Formulating Graph Covering Problems for Adiabatic Quantum Computers

Michael J. Dinneen
Rong Wang

Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-495
February 2016 (Updated August 2016)

Formulating Graph Covering Problems for Adiabatic Quantum Computers

Michael J. Dinneen and Rong Wang

Department of Computer Science, University of Auckland, Auckland, New Zealand

mjd@cs.auckland.ac.nz   rwan074@aucklanduni.ac.nz

Abstract

We provide efficient quadratic unconstrained binary optimization (QUBO) formulations for the Dominating Set and Edge Cover combinatorial problems suitable for adiabatic quantum computers, which are viewed as a real-world enhanced model of simulated annealing (e.g. a type of genetic algorithm with quantum tunneling). The number of qubits (dimension of QUBO matrices) required to solve these set cover problems are $O(n + n \lg n)$ and $O(m + n \lg n)$ respectively, where n is the number of vertices and m is the number of edges. We also extend our formulations for the Minimum Vertex-Weighted Dominating Set problem and Minimum Edge-Weighted Edge Cover problem. Experimental results for the NP-hard Dominating Set problem using a D-Wave Systems quantum computer with 1098 active qubit-coupled processors are also provided for a selection of known common graphs.

1 Introduction

Adiabatic quantum computing is based on the process of evolving a ground state of a Hamiltonian representing a problem to a minimum-energy solution state [12, 11]. It has been shown to be equivalent to the more traditional quantum "circuit model" [2]. Other introductory details about the application of adiabatic quantum computing may be found in [21, 5]. The current family of D-Wave computers can solve problems formulated in either *Ising* form or *Quadratic Unconstrained Binary Optimization* (QUBO) form, defined later. There is a simple translation between the variable spin values -1/+1 of the Ising (physics) model and the binary values 0/1 of QUBO (logic) model (see [7]). The focus of this paper is to use the mathematical QUBO formulation to solve hard combinatorial problems and not to be overly concerned about the actual physics theory required for actual computation. The paper by Lucas [19] provides a good foundation of Ising/QUBO formulations of many hard combinatorial problems. Some of these initial formulations have recently be improved by several authors, including us, motivated by the limitations on the number of actual available qubits in existing machines.

We study two main optimization problems in this paper. One is NP-hard and the other is polynomial-time solvable, but our QUBO formulations are very similar in complexity (e.g. the two problems require $O(n + n\lg n)$ and $O(m + n\lg n)$ qubits respectively for graphs of order n and size m). Given a graph $G = (V, E)$, a *dominating set* D of G is a subset of V, such that for every vertex $v \in V$, either $v \in D$ or $w \in D$, where w is a neighbor of v. An *edge cover* C of G is a subset of E, such that for every vertex $v \in V$, v is incident to at least one edge in C. The two problems defined below involve finding the smallest such D and C, that is, a dominating set with the minimum number of vertices and an edge cover with the minimum number of edges. For convenience, we assume all graphs are connected and have at least one edge.

Dominating Set Problem:

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

Edge Cover Problem:

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

The decision version of the Dominating Set problem was one of the original classic problems included by Garey and Johnson [14]. It is also one of the harder NP-complete problems being classified as W[2]-hard when considering parameterized complexity [3]. An extensive history on this problem may be found in [16]. Contrastly, solving the Edge Cover problem for graphs (without isolated vertices) is easily achievable in polynomial time. This is done by observing that the smallest edge cover is equal to the order of the graph minus its maximum matching size [18].

The paper is organized as follows. In Section 2, we present efficient QUBO formulations, along with proofs of correctness, of the Dominating Set and Edge Cover problems. Then in Section 3, we address the weighted version of the combinatorial problems. Finally, in Section 4 we present our experimental results and some discussion about using actual quantum annealing hardware.

2 QUBO Formulation

QUBO is an NP-hard mathematical optimization problem of minimizing a quadratic objective function $x^* = \mathbf{x}^T Q \mathbf{x}$, where $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ is a n-vector of binary (Boolean) variables and Q is an upper-triangular $n \times n$ matrix. Formally, QUBO problems are of the form:

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

2.1 Dominating Set

We provide a simple QUBO formulation of the Dominating Set problem. The best known exact algorithm to solve the Dominating Set problem has time complexity $O(2^{0.610n})$ [13]. Given a graph $G = (V, E)$ with n vertices, let $V = \{v_0, v_1, \ldots, v_{n-1}\}$, $\Delta(v)$ denote the degree of the vertex v and $N(v)$ denotes the set of neighbors of vertex v. This formulation requires $n + \sum_{v_i \in V}(\lfloor \log(\Delta(v_i)) \rfloor + 1)$ binary variables, that is, for every vertex v_i in G, we need one variable x_i to represent v_i as well as $\lfloor \lg(\Delta(v_i)) \rfloor + 1$ redundant binary variables for each vertex. For the sake of readability, we will label these redundant variables as $y_{i,k}$, where $0 \le k \le \lfloor \log(\Delta(v_i)) \rfloor$. Thus we have a vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}, y_{0,0}, \ldots, y_{n-1, \lfloor \lg(\Delta(v_n)) \rfloor})$ of named variables.

The objective function to be minimized is of the form:

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

where

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

To obtain a solution of the Dominating Set problem, an additional 'decoder' function $D(\mathbf{x}) : \mathbb{Z}_2^{|\mathbf{x}|} \to 2^V$ is required where 2^V is the power set of V. We take $D(\mathbf{x}) = \{v_i \mid x_i = 1\}$, a subset of V as the dominating set. In the objective function, $A > 1$ is a positive real constant, the term $\sum_{v_i \in V} x_i$ represents a penalty for the size of the chosen set, and P_i serves as a penalty if a non-dominating set is chosen. If the assignment of the variables is a dominating set, then for each vertex v_i in G, we have $x_i + \sum_{v_j \in N(v_i)} x_j \ge 1$. And therefore $1 - (x_i + \sum_{v_j \in N(v_i)} x_j) \le 0$. Finally, we use the term $\sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k}$ to counter balance the penalty if more than one vertex in the set $v_i \cup N(v_i)$ is chosen (as it does not violate the definition of a dominating set and should not be penalized). In the worst case, $1 - (x_i + \sum_{v_j \in N(v_i)} x_j) = -\Delta(v_i)$ where v_i and all of its neighbors are chosen, so a total number of $(\lfloor \lg(\Delta(v_i)) \rfloor + 1)$ redundant variables are needed to represent integers up to $\Delta(v_i)$. Hence the total number of binary variables of this formulation is $O(n + n \lg n)$ in the worst case.

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

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

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

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

2.2 Dominating Set Q_3 example

In this subsection, we will provide an example of the QUBO formulation (1) on Q_3. Formally, The hypercube Q_3 is defined as follows. The vertices of Q_3 are $V = \{0, 1, \ldots, 7\}$ and the edges are $E = \{\{0,1\}, \{0,2\}, \{0,4\}, \{1,3\}, \{1,5\}, \{2,3\}, \{2,6\}, \{3,7\}, \{4,5\}, \{4,6\}, \{5,7\}, \{6,7\}\}$. It can be visualized as a 3-dimensional cube where the each corner of the cube is a vertex. Now, by expanding the bracket in objective function (1), we get

$$\begin{aligned}
&\sum_{v_i \in V} x_i + A \sum_{v_i \in V} \Bigg(1 - x_i - \sum_{v_j \in N(v_i)} x_j + \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} - x_i + x_i^2 + x_i \sum_{v_j \in N(v_i)} x_j - x_i \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} \\
&\quad - \sum_{v_j \in N(v_i)} x_j + x_i \sum_{v_j \in N(v_i)} x_j + \Bigg(\sum_{v_j \in N(v_i)} x_j \Bigg)^2 - \sum_{v_j \in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} \\
&+ \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} - x_i \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} - \sum_{v_j \in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} + \Bigg(\sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} \Bigg)^2 \Bigg) \qquad (2)
\end{aligned}$$

Technically, the objective function of a QUBO problem can only contain quadratic terms, so a few terms in (1) have to be modified. Firstly, any constant terms can be ignored. Removing a constant does not have any impact over the optimal solutions of the QUBO problem. As removing this constant (e.g. nA) will reduce the value of the objective function by a fixed amount across all different assignments of all the binary variables, therefore even though the value of the objective function will decrease, the assignment of variable will remain the same regardless. Secondly, all linear terms will be converted into quadratic terms, that is, we will replace all x_i and $y_{i,k}$ by x_i^2 and $y_{i,k}^2$ respectively. Since all variables are binary, we have $x_i = x_i^2$ and $y_{i,k} = y_{i,k}^2$ for $x_i, y_{i,k} \in \{0, 1\}$ so it will not affect the value of the objective function.

After applying the two steps described in the paragraph above and summing up similar terms, we get

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

$$-2x_i\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} + \Bigg(\sum_{v_j\in N(v_i)} x_j\Bigg)^2 - 2\sum_{v_j\in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k} + \Bigg(\sum_{k=0}^{\lfloor \lg(\Delta(x_i))\rfloor} 2^k y_{i,k}\Bigg)^2\Bigg) \tag{3}$$

Now, we can finally obtain a valid matrix representation of the objective function. Let $A = 2$, the matrix representation of the quadratic objective function (3) for Q_3 is shown in Table 1. The entries $Q_{i,j}$ where $i \leq j$ in the matrix is computed by extracting the coefficient of each quadratic term from the objective function.

Table 1: Dominating Set QUBO matrix for Q_3

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

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

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

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

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

and

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

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

2.3 Edge Cover

The QUBO formulation of the Edge Cover problem here is quite similar to the Dominating Set problem given in the previous subsection. The Edge Cover problem can be solved in polynomial time by exploiting the relationship between an Edge Cover and a Maximum Matching [10, 20]. Recall a *matching* in a graph is a set of edges that are not incident to each other.

Proposition 2. *The order of a graph G is equal to the size of its maximum matching plus the size of its minimum edge cover.*

Given a graph $G = (V,E)$ with n vertices and m edges, let $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E = \{e_{i,j} \mid v_j \in N(v_i)\}$. Since digraphs are not considered here, we will take $e_{i,j}$ and $e_{j,i}$ as the same element and only one of them will appear in E for each edge in the graph. We will use $\Delta(v)$ to denote the degree of the vertex v and $I(v)$ to denote the set of edges incident to v. This formulation requires one binary variable $x_{i,j}$ for each $e_{i,j} \in E$, as well as $\lfloor \lg(\Delta(v_i) - 1) \rfloor + 1$ redundant binary variables for each $v_i \in V$.

The objective function to be minimized is of the form:

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

where

$$P_i = \left(1 - \sum_{e_{i,j} \in I(v_i)} x_{i,j} + \sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1) \rfloor} 2^k y_{i,k}\right)^2 \tag{4}$$

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

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

Theorem 3. *The objective function (4) is correct.*

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

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

2.4 Edge Cover S_{15} Example

Similar to the Dominating Set problem, we will provide an example of the actual encoding of the objective function (4) to a QUBO matrix here. The Star graph S_n with $n+1$ vertices is defined as follows. The vertices are $V = \{0, 1, 2, \ldots, n\}$ and the edges are $E = \{\{0, i\} \mid 1 \leq i \leq n\}$. Once again, the objective function (4) can not be encoded straight away into QUBO, and constant and linear terms have to be replaced just like in the Dominating Set problem. Doing so will give us the expression

$$\sum_{e_{i,j} \in E} x_{i,j}^2 + A \sum_{v_i \in V} \Bigg(-2 \sum_{e_{i,j} \in I(v_i)} x_{i,j}^2 + 2 \sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1) \rfloor} 2^k y_{i,k}^2 + \sum_{e_{i,j} \in I(v_i)} x_{i,j} \sum_{e_{i,j} \in I(v_i)} x_{i,j} \\ -2 \sum_{e_{i,j} \in I(v_i)} x_{i,j} \sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1) \rfloor} 2^k y_{i,k} + \Bigg(\sum_{k=0}^{\lfloor \lg(\Delta(x_i)-1) \rfloor} 2^k y_{i,k} \Bigg)^2 \Bigg) \quad (5)$$

The encoded QUBO matrix corresponding to objective function (5) is shown in Table 2. The solution to the minimum Edge Cover problem is trivial for the family of star graphs, since all vertices labeled from 1 to n are all of degree 1 and connected to vertex 0, any edge cover in star graphs would have to consists of all the edges in the graph. By solving $x^* = \min_{\mathbf{x}} \sum_{i \leq j} x_i Q_{(i,j)} x_j$, we obtain an unique solution:

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

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

3 Weighted Problems

The formulations provided in the previous section can be modified quite easily to adapt to weighted graphs. The definitions of the input and output of the Dominating Set and Edge Cover problems are slightly different in weighted graphs. For the Weighted Dominating Set problem, each vertex v_i in the graph is assigned a real positive weight w_i and the goal is to find a dominating set that has a minimum sum of the weights. Likewise, in the Weighted Edge Cover problem, each edge $e_{i,j}$ in G is associated with a real positive weight $w_{i,j}$ and the goal is to find an edge cover that minimizes the sum of the weights of the edges in the edge cover set. Formally, we have the following definitions.

The input to the **Weighted Dominating Set** problem consists of a graph $G = (V, E)$ as well as a weight function $W : V \to \mathbb{R}^+$ that maps each vertex in G to some positive real weights. The *weighted sum* function $S : 2^V \to \mathbb{R}^+$ is defined as $S(A) = \sum_{v \in A} W(v)$. The goal is to find a dominating set D such that $S(D)$ has the minimum value over all possible dominating sets.

Similarly, the **Weighted Edge Cover** problem takes $G = (V, E)$ and $W : E \to \mathbb{R}^+$ as the input. And this time the weighted sum function $S : 2^E \to \mathbb{R}^+$ is defined over a subset of E and $S(A) = \sum_{e \in A} W(e)$. Once again, the goal is to find an edge cover C of G such that $S(C)$ has the minimum value over all possible edge covers.

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

3.1 Weighted Dominating Set

For the Weighted Dominating Set problem, the objective function $F(\mathbf{x})$ is almost identical to the unweighted version. With $w_i = W(v_i)$, we have

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

where

$$P_i = \left(1 - \Big(x_i + \sum_{v_j \in N(v_i)} x_j\Big) + \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k}\right)^2 \tag{6}$$

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

Theorem 4. *The QUBO formulation in (6) is correct.*

Proof. First, we show that it is always possible to transform a non-dominating set into a dominating set with a smaller value in the objective function. Suppose we have $x^* =$

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

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

3.2 Weighted S_5 Example

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

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

Table 2: Edge Cover QUBO matrix for S_{15}

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

The encoded QUBO matrix is shown in Table 3. Solving $x^* = \min_{\mathbf{x}} \sum_{i \leq j} x_i Q_{(i,j)} x_j$ this time gives two different solutions with identical value objective function (6). The two solutions are $\mathbf{x}_1 = [1,0,0,0,0,0,0,0,0,0,0,0,0,0]$ and $\mathbf{x}_2 = [0,1,1,1,1,1,0,0,1,0,0,0,0,0]$. The number of vertices in these two dominating sets is different, $D(\mathbf{x}_1)$ has only one vertex while $D(\mathbf{x}_2)$ has five vertices.

Table 3: Dominating Set QUBO matrix for weighted S_5

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

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

3.3 Weighted Edge Cover

Similar to the Edge Cover problem, the Weighted Edge Cover problem can be reduced to Weighted Perfect Matching problem which is solvable in time $O(n^3)$ [22, 17][1]. As in the previous subsection for the Dominating Set problem, we only need to do some small modification to the Edge Cover problem to obtain a QUBO formulation for the weighted version:

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

where

$$P_i = \left(1 - \sum_{e_{i,j} \in I(v_i)} x_{i,j} + \sum_{k=0}^{\lfloor \lg(\Delta(x_i) - 1) \rfloor} 2^k y_{i,k} \right)^2 \tag{7}$$

Once again, we need to have $A > \max\{w_{i,j} \mid e_{i,j} \in E\}$. The function $C : \mathbb{Z}_2^{|\mathbf{x}|} \to 2^E$ from Section 2.3 will be used again to obtain the subset of edges. Although the argument may

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

seem almost identical to the unweighted version, for the sake of completeness, we will present the theorem and proof formally below.

Theorem 5. *The QUBO formulation in (7) is correct.*

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

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

3.4 Weighted W_5 Example

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

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

Let $A = 20$, the QUBO matrix encoded from objective function (7) is shown in Table 4. Once again, we get two optimal solutions which both have the same value in objective function (7) in this case.

$$\mathbf{x}_1 = [0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$\mathbf{x}_2 = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

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

Table 4: Edge Cover QUBO matrix for weighted W_5

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

4 Experimental Results and Discussion

Experiments were conducted on the D-Wave quantum computer. The chip had 1098 active qubits and could support problems with up to that many variables in theory. In practice, before the QUBO formulations could be executed on the D-Wave computer, an embedding on the hardware must first be found. That is, we need to check if the QUBO instance, referred to as the guest graph, is a graph minor[2] of the actual physical qubit architecture which is an induced subgraph of a specific Chimera host graph. Python scripts which generate the QUBO instances of the objective functions for Dominating Set and Edge Cover are available in Appendices A and B. We used the NetworkX graph package [15] in both scripts in addition to the D-Wave library [8].

4.1 Presentation of Results

For both the Dominating Set and the Edge Cover problem, we did 2500 trials on each of the graphs listed in Tables 5 and 6. The first three columns of Tables 5 and 6 contain standard information about the graphs; the *order* and *size* are with respect to the input graphs rather than the QUBO formulations. We used the same graph specifications as in [6]. The next three columns contains information on the embeddings. *Logical qubits* is the number of

[2]A graph G is a *minor* of a graph H if G is isomorphic to a graph obtained from H by repeatedly deleting vertices, deleting edges or contracting edges.

variables of the formulation and *physical qubits* is the number of hardware qubits required after embedding the QUBO instance into the specific Chimera graph. As can be seen from the table, the difference between the number of variables and the actual number of hardware qubits needed varies quite a lot. The high scaling factor is mostly due to the high density of the QUBO matrices. High density means the size (number of non-zero QUBO entries) of the guest graph which the QUBO matrix represents is high, and since the Chimera graph has a fixed architecture, it is harder to embed such guest graphs with few edge contractions. Hence more active physical qubits are needed. The *embedding max chain* column contains the maximum number of physical qubits a single logical qubit is mapped to.

Note that, deciding minor containment is a well-known NP-complete problem, but since it is not the focus of study here, we did not implement our own embedding algorithm here. The algorithm used in the study is provided by the D-Wave software package; more details about this particular embedding algorithm can be found in [7] and [4]. WE also note that we did not try to minimize the number of variables nor the density of the QUBO matrix when developing the objective functions given in Section 2. It is quite possible that better formulations exist for there problems, that is, formulations with less number of logical qubits and lower density that will make the minor containment problem on them easier to solve.

The *best answer* column is the best (smallest) solution, in terms of the size of the covering set of vertices or edges respectively for each of the problems, the D-Wave machine was able to find. The *optimal answer* column is the true optimal solution of the particular graph. The optimal solutions for the Dominating Set problem was computed by first computing an Integer Programming formulation of the problem and then Sage Maths [23] software was used to compute the solution. A script used for computing the optimal solution is given in Appendix C. The optimal solution to the Edge Cover problem was computed by first realizing that the order of a graph G is equal to the sum of the number of edges in its maximum matching and minimum edge cover [20]. The maximum matching for each test graphs was computed using the built-in function provided by the NetworkX package.

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

4.2 Scaling the Ising

The Python program in Appendix D access the quantum machine when solving $x^* = \min_{\mathbf{x}} f(\mathbf{x})$. Note that we explicitly converted the QUBO formulations of each test cases to its corresponding Ising form as the D-Wave software package API (Application Program Interface) gives more direct control for more fine tuned test with respect to the Ising model. The QUBO to Ising transformation function is also provided by the D-Wave API. We then used two extra parameters s and s_2 to scale the Ising model.

The parameter s_2 is used before the embedding is applied. All entries in the Ising model is scaled linearly by $f(x) : \mathbb{R} \to \mathbb{R}$ where $f(x) = s_2 x/\texttt{maxV}$ and $\texttt{maxV}$ is the maximum value over all entries. This scaling is applied because of the current D-Wave hardware coupling restrictions need to be in the range [-1,1]. After the embedding is computed, $0 < s \leq 1$ is used as a factor to scale all entries corresponding to the same logical qubits. Lower values of s emphasizes that it is more important for physical qubits corresponding to the same logical qubit to be in a consistent state (spin). The lowering the value of s was recommended in [8] if the unscaled case $s = 1$ does not provide expected results.

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

4.3 Embedding with Long Chains

The entries highlighted in red in Table 5 and 6 are the test cases where the optimal solution was never found out of the 2500 trials. An interesting observation we can make here is that in Table 5, all such entries have a max chain length of bigger than or equal to 10. It coincides with our expectation that longer chain lengths are more likely to leads to less accurate solutions. The embedding of the problem can be seen as a transformation to a different problem which has equivalent optimal solutions. Since one logical qubit could be mapped to several different physical qubits, new constraints have to be introduced to enforce all the physical qubits to be in the same consistent states. This is achieved by adding extra penalties if the set of physical qubits, representing the same logical qubits, are in inconsistent (different) states [7]. Therefore, doing such transformation will make the annealing process harder and lead to a lower probability of finding the optimal solution [7].

For for Table 6 some of the highlighted entries have a max chain length of less than 10. However, the best answer for such entries are at most 1 bigger than the true optimal solution. So in some sense, it does not contradict what we stated in the previous paragraph, the shorter the chains are, the more accurate (closer to optimal) the solution becomes.

4.4 The Family of Star Graphs

The three rows with dashes in Table 6 corresponds to the test cases where no valid solution was found out of all 2500 trials.

Recall the definition of the family of Star graphs from Section 2.4. For all $n \in \mathbb{Z}$, the only edge cover of $S_n = (V, E)$ is E. Since all vertices $V \setminus \{0\}$ are of degree 1, and all edges are incident to vertex 0, the only way to cover vertex $1 \leq i \leq n$ is to pick the edge $\{0, i\}$. Hence all edges have to be picked to have a correct covering set. We suspect that this uniqueness about the solution is what leads to its none-discovery in these three cases. The physical nature of the computation could make the desired unique configuration of qubits impossible to reach for the quantum machine. That is why we can see that from Table 6, despite the fact that the Star graphs we used in the experiments being relatively small, in terms of both the order and size, the probability of finding a valid edge cover for S_n decreases dramatically as n increases when compared with other families of graphs we had.

The case of finding a valid dominating set for S_n is slightly different. Since all other vertices are connected with vertex 0, the optimal solution is obviously just the set $\{0\}$. For the dominating set experiment, the optimal solution was found for all S_n we tested. However, a valid dominating set for S_n could also include any number of other vertices. Hence it leads to what we see in Table 5, the probability of finding a valid dominating set is relatively high for all S_n, but the probability for finding the optimal solution once again reduces dramatically as the order increases.

4.5 Final Comments and Future Work

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

Google has recently published a paper that had some experimental result on the new D-Wave 2X [9], the same model we have used, and had much more successful results. Namely, they were able to achieve a 99% success rate in which the optimal solution was found. Although it is difficult to determine what exactly is the cause for such a big difference, the authors did considered several factors that could have lead to this observation. First of all, in the experiment Google did [9] problems sizes vary from 200 to around 1000, in terms of the number of physical qubits. Google had found the best suited annealing time for each case, that is, the annealing time for each test case that has the highest probability of obtaining the optimal solution. We used an annealing time of 20 microseconds for all of our test cases as with a limited access to the machine it was difficult for us to run multiple fine-tuned tests. Secondly, the test cases Google used in their benchmark paper were hand crafted instances fitting directly on the actual hardware. In other words, the underlying graphs they used are subgraphs of the Chimera graph, so the minor containment problem did not need to be considered. As we mentioned in Subsection 4.3, longer chains probably lead to less accurate solutions.

We performed software simulations (e.g. conventional evolutionary search) on our QUBO matrices to verify optimal dominating sets were possible. The conventional evolutionary search algorithm we used was also provided by the D-Wave software package [8]. The authors plan to experiment with new problems whose QUBO matrices have a lower density and possibly with smaller max chain length. Hopefully, the new results can either confirm or deny our suspicion about the chain length. We are also planning on testing the weighted covering problems described in this paper. The weighted version of each problems will obviously have the same embedding as the original problem, the only difference is the weight on each couplers (allowing for *quasical* computations as discussed in [1]). It would be interesting to see to what extend will the coupler weight affect the solutions.

Acknowledgment

This work was supported in part by the Quantum Computing Research Initiatives at Lockheed Martin. We thank Cris Calude for useful comments on an earlier draft.

References

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

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

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

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

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

[6] Cristian S. Calude and Michael J. Dinneen. Solving the broadcast time problem using a D-Wave quantum computer. Technical Report CDMTCS-473, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, November 2014.

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

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

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

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

[11] 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.

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

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

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

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

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

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

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

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

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

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

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

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

Table 5: Results for some small graphs families for Dominating Set.

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

Table 6: Results for some small graphs families for Edge Cover.

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

A Python Program to Generate QUBO Formulation of the Dominating Set Problem

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

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

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

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

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

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

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

    # pick constant A > 1
    A = 2

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

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

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

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

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

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

listings/DS_generate_QUBO.py

B Python Program to Generate QUBO Formulation of the Edge Cover Problem

```
import sys, math, networkx as nx

def generateQUBO(G):

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

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

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

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

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

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

    # pick constant A > 1
    A = 2
    # sum x_i,j
    for e in G.edges():
        Q[edgeDict[e],edgeDict[e]] = 1

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

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

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

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

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

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

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

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

listings/EC_generate_QUBO.py

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

```python
import sys, networkx as nx

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

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

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

listings/DS_sage.py

D Python Program to Scale the Ising Instances

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

import sys, time, math, traceback

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

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

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

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

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

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

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

# read minor embedding
embedding=eval(sys.stdin.readline())
```

```python
print 'embedding=', embedding
print 'Physical qubits used= %s' % sum(len(embed) for embed in embedding)

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

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

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

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

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

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

listings/scale_Ising.py