsDict[(i,iprime1)]
                index2 = varsDict[(i,iprime2)]
                Q[index1, index2] += 1*a
    # HA part 2
    for iprime in range(n2):
        for i in range(n1):
```

```
            index = varsDict[(i,iprime)]
            Q[index,index] -= 2
        for i1 in range(n1):
            for i2 in range(n1):
                index1 = varsDict[(i1,iprime)]
                index2 = varsDict[(i2,iprime)]
                Q[index1, index2] += 1

    # Pij
    for (i,j) in G1.edges():
        for iprime in range(n2):
            xiiprime = varsDict[(i,iprime)]
            for jprime in range(n2):
                xjjprime = varsDict[(j,jprime)]
                Q[xiiprime,xjjprime] += b*(1-edgeDict[iprime,jprime])

    # Making Q uppertriangular
    for i in range(n1*n2):
        for j in range(n1*n2):
            if (i > j) and (not(Q[i,j]==0)):
                Q[j,i] += Q[i,j]
                Q[i,j] = 0

    return Q,n1*n2,varsDict

def generate_deg_map_qubo(G1, G2):
    n1 = G1.order()
    n2 = G2.order()
    if not(n1 == n2):
        print 'Order of graphs not the same'
        return

    varsDict,edgeDict = {},{}
    for i in range(n2):
    for j in range(n2):
        if ((i,j) in G2.edges()) or ((j,i) in G2.edges()):
                edgeDict[i,j],edgeDict[j,i] = 1,1
            else:
                edgeDict[i,j],edgeDict[j,i] = 0,0
    index = 0
    total_num_var = 0
    for i in range(n1):
        for j in range(n2):
            if G1.degree(i) == G2.degree(j):
                varsDict[(i,j)] = index
                index += 1
                total_num_var += 1

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

```
a = 2
b = 3
# HA part 1
# -2 sum xi,i'
for i in range(n1):
    for iprime in range(n2):
        if (i, iprime) in varsDict:
            index = varsDict[(i,iprime)]
            Q[index,index] -= 2*a

for iprime1 in range(n2):
    for iprime2 in range(n2):
            if (i, iprime1) in varsDict and (i,iprime2) in varsDict:
                index1 = varsDict[(i,iprime1)]
                index2 = varsDict[(i,iprime2)]
                Q[index1, index2] += 1*a
# HA part 2
for iprime in range(n2):
    for i in range(n1):
        if (i,iprime) in varsDict:
            index = varsDict[(i,iprime)]
            Q[index,index] -= 2
    for i1 in range(n1):
        for i2 in range(n1):
            if (i1, iprime) in varsDict and (i2, iprime) in varsDict:
                index1 = varsDict[(i1,iprime)]
                index2 = varsDict[(i2,iprime)]
                Q[index1, index2] += 1

# Pij
for (i,j) in G1.edges():
    for iprime in range(n2):
        if (i, iprime) in varsDict:
            xiiprime = varsDict[(i,iprime)]
            for jprime in range(n2):
                if (j,jprime) in varsDict:
                    xjjprime = varsDict[(j,jprime)]
                    Q[xiiprime,xjjprime] += b*(1-edgeDict[iprime,jprime])

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

return Q,total_num_var,varsDict
```

listings/graph_util.py

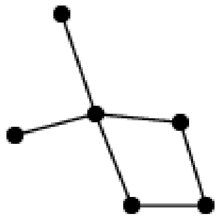
D Python Script to Solve QUBO Using D-Wave 2X

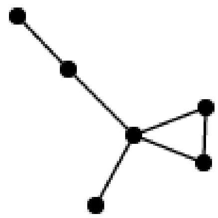
```
import sys, time, math, traceback
from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.util import get_hardware_adjacency
from dwave_sapi2.embedding import embed_problem, unembed_answer
from dwave_sapi2.util import qubo_to_ising, ising_to_qubo
from dwave_sapi2.core import solve_ising
from sys import exc_info

# read input

line=sys.stdin.readline().strip().split()
print('header:',line)
n=int(line[0])

#Q = defaultdict(int)
Q = {}
for i in range(n):
    line=sys.stdin.readline().strip().split()
    for j in range(n):
        t = float(line[j])
        if t==0: continue
        if i <= j: Q[(i,j)]=Q.setdefault((i,j),0)+t
        else:      Q[(j,i)]=Q.setdefault((j,i),0)+t

print('Q=',Q)

(H,J,ising_offset) = qubo_to_ising(Q)

print sys.stdin.readline().strip()
print sys.stdin.readline().strip()

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

# create a remote connection using url and token and connect to solver
#

print('Attempting to connect to network...')
try:
    remote_connection = RemoteConnection(url, token)
    solver = remote_connection.get_solver(solver_name)
except:
    print('Error: %s %s %s' % sys.exc_info()[0:3])
    traceback.print_exc()

#print('Solver properties:\n%s\n' % solver.properties)
A = get_hardware_adjacency(solver)

# Embed problem into hardware
```

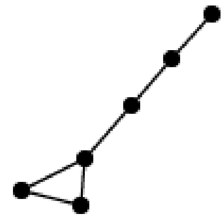
```
(h0, j0, jc, new_emb) = embed_problem(H, J, embedding, A)
#print 'new_emb=',new_emb
assert new_emb==embedding

# compute scale s so in range [-.8,1]
print "min h0,j0",min(h0)/2.1,min(j0.values())
print "max h0,j0",max(h0)/2.1,max(j0.values())
maxH=0.0
if len(h0): maxH=max(abs(min(h0)),abs(max(h0)))
maxJ=max(abs(min(j0.values())),abs(max(j0.values())))
maxV=max(maxH/2.0,maxJ)
s = 0.8/maxV

h1= [val*s for val in h0]
j1 = {}
for (key, val) in j0.iteritems():
    j1[key]=val*s

print 'd-wave Ising'
print 'h1=',h1
print 'j1=',j1

print "min h1,j1",min(h1),min(j1.values())
print "max h1,j1",max(h1),max(j1.values())
assert max(h1) <= 2.01
assert min(h1) >= -2.01
assert max(j1.values()) <= 1.01
assert min(j1.values()) >= -0.81

j1.update(jc)

for spins in [2]: # [[2,4,8,16]:
  #break
  annealT,progT,readT=20,1000,100
  print 'annealT=',annealT,'progT=',progT,'readT=',readT,'spins=',spins
  result = solve_ising(solver, h1, j1, num_reads=5000, annealing_time=annealT
    ,\
      programming_thermalization=progT, readout_thermalization=readT,
    postprocess='optimization',\
      num_spin_reversal_transforms=spins,auto_scale=False)
  print 'result:', result


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

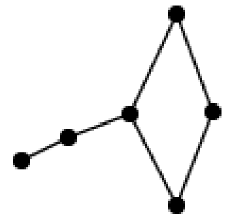
listings/preembedIsoDegScale.py

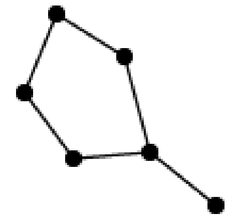
E Drawings of Test Graphs

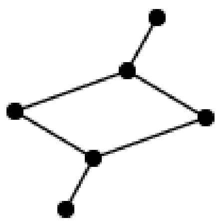

seq_0_0

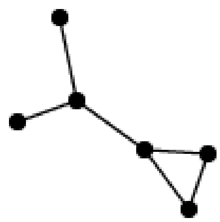

seq_0_1

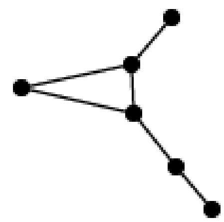

seq_1_0

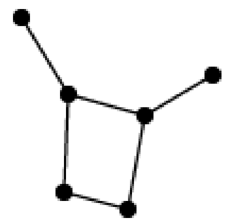

seq_1_1

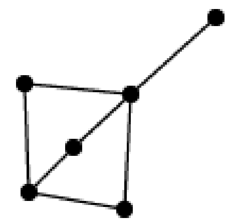

seq_1_2

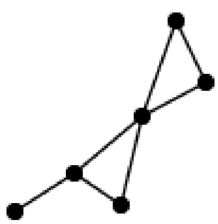

seq_2_0

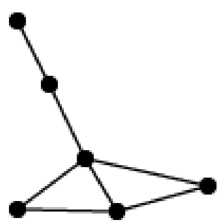

seq_2_1

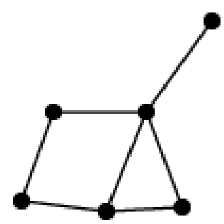

seq_2_2

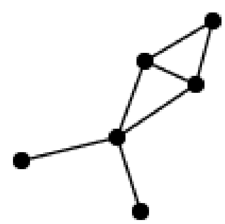

seq_2_3

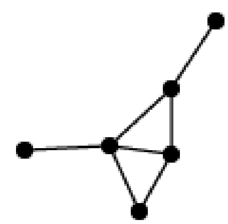

seq_3_0

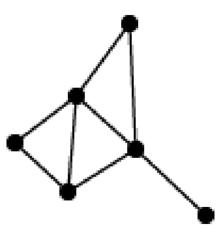

seq_3_1

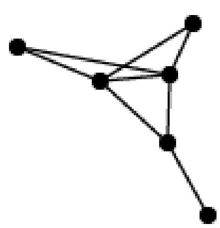

seq_3_2

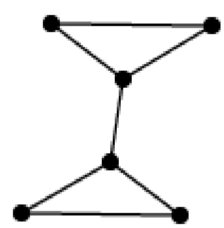

seq_3_3

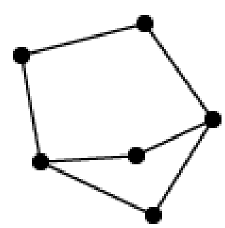

seq_4_0

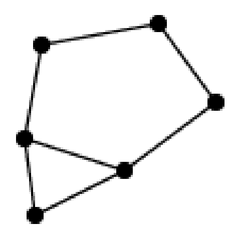

seq_4_1

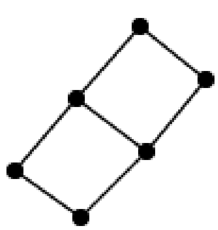

seq_5_0

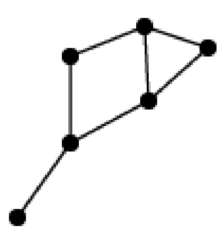

seq_5_1

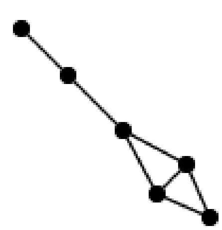

seq_6_0

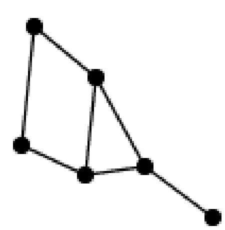

seq_6_1

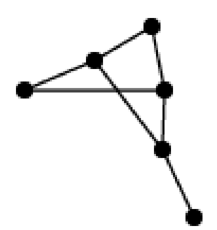

seq_6_2

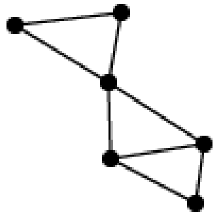

seq_6_3

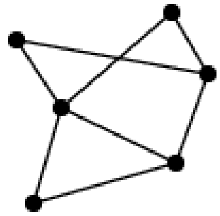

seq_7_0

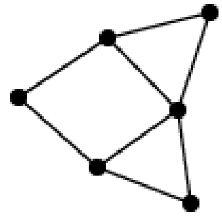

seq_7_1

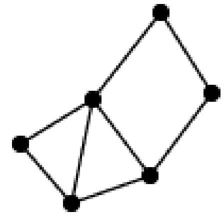

seq_7_2

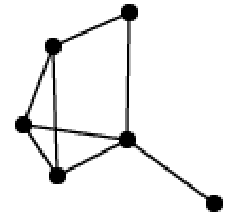

seq_7_3

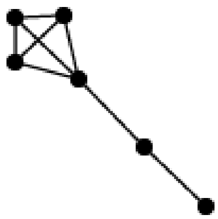

seq_8_0

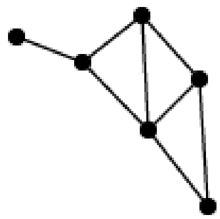

seq_8_1

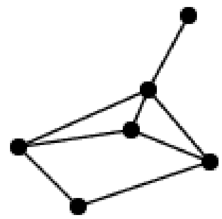

seq_8_2

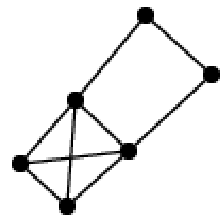

seq_8_3

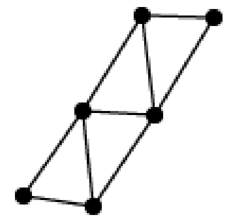

seq_9_0

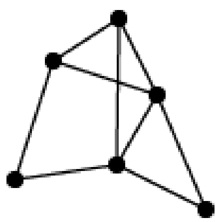

seq_9_1

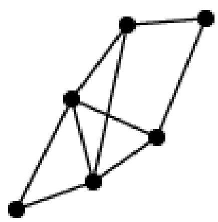

seq_9_2

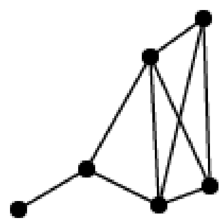

seq_9_3

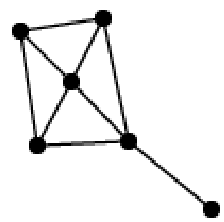

seq_10_0

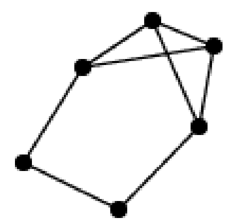

seq_10_1

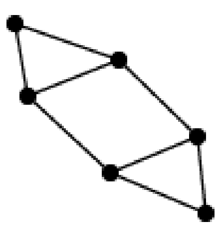

seq_10_2

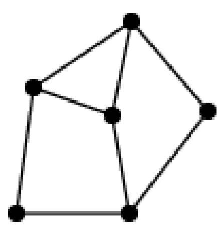

seq_10_3

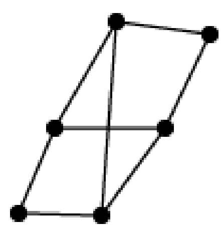

seq_11_0

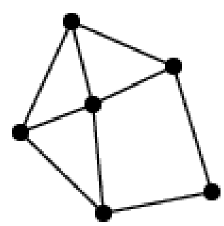

seq_11_1

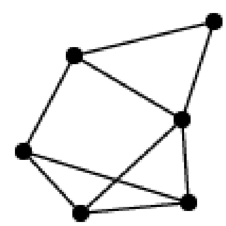

seq_12_0

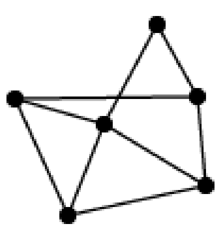

seq_12_1


seq_12_2


seq_12_3


seq_13_0


seq_13_1


seq_13_2