CDMTCS
Research
Report
Series

QUBO Formulations for the Graph Isomorphism Problem and Related Problems

Cristian S. Calude
Michael J. Dinneen
Richard Hua
Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-499
April 2016 (revised)

Centre for Discrete Mathematics and
Theoretical Computer Science

QUBO Formulations for the Graph Isomorphism Problem and Related Problems

Cristian S. Calude, Michael J. Dinneen and Richard Hua
Department of Computer Science, University of Auckland,
Auckland, New Zealand
cristian@cs.auckland.ac.nz mjd@cs.auckland.ac.nz rwan074@aucklanduni.ac.nz

Abstract

We present and compare various methods to construct efficient QUBO formulations for the Graph Isomorphism Problem—one of a very few problems in NP that is neither known to be solvable in polynomial time nor NP-complete—and two related Subgraph Isomorphism Problems that are NP-hard. Experimental results on two QUBO formulations of the Graph Isomorphism Problem suggest that our direct formulation is more practical than the others with respect to running on the D-Wave architecture.

Keywords: Adiabatic quantum computing, Quadratic Unconstrained Binary Optimization, Chimera graph, Graph Isomorphism Problem, Subgraph Isomorphism Problem.

1 Introduction

The D-Wave computers use quantum annealing to improve convergence of the system's energy towards the ground state energy of a *Quadratic Unconstrained Binary Optimisation* (QUBO) problem. The computer architecture consists of qubits arranged with a host configuration as a subgraph of a *Chimera graph* 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.

D-Wave qubits are loops of superconducting wire, the coupling between qubits is magnetic wiring and the machine itself is supercooled. See more in [13, 5]. The model D-Wave 2X$^{\text{TM}}$ uses a Chimera graph with 1,152 qubits based on $L = 4$ and $M = N = 12$ (out of which 1,098 qubits are active as currently calibrated) chilled close to absolute zero to get quantum effects [21].

The standard way to solve a problem with D-Wave is to find an equivalent QUBO formulation of the problem (or, alternatively, an Ising formulation).

In order to "solve" a QUBO problem with the D-Wave machine the logical qubits have to be "mapped" onto the physical qubits of the Chimera graph of the machine, process known as "embedding". Each logical qubit corresponds (via an embedding) to one or more connected physical qubits, called *chain*. As the number of physical qubits is severely limited, it is desirable to minimise the number of variables, that is, logical qubits, in the QUBO formulation as well as the number of extra physical qubits. The efficiency of an embedding—which is an important component of the efficiency of the overall solution—is measured by the number of resulting physical qubits and the maximum chain size. In this process the density of the QUBO matrix plays an important role.

This paper studies comparatively different methods for constructing efficient QUBO formulations for the Graph Isomorphism Problem, the Subgraph Isomorphism Problem and the Induced Subgraph Problem.

Most, but not all, of our generated QUBO matrices were too big to be embedded in the actual D-Wave 2X chipset; consequently, we have used the D-Wave software package [20] to experimentally compare the efficiency of the QUBO formulations in terms of the number of logical qubits, density of the QUBO matrices and the quality of the embeddings (which relates to the number of physical qubits). The results obtained using the two QUBO formulations of size n^2 for the Graph Isomorphism Problem on graphs of order n suggest that the direct formulation is more practical than the other ones.

Other quantum solutions for (classes of) Graph Isomorphism Problems include quantum walks approaches [3] and quantum annealing [9, 18].

2 Prerequisites

In this section we present the notation used in what follows.

The cardinality of a set X is denoted by $|X|$. By lg we denote the logarithm in base 2 and $\mathbb{Z}_2 = \{0, 1\}$. The power set of X will be denoted by 2^X. A partial function $f\colon X \to Y$ is a function which can be *undefined* for some values $x \in X$: the domain of f, denoted by $\mathrm{dom}(f)$ is the set of all $x \in X$ for which $f(x)$ is not undefined.

A graph $G = (V, E)$ consists 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 labelled 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 $e = uv$ or $e = \{u, v\}$. The number of edges, denoted by m, is called the *size* of G.

In what follows 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 $vv \notin E$.

Discrete optimization problems for the adiabatic quantum (annealing) computing model are specified using the Ising or QUBO models [14]; these equivalent formulations are the standard representations for problems for the D-Wave.

The QUBO is an NP-hard mathematical problem consisting in the minimisation of a quadratic objective function $f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x}$, where $\mathbf{x}$ is an n-vector of binary variables and Q is an upper-triangular $n \times n$ matrix of real numbers:

$$x^* = \min_{\mathbf{x}} \sum_{i \leq j} x_i Q_{(i,j)} x_j, \text{ where } x_i \in \{0, 1\}. \tag{1}$$

The concatenation $\mathbf{z}$ of the vectors $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{m-1})$, is defined as $\mathbf{z} = \mathbf{xy} = (x_0, x_1, \ldots, x_{n-1}, y_0, y_1, \ldots, y_{m-1})$.

A *minor embedding* of a graph $G_1 = (V_1, E_1)$ onto a graph $G_2 = (V_2, E_2)$ is a function $f : V_1 \to 2^{V_2}$ that satisfies the following three conditions:

1. The sets of vertices $f(u)$ and $f(v)$ are disjoint for $u \neq v$.
2. For all $v \in V_1$, there is a subset of edges $E' \subseteq E_2$ 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.

For the minor embedding f defined above, G_1 is referred to as the *guest graph* while G_2 is called the *host graph*. We view a QUBO matrix Q as a weighted adjacency matrix (guest graph) to be embedded onto the D-Wave's Chimera graph (host graph).

3 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 [10] 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 [15]. L. Babai posted a paper [1] 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 the 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 $f : V_1 \to V_2$ (which is calculated); edge-invariant means 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 the decision version of 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 $|V_1| = |V_2|$ and $|E_1| = |E_2|$.

Question: Determine whether there exists a bijective edge-invariant vertex mapping (isomorphism) $f : V_1 \to V_2$.

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

3.1 Earlier approaches for solving the Graph Isomorphism Problem

The paper [12] includes the first-known Ising formulation for the Graph Isomorphism Problem.

Another approach is to develop an Integer Programming (IP) formulation then use the standard polynomial-time reduction techniques to convert to QUBO (see [6, 5]).

Recall the input consists of two graphs G_1 and G_2 with both being of order n and size m. For a simple IP formulation we used the following $n + 2m$ integer variables (later to be converted to binary 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 (oriented) 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:

$$0 \leq v_i < n, \text{ for all } 0 \leq i < n. \tag{2}$$

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

$$(v_i - v_j)^2 > 0, \text{ for all } 0 \leq i < j < n. \tag{3}$$

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:

$$nv_i + v_j = x_k, \text{ for } i \neq j, (i,j) \in E_1^* \text{ and } \operatorname{rank}(i,j) = k. \tag{4}$$

Note that constraints (2)–(4) 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 $\operatorname{rank}(a,b) = l$:

$$\Pi_{y_l \in E_2^*}(x_k - y_l) = 0, \text{ for all } x_k. \tag{5}$$

The constraints given in (5) 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 [6]): 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 $\mathbf{x} = (x_0, x_2, \ldots, x_{n-1})$.

Theorem 1. *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_iv_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 (5), 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 [5], we need at most $\lg n$ binary slack variables for constraints of type (2) and at most $2 \log n$ binary slack variables for constraints of type (3). The other constraints are already in equality form. Thus the total number of binary variables is $O(m^3 \lg n)$. □

A Python script, corresponding to Theorem 1, for calculating the exact number of qubits required for testing the isomorphism of graphs of order n and size m is given in B. The correctness is illustrated in more detail in the following example.

3.1.1 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.

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

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

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

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

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 (2) and 6 additional for constraints of type (3). 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 (3) 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 (5). Thus, 6+12+6+6+36+124=190 total qubits required for the final QUBO matrix for this input.

3.2 A direct QUBO formulation

We present an improved, direct QUBO objective function F for the Graph Isomorphism Problem. The formulation requires n^2 binary variables represented by a binary vector $\mathbf{x} \in \mathbb{Z}_2^{n^2}$:

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

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 $e_{i,j}$, $0 \le i < n$ and $0 \le j < n$: $e_{i,j} = 1$ if $ij \in E_2$.

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 $f(u)f(v) \notin E_2$ or if there is a non-edge $uv \notin E_1$ such that $f(u)f(v) \in E_2$.

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

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

where

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

and

$$P_{i,j}(\mathbf{x}) = \sum_{0\le i'<n}\left(x_{i,i'} \sum_{0\le j'<n} x_{j,j'}(1 - e_{i',j'})\right). \tag{8}$$

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 $D : \mathbb{Z}_2^{n^2} \to \mathcal{F}$ 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 $\mathbf{x} \in \mathbb{Z}_2^{n^2}$ that can be 'decoded' into a bijective function f:

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

$$\left.\text{and} \sum_{0\le i<n} x_{i,i'} = 1,\ \text{for all } 0 \le i' < n\right\},$$

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

Let $I(v)$ denote the set of edges incident to the vertex v. The term $x_{i,i'}x_{j,j'}$ in the right-hand side of (8) 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 (8) can be given without $e_{i',j'}$:

$$P_{i,j}(\mathbf{x}) = \sum_{0\le i'<n} \left(x_{i,i'} \sum_{i'j' \notin I(v_{i'})} x_{j,j'} \right). \tag{9}$$

The following two lemmata will be used to prove the correctness of the objective function F in (6).

Lemma 2. *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,

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

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, that is, every vertex $v \in V_1$ has an image.

Second, with the same argument,

$$\sum_{0\le i'<n} \left(1 - \sum_{0\le i<n} x_{i,i'}\right)^2 = 0 \tag{11}$$

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 (10) and (11) 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 $\mathbf{x} \in \mathbb{Z}_2^{n^2}$ 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 (8) does not contain cubic terms, so, as all $e_{i',j'}$ are constants, $P_{i,j}(\mathbf{x})$ contains only quadratic terms (see also (9)); consequently, $P_{i,j}(\mathbf{x}) \geq 0$, 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 (8), we get

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

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

Assume that $\sum_{ij \in E_1} P_{i,j}(\mathbf{x}) \neq 0$, i.e. for some $ij \in E_1$ we have $P_{i,j}(\mathbf{x}) \neq 0$. It is easy to see that $P_{i,j}(\mathbf{x}) \neq 0$ if and only if $x^*_{i,i'} x^*_{j,j'}(1 - e_{i',j'}) \neq 0$, 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. □

Using Lemmata 2 and 3 we now prove the main result of the section.

Theorem 4. *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 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. □

3.2.1 Populating the QUBO matrix

The matrix QUBO can only contain quadratic terms, so some terms of $F(\mathbf{x})$ have to be modified in such a way that this condition is satisfied and the optimal solutions of (1) are preserved. Two operations will be performed. First, any constant term is ignored because removing it does not modify the optimal solutions of (1): the value of $F(\mathbf{x})$ is reduced by a constant amount for all $\mathbf{x} \in \mathbb{Z}_2^n$. Second, as all variables $x_{i,i'}$ are binary, we replace $x_{i,i'}$ with $x_{i,i'}^2$ for all $x_{i,i'}$ with no effect on the value of $F(\mathbf{x})$.

3.2.2 An example: the graph P_3 revisited

We use the same instances of G_1 and G_2 as described in Section 3.1.1. 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 (7), we have the following penalty terms:

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

Using definition of $P_{i,j}$ given by equation (8) 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 (8), we obtain the following penalty terms:

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

By the definition of a QUBO problem given in Section 2, the objective function $F(\mathbf{x})$ can only contains quadratic terms. Therefore 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 3.2.1, the constant term 1 can be ignored. Second, all linear terms of the form $cx_{i,i'}$ will be replaced by $cx_{i,i'}^2$. 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-triangular 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 1.

Table 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. In our case, we have removed a constant value of 6 from $F(\mathbf{x})$ when encoding it into Q. As a result, the value of the optimal solution of $f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$ will be decreased by 6.

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

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

3.2.3 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. The graph C_4 consists of $V = \{0, 1, 2, 3\}$ and

$$E = \{\{0, 1\}, \{1, 2\}, \{2, 3\}, \{0, 3\}\}.$$

Let $G_1 = G_2 = C_4$. Applying the procedure described in Section 3.2.1 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 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 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 these ideas, we find the eight optimal solutions of the equation $x^* = \min_{\mathbf{x}} F(\mathbf{x})$:

Table 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

$$\begin{aligned}
\mathbf{x}_1 &= (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1), & \mathbf{x}_2 &= (1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0),\\
\mathbf{x}_3 &= (0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0), & \mathbf{x}_4 &= (0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0),\\
\mathbf{x}_5 &= (0,0,1,0,0,0,0,1,1,0,0,0,0,1,0,0), & \mathbf{x}_6 &= (0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1),\\
\mathbf{x}_7 &= (0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0), & \mathbf{x}_8 &= (0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0),
\end{aligned}$$

each of which gives an isomorphism between G_1 and G_2.

3.3 Isomorphism formulation via reduction to the Clique Problem

In this section we give an alternative 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 [2]. Recall that a *clique* of a graph $G = (V, E)$ 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 (induced by) $V' \subseteq V$ of size k?

The maximum clique of a graph is called the *clique number*, denoted by $\chi(G)$. For our construction we use some ideas developed in [16]. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ define the associated graph product $\Psi(G_1, G_2)$ with vertices $V = V_1 \times V_2$ and edges $E = \{((a, b), (c, d)) \in V \times V \mid a \neq c, b \neq d \text{ and } ac \in E_1 \Leftrightarrow bd \in E_2\}$

Theorem 5. *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 $(c, f(c))$, using the definition of E we must have $ac \in E_1 \Leftrightarrow f(a)f(c) \in E_2$. $\square$

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-triangular 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 6. *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 $\sum_{i \leq j} x_i Q_{(i,j)} x_j$ 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 will increase by 1 for the diagonal term $x_i Q_{(i,i)} x_i$. The sum has globally decreased by at least 1 in contradiction with 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. $\square$

3.3.1 Example: the graph P_3 revisited

We use the same instances of G_1 and G_2 as described in Section 3.2.2. The associated product graph with nine vertices is given in Figure 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.

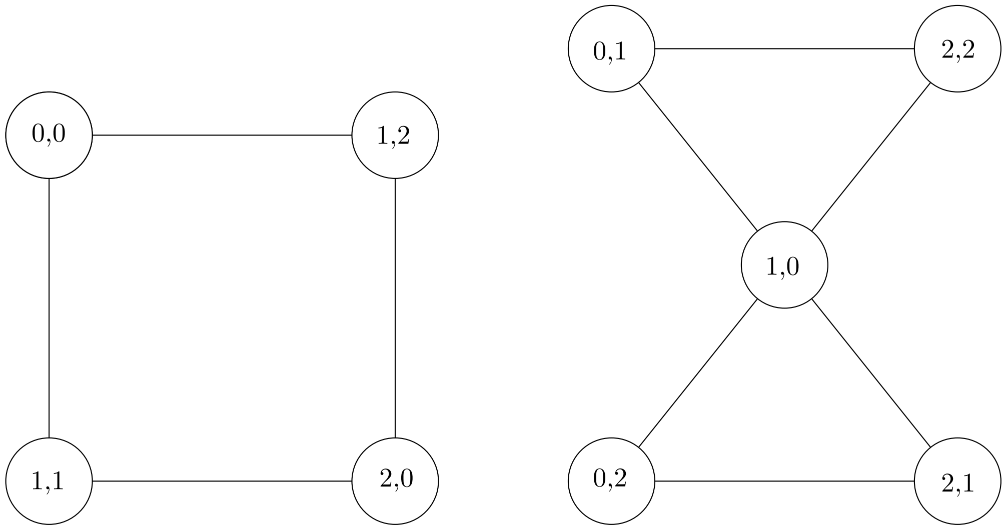

Figure 1: The graph $\Psi(G_1, G_2)$.

The QUBO matrix for the Clique Problem applied to this graph $\Psi(G_1, G_2)$ is given in Table 3.

Table 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

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 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.

4 Minor embedding comparison

We ran several experiments to investigate the difference hardware embeddings can make between the clique and direct formulations. Most of our test graphs were too big to be embedded in the actual D-Wave 2X chipset, hence we embedded these cases in two larger Chimera graphs. Consequently, most of our results have no direct relevance for the performance of D-Wave 2X, but could be useful for future versions of the D-Wave machine [19] which will have more qubits.

The first additional host graph is a Chimera graph with $7,200$ vertices and $21,360$ edges ($L = 4$, $M = N = 30$). The second is a Chimera graph with $6,800$ vertices and $31,680$ edges ($L = 8$, $M = N = 20$). For convenience, we will call them $host_1$ and $host_2$, respectively. The hardware structure of actual D-Wave quantum computers has blocks of $K_{4,4}$ graphs; $host_2$ graphs have blocks of $K_{8,8}$ graphs, which doubles the connectivity (number of edges) inside each block. As a result, $host_2$ has about 50% more edges than $host_1$. We have purposely chosen $K_{8,8}$ in order to check whether a substantial increase in the connectivity inside each block (which seems to be a challenging engineering task) leads to better embeddings.

In our experiments we used the minor embedding algorithm provided by the D-Wave software package [20]. The input graphs are the graphs studied in [6] (see also the supplemental data of its corresponding CDMTCS report). A random permutation of the vertices of each graph G_1 was generated to obtain G_2, hence the graphs G_1 and G_2 are always isomorphic. The QUBO instances for the clique and direct formulations have been generated using scripts in C–E (also see [11]). As the minor embedding algorithm is very time consuming, we run two trials on each test case and only the best result (the one with the smallest number of physical qubits) is shown in the following tables. The algorithm was terminated after several minutes, even if a minor embedding was not found. By using a 'verbose option' we could judge whether progress towards a better solution seemed to be possible.

4.1 Results

In Table 4, the second and third columns contain the *order* and *size* of the input graphs G_1 and G_2. The next two columns contain the number of variables of the QUBO instances obtained through the three different approaches described in Sections 3.1–3.3. Note, for the fifth column, the number of variables obtained in Section 3.2 and Section 3.3 are the same. The last two columns contain the densities of the QUBO instances for the direct and clique formulations. The density of a QUBO instance is defined as the number of non-zero entries in $Q_{(i,j)}$ with $i < j$, divided by the total number of entries in the same part of the matrix. The main diagonal was excluded because the connection of a logical qubit to itself does not affect the minor embedding of the guest graph.

In Table 5, we show the graph cases that could be embedded on the D-Wave 2X machine with 1098 active qubits. In Table 6, we provide the embedding results for both $host_1$ and $host_2$. The column *physical qubits* contains the number of physical qubits required to embed the QUBO instance and *max chain size* is the maximum number of physical qubits a single logical qubit is mapped to.

4.2 Discussion

The IP formulation does not seem very useful as the number of variables (logical qubits) required quickly grows. The same behaviour was noted also for the IP formulation of the Broadcast Time Problem [6]. In spite of the fact that the number of logical qubits is the same for the clique and direct formulations, 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 formulation. One would have expected that the QUBO instance of the direct formulation is easier to embed on the host graph. Due to the non-deterministic nature the minor embedding algorithm—a heuristic algorithm [4]—there are several test cases where the direct formulation generates more physical qubits than the clique formulation.

The results obtained show that the direct QUBO formulation requires less physical qubits to embed on the host than the clique approach and the embedding max chain size is shorter too. This pattern becomes more visible as the QUBO instance gets larger. We suspect that the difference in densities is related to the fact that the clique approach does not make any assumptions on the two input graphs G_1 and G_2 (with the same order and size). In other words, the direct formulation uses more information of the input, hence better results. Both formulations get better results than the more generic IP approach.

The entries in Table 6 marked by a dash '-' 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 increase in the number of physical qubits required. More precisely, as the order of input graphs increase by one, the number of physical qubits approximately doubles. In spite of the large increase in the connectivity or $host_2$, the algorithm was not able to embed any test cases of order bigger than 12 (QUBO of size 144). With $host_1$, the largest embeddable case has order 11 (QUBO of size 121).

Previous experimental studies (see [8, 17]) have shown that bigger embedding chains are correlated with less accurate results. The large embedding chains which appeared in both test cases can be a reason to expect relatively poor performance of the machine.

5 The Subgraph Isomorphism Problem

The Subgraph Isomorphism Problem—a generalisation of the Graph Isomorphism Problem—is the following NP-hard problem.

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-preserving injective function $f : V_1 \to V_2$.

Note that for this problem the function f is not necessarily edge-invariant: it has only to be "edge-preserving", that is, for every $uv \in E_1$ we have $f(u)f(v) \in E_2$.

5.1 A direct QUBO formulation

The objective function (6) can be modified to solve the Subgraph Isomorphism Problem using the same method as in Section 3.2. 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 $\mathbf{z} \in \mathbb{Z}_2^{(n_1+1)n_2}$: $\mathbf{z} = \mathbf{xy}$.
Let

$$F(\mathbf{z}) = H(\mathbf{z}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}), \tag{12}$$

where

$$H(\mathbf{z}) = \sum_{0 \le i < n_1} \left(1 - \sum_{0 \le i' < n_2} x_{i,i'}\right)^2 + \sum_{0 \le i' < n_2} \left(1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'}\right)^2, \tag{13}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{0 \le i' < n_2} \left(x_{i,i'} \sum_{0 \le j' < n_2} x_{j,j'}(1 - e_{i',j'})\right). \tag{14}$$

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

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

and

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

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

Lemma 7. *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. The proof is similar in spirit to the proof of Lemma 2, but different in detail. 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,

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

so $H(\mathbf{z}) = 0$ if and only if both components are equal to 0. First,

$$\sum_{0 \le i < n_1} \left(1 - \sum_{0 \le i' < n_2} x_{i,i'}\right)^2 = 0 \tag{15}$$

if and only if for each $0 \le i < n_1$, 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,

$$\sum_{0 \le i' < n_2} \left(1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'}\right)^2 = 0 \tag{16}$$

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 $\{x_{i,i'} \mid 0 \le i < n_1\}$ have value of 1.

In the first case, we have $1 - \sum_{0 \le i < n_1} x_{i,i'} = 1$, so setting $y_{i'} = 1$ will avoid the penalty. By assumption, $n_1 \le n_2$, 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 (15) 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 $\{x_{i,i'} \mid 0 \le i < n_1\}$ 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, $1 - \sum_{0 \le i < n_1} x_{i,i'} - y_{i'} = 0$ 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 (15) and (16) 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. □

In the proof of Lemma 3, 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 8. *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 9. *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 4, the statement of the theorem is a direct consequence of Lemmata 7 and 8. □

5.2 Another direct QUBO formulation

An alternative formulation with $n_1 n_2$ variables can be obtained using

$$F(\mathbf{z}) = H(\mathbf{z}) + b \sum_{ij \in E_1} P_{i,j}(\mathbf{x}),\ b \in \mathbb{R}^+ \tag{17}$$

where

$$H(\mathbf{z}) = a \sum_{1 \le i \le n_1} (1 - \sum_{1 \le i' \le n_2} x_{i,i'})^2 + \sum_{1 \le i' \le n_2} (1 - \sum_{1 \le i \le n_1} x_{i,i'})^2,\ a > 1, \tag{18}$$

and

$$P_{i,j}(\mathbf{x}) = \sum_{1 \le i' \le n_2} \left(x_{i,i'} \sum_{1 \le j' \le n_2} x_{j,j'}(1 - e_{i',j'}) \right). \tag{19}$$

This formulation has no extra variables but the result is weaker because it requires a post-processing. Note that the constant b has to be sufficiently larger than a (for a proof see [11]).

Theorem 10. *If G_1 is a subgraph of G_2, then solving $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ will produce a valid vertex mapping.*

We also have the following corollary as a direct consequence (contrapositive) of Theorem 10.

Corollary 11. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs, and let $x^* = \min_{\mathbf{x}} F(\mathbf{x})$. 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.*

5.2.1 Post-processing verification

Theorem 10 is a weaker form of Theorem 9. To demonstrate the difference, suppose we have $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let $x^* = F(\mathbf{x}^*)$ and $y^* = F'(\mathbf{y}^*)$ be the optimal solutions and their corresponding variable assignments obtained using the objective functions (12) and (17), respectively.

If $x^* = 0$, then by Theorem 9, G_1 is a subgraph of G_2. As the correctness of f is encoded in x^* we do not need to generate the actual mapping $f : V_1 \to V_2$ using the decoder function D to verify its correctness. This is not the case with y^* and $\mathbf{y}^*$. Theorem 10 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 that the mapping f encoded in $\mathbf{y}^*$, $f = D(\mathbf{y}^*)$, is injective and edge-preserving. Both steps can be done efficiently and the overall verification can be performed in polynomial time. More precisely, we check i) in $O(n_1 n_2)$ time that the solution is an injective function by verifying that $\mathbf{x}$ is in $\mathrm{dom}(D)$, and ii) in $O(n_1 n_2 + m_1)$ time (assuming we build a lookup-table for f and use adjacency matrix representations of graphs) the edge-preserving condition by verifying that each edge $uv \in E_1$ is mapped to an edge $f(u)f(v) \in E_2$.

5.3 A formulation via reduction to the Clique Problem

In this section we give an alternative QUBO formulation that requires n_1n_2 binary variables for checking if a graph of order n_1 is a subgraph of a graph of order n_2. To this aim we slightly modify the product graph construction given in Section 3.3.

For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ we define the associated graph product $\Psi'(G_1, G_2)$ having the vertices $V = V_1 \times V_2$ and the edges $E = \{((a, b), (c, d)) \in V \times V \mid a \neq c, b \neq d$ and $(ac \notin E_1$ or $bd \in E_2)\}$.

Theorem 12. *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. $\square$

Finally, to solve the Subgraph Isomorphism Problem we just construct a QUBO instance for the Clique Problem as in Section 3.3.

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 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 $f : V_1 \to V_2$.

6.1 A direct formulation

We can extend the QUBO formulation given for the Subgraph Isomorphism Problem in Section 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 5.1: $\mathbf{x}$ encodes the injective function f, $\mathbf{y}$ counter balances unnecessary penalties. The decoder function $D : \mathbb{Z}_2^{n_1 n_2} \to \mathcal{F}$ described in Section 5.1 can be used again to obtain the actual mapping.

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

$$F(\mathbf{z}) = H(\mathbf{z}) + \sum_{ij \in E_1} P_{i,j}(\mathbf{x}) + \sum_{ij \notin E_1} N_{i,j}(\mathbf{x}), \tag{20}$$

$$H(\mathbf{z}) = \sum_{0 \le i < n_1} (1 - \sum_{0 \le i' < n_2} x_{i,i'})^2 + \sum_{0 \le i' < n_2} (1 - \sum_{0 \le i < n_1} x_{i,i'} - y_i)^2, \tag{21}$$

$$P_{i,j}(\mathbf{x}) = \sum_{0 \le i' < n_2} \left(x_{i,i'} \sum_{0 \le j' < n_2} x_{j,j'} (1 - e_{i',j'}) \right), \tag{22}$$

and

$$N_{i,j}(\mathbf{x}) = \sum_{0 \le i' < n_2} \left(x_{i,i'} \sum_{0 \le j' < n_2} x_{j,j'} e_{i',j'} \right). \tag{23}$$

The parts $H(\mathbf{z})$ and $\sum_{ij \in E_1} P_{i,j}(\mathbf{x})$ serve the same purpose as in Equation (12), 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 (12), Lemmata 7 and 8 hold and can be proved with the same argument as in Section 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 13. *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 expressions $P_{i,j}(\mathbf{x})$ 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 8 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 (8) 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 8 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 14. *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 7, 8 and 13, 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 7 and 13, at least one of the requirement for an induced subgraph isomorphism is not met, hence f is not an induced subgraph isomorphism.

□

6.2 Formulation via reduction to the Clique Problem

We now give another QUBO formulation that uses $n_1 n_2$ binary variables for checking whether a graph of order n_1 is an induced subgraph of a graph of order n_2. For this we can use the same product graph construction given in Section 3.3. In the proof of Theorem 5 we do not actually need that f is a bijection, we only require injectivity. As a result we have the following corollary.

Corollary 15. *Consider the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $n_1 = |V_1| \leq |V_2| = n_2$. Then, G_1 is an induced subgraph of G_2 if and only if $\chi(\Psi(G_1, G_2)) = n_1$.*

The Induced Subgraph Isomorphism Problem can be solved using Corollary 15 and a QUBO instance for the Clique Problem.

6.2.1 Example: the graphs P_3 and C_3

Consider again $G_1 = P_3$ with edges $E_1 = \{\{0, 1\}, \{1, 2\}\}$, but now with $G_2 = C_3$ with edges $E_2 = \{\{0, 1\}, \{0, 2\}, \{1, 2\}\}$. The associated product graph with nine vertices is given in Figure 2. This time we see that the maximum clique is of size 2. Thus, P_3 is *not* an induced subgraph of C_3 since we would require a clique (mapping set) of size 3.

7 Conclusions and open problems

We have presented different methods for constructing efficient QUBO formulations for the Graph Isomorphism Problem, the Subgraph Isomorphism Problem and the Induced Subgraph Isomorphism Problem.

We experimentally compared the efficiency of two QUBO formulations of the Graph Isomorphism Problem. Efficiency was measured in terms of the number of logical qubits and physical qubits, along with the quality (size of max chains) of embeddings.

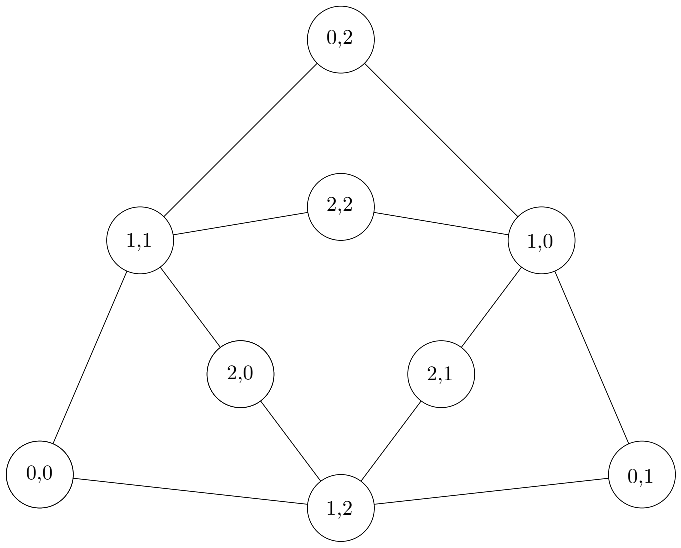

Figure 2: The graph $\Psi(P_3, C_3)$.

Because relevant test cases are too big to be embedded in the actual D-Wave 2X chipset, we used larger Chimera graphs: the first one is a Chimera graph with $7,200$ vertices and $21,360$ edges and the second one is a Chimera graph with $6,800$ vertices and $31,680$ edges. The expectation—that the second denser Chimera graph will allow for better embeddings—was not emphatically confirmed by the results obtained. This may suggest some foreseeable scalability issues with the Chimera graphs. This conjecture has to be further studied.

We conjecture that the lowest number of logical qubits required for the general Graph Isomorphism Problem for two graphs of order n is n^2. Another open problem is to reduce the density of the resulting QUBO matrices by exploiting other known properties of the input graphs, such as their degree sequences, which limits the total number of feasible bijections.

Acknowledgement and dedication

We thank A. Abbott, N. Allen, K. Pudenz and the anonymous referees for critical comments and suggestions. R. Hua thanks the Department of Computer Science for a PBRF Research Assistantship Award to work on this project. This project was supported in part by the Quantum Computing Research Initiatives at Lockheed Martin.

Calude dedicates his contribution to this paper to the memory of his teacher, supervisor, mentor, collaborator and friend, Solomon Marcus (1925-2016).

Bibliography

References

[1] L. Babai. Graph isomorphism in quasipolynomial time, accessed 22 March 2016 (11 December/19 January 2015/2016). Extended abstract in *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing* 2016, 684–697.

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

[3] S. D. Berry and J. B. Wang. Two-particle quantum walks: Entanglement and graph isomorphisms, *Physical Review A* 83 (2011), 042317.

[4] J. Cai, W. G. Macready, A. Roy. A practical heuristic for finding graph minors, http://arxiv.org/abs/1406.2741, 2014.

[5] C. S. Calude, E. Calude, M. J. Dinneen. Adiabatic quantum computing challenges, ACM SIGACT News 46 (1) (2015) 40–61.

[6] C. S. Calude, M. J. Dinneen. Solving the broadcast time problem using a D-Wave quantum computer, in: A. Adamatzky (Ed.), Advances in Unconventional Computing, Vol. 22 of Emergence, Complexity and Computation, Springer, 2016, 439–453.

[7] C. S. Calude, M. J. Dinneen and R. Hua. QUBO Formulations for the Graph Isomorphism Problem and Related Problems, Tech. Rep. CDMTCS-499, v. 2, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand (2017).

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

[9] F. Gaitan and L. Clark. Graph isomorphism and adiabatic quantum computing, *Physical Review A* 89 (2014), 022342.

[10] M. R. Garey, D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, 1979.

[11] R. Hua. Adiabatic Quantum Computing with QUBO Formulations, MSc. Thesis, University of Auckland, 2016.

[12] A. Lucas. Ising formulations of many NP problems, Frontiers in Physics 2, 5 (2014), doi:10.3389/fphy.2014.00005.

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

[14] C. McGeoch, C. Wang. Experimental evaluation of an adiabatic quantum system for combinatorial optimization, in: *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, ACM, New York, 2013, 23:1–23:11.

[15] B. D. McKay. Practical graph isomorphism, Congressus Numerantium 30 (1981), 45–87.

[16] M. Pelillo. Replicator equations, maximal cliques, and graph isomorphism, *Neural Computation* 11 (1999), 1933–1955.

[17] K. L. Pudenz, T. Albash, D. A. Lidar. Error-corrected quantum annealing with hundreds of qubits, Nature Communications 3243 (2014).

[18] K. M. Zick, O. Shehab and M. French. Experimental quantum annealing: case study involving the graph isomorphism problem, *Nature Scientific Reports* 2015, `http://dx.doi.org/10.1038/srep11168`.

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

[20] D-Wave. Developer Guide for Python, Tech. Rep. 09-1024A-A, D-Wave Systems, Inc., Python Release 2.3.1 (for Mac/Linux) (2016).

[21] D-Wave. The D-Wave 2X$^{\text{TM}}$ System. Accessed: 2017-03-17. (2017), `https://www.dwavesys.com/d-wave-two-system`.

A Experimental results

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

Table 5: Minor embedding results for the current D-Wave 2X architecture

Graph	Lucas' Method [12]		Clique Method		Direct Method	
	max chain	phys qubits	max chain	phys qubits	max chain	phys qubits
Bull	12	224	11	227	8	179
Butterfly	12	227	10	218	9	162
C4	5	58	4	56	6	70
C5	12	237	11	198	10	185
C6	21	594	18	525	15	433
Diamond	5	61	6	66	6	67
Grid2x3	21	522	21	538	20	446
House	11	228	14	227	9	170
K2	1	4	1	4	1	4
K3	3	23	3	20	3	20
K4	5	61	5	60	5	62
K5	8	151	9	150	9	150
K6	12	322	11	312	14	300
K7	19	575	19	620	18	639
K2,3	9	172	11	200	12	188
K3,3	22	538	18	492	13	380
K3,4	24	896	27	920	23	778
Octahedral	14	402	18	440	15	345
S2	4	23	4	23	3	22
S3	6	77	7	77	5	68
S4	11	206	11	223	10	186
S5	18	534	20	449	19	383

Table 6: Minor embedding results for larger Chimera graphs

	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
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
C10	4216	3402	1997	1857	66	51	31	28
C11	5996	5685	3534	2699	79	76	44	30
C12	-	-	5579	4735	-	-	66	47
Chvatal	-	-	5440	4710	-	-	55	53
Diamond	70	65	44	41	6	5	3	4
Dinneen	3021	2232	1283	1363	64	37	21	24
Durer	-	-	4833	4961	-	-	54	51
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
Grotzsch	6204	5990	3225	3404	83	95	47	43
Herschel	6059	5986	3181	3043	75	80	41	40
Hexahedral	1754	1385	812	857	38	31	19	17
House	210	178	117	110	10	8	6	7
Icosahedral	-	-	4738	4775	-	-	53	46
K2	4	4	4	4	1	1	1	1
K3	20	20	12	12	3	3	2	2
K4	58	60	44	43	4	5	3	3
K5	148	150	95	94	7	8	5	5
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
K10	2513	2381	1361	1233	54	43	21	18
K2,3	192	179	96	92	12	8	5	6
K3,3	312	380	173	187	11	14	7	7
K3,4	765	698	319	329	21	19	9	10
K4,4	1792	1205	558	715	41	26	10	17
K4,5	1622	2019	1373	954	26	34	28	19
K5,5	5178	3342	2178	1651	67	41	28	20
K5,6	6385	5553	3216	2498	83	78	42	32
K6,6	-	-	4856	4182	-	-	50	50
Krackhardt	4132	4168	2133	1871	67	55	30	28
Octahedral	434	306	212	196	15	12	8	7
Petersen	5030	4044	2299	2444	78	60	33	33
Q3	1953	1671	854	807	41	37	19	17
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
S10	5935	4956	2801	2267	79	70	35	31
Tietze	-	-	4648	3892	-	-	50	55
Wagner	1728	1408	875	718	43	31	21	18

B Python program to count the IP formulation variables

```
#!/usr/bin/env python
# IP formulation of Graph Isomorphism problem (count binary
   variables)

import sys,math

assert len(sys.argv)==3
n = int(sys.argv[1]) # graph order
m = int(sys.argv[2]) # graph size

# number of binary variables to represent vertex (constraints
   1); also their slack
L1 = int(math.log(2*n-1,2)) # store ints 0,1,...,n-1

# number of binary variables to represent slack for diff
   squared (constraints 2)
L2 = int(math.log(2*(n-1)*(n-1)-1,2)) # store ints 0,1,...,(n
   -1)^2-1

# number of binary variables to represent edge index x_i in E^*
    (constraints 3)
L3 = int(math.log(2*(n*n-2)-1,2)) # store 1,...,n^2-2

print L1, L2, L3

print 'total binary variables (before products):', 2*n*L1 + n*(
   n-1)/2*L2 + 2*m*L3

# product variables for constraints 2
P2 = (n*L1)**2

def nCr(n,r):
    f = math.factorial
    return f(n) / f(r) / f(n-r)

# product variables for constraints 4
P4 = 2*m*sum( nCr(L3+i-1,i) for i in range(2,2*m+1) )

print 'total binary variables (constraints):', \
      2*n*L1 + n*(n-1)/2*L2 + 2*m*L3 + P2 + P4
```

listings/IPv2.py

C Python program to create Ψ product graph

```
#!/usr/bin/env python3
#
# 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):  # CS220 adjacency list
   format
    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)==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

D Python program to create clique QUBO

```
#!/usr/bin/env python
# Max Clique graph to QUBO Hamiltonian objective

import sys, networkx as nx

def read_graph(infile=sys.stdin):  # CS220 adjacency list
   format
    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()

print n
```

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

listings/cliqueQUBO.py

E Python program to create Direct Isomorism QUBO

```
#!/usr/bin/env python3
#
# direct QUBO formulation for graph iso problem

import networkx as nx
import sys, math

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()

assert G1.order()==G2.order()
assert G1.size()==G2.size()

def generateQUBO(G1, G2):
    n = G1.order()
    varsDict = {}
    edgeDict = {}
    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
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
# -2 sum xi,i'
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


    print(n*n)
    for i in range(n*n):
        for j in range(n*n):
            print(Q[i,j], end=' ')
        print()

    return Q,varsDict,n*n


QUBO = generateQUBO(G1,G2)
```

listings/isomorphismQUBO.py

F Python program to create Lucas' QUBO

```
#!/usr/bin/env python3
#
# Lucas 2014 QUBO formulation for graph iso problem

import networkx as nx
import sys, math

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()

assert G1.order()==G2.order()
assert G1.size()==G2.size()

def generateQUBO(G1, G2):
        n = G1.order()
        varsDict = {}
        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
        # -2 sum xi,i'
        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

    # HB part 1
    for i in range(n):
        for j in range(i,n):
            if i not in G2[j]: continue
            for (u,v) in G2.edges():
                x1=varsDict[(u,i)]
                x2=varsDict[(v,j)]
                Q[x1,x2] += 2
                x1=varsDict[(v,i)]
                x2=varsDict[(u,j)]
                Q[x1,x2] += 2

    # HB part 2
    for (i,j) in G1.edges():
        for u in range(n):
            for v in range(u,n):
                if u not in G2[v]: continue
                x1=varsDict[(u,i)]
                x2=varsDict[(v,j)]
                Q[x1,x2] += 2
                x1=varsDict[(u,j)]
                x2=varsDict[(v,i)]
                Q[x1,x2] += 2

    # 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

    print(n*n)
    for i in range(n*n):
            for j in range(n*n):
                    print(Q[i,j], end=' ')
            print()
        return Q,varsDict,n*n

QUBO = generateQUBO(G1,G2)
```

listings/lucasQUBO.py