Formulating Graph Covering Problems for Adiabatic Quatumn Computers

Michael J. Dinneen
Rong Wang

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

CDMTCS-495
Februarry 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 both these set cover problems is $O(n \lg n)$, where n is the number of vertices, even though the classical complexities differ. 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 424 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 [9, 8]. It has been shown to be equivalent to the more traditional quantum "circuit model" [1]. Other introductory details about the application of adiabatic quantum computing may be found in [18, 4]. 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 [5]). 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 [16] 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. both require $O(n \lg n)$ qubits for graphs of order n). 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 involves 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 [11]. It is also one of the harder NP-complete problems being classified as W[2]-hard when considering parameterized complexity [2]. An extensive history on this problem may be found in [13]. 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 [15].

The paper is organized as follows. In Sections 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 for the Dominating Set problem.

2 QUBO Formulation

QUBO is an NP-hard mathematical optimization problem of minimizing a quadratic objective function $x^* = x^T Q x$, where $x = (x_1, x_2, \ldots, x_n)$ is a n-vector of binary (Boolean) variables and Q is a symmetric $n \times n$ matrix. Formally, QUBO problems are of the form:

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

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 $O(2^{0.610n})$ [10].

Given a graph $G = (V, E)$ with n vertices, let $V = \{v_1, v_2, \ldots, v_n\}$, $\Delta(v)$ 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 variables for each vertex. For the sake of readability, we will label these redundant variables as $y_{i,k}$, where $0 \leq k \leq \lfloor \log(\Delta(v_i))\rfloor$. Thus we have a vector $x = (x_1, x_2, \ldots, x_n, y_{1,0}, \ \ldots, y_{n,\lfloor \lg(\Delta(v_n))\rfloor})$ of named variables.

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

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

where

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

To obtain a solution of the Dominating Set problem, we take $D(x) = \{v_i \mid x_i = 1\}$, a subset of V as the dominating set. In the objective function, $A > 1$ is a real positive constant and the term $\sum_{v_i \in V} x_i$ represents a penalty for the size of the chosen set, and P_i serves as a penalty if a non-dominating set is chosen. If the assignment of the variables is a dominating set, then for each vertex v_i in G, we have $x_i + \sum_{v_j \in N(v_i)} x_j \geq 1$. And therefore $1 - (x_i + \sum_{v_j \in N(v_i)} x_j) \leq 0$. 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 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 which will have a smaller value in the objective function.

Suppose we have $x^* = \min_x F(x)$ and $D(x^*)$ is not a dominating set. Then there must exist some vertices such that these vertices themselves nor any of their neighbors are present in $D(x^*)$. 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 $F(x^*)$ will be reduced by $A - 1$ at least. Hence the solution from $x^* = \min_x F(x)$ 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 smallest value in the objective function. This is trivial as if $D(x)$ 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 $D(x)$. □

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

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

Technically, the objective function of a QUBO problem can only contain quadratic terms, so a few terms in (2) have to be modified. Firstly, the constant term nA where n is the order of the graph, is ignored completely. Removing this constant does not have any impact over the optimal solutions of the QUBO problem. As removing nA will reduce the value of the objective function by nA 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

$$\begin{aligned}(1 - A) \sum_{v_i \in V} x_i^2 + A \sum_{v_i \in V} \Bigg(& -2 \sum_{v_j \in N(v_i)} x_j^2 + 2 \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k}^2 + 2x_i \sum_{v_j \in N(v_i)} x_j \\ & -2x_i \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} + (\sum_{v_j \in N(v_i)} x_j)^2 - 2 \sum_{v_j \in N(v_i)} x_j \sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k} + (\sum_{k=0}^{\lfloor \lg(\Delta(x_i)) \rfloor} 2^k y_{i,k})^2 \Bigg) \end{aligned} \tag{3}$$

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 and entries $Q_{i,j}$ where $i \geq j$ are set to the values of $Q_{j,i}$ in order to obtain a symmetric matrix.

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

After solving for $x^* = \min_x F(x)$, we obtain four optimal solutions.

$$x_1 = [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$x_2 = [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$x_3 = [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

and

$$x_4 = [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

And we have $D(x_1) = \{0, 7\}$, $D(x_2) = \{1, 6\}$, $D(x_3) = \{2, 5\}$ and $D(x_4) = \{3, 4\}$. 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 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 [7, 17]. Given a graph $G = (V, E)$ with n vertices and m edges, let $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{e_{i,j} \mid v_j \in N(v_i)\}$. 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.

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

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

where

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

At the end, we take $C(x) = \{e_{i,j} \mid x_{i,j} = 1\}$ 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 are chosen as the edge cover when $\Delta(v_i) > 1$, and if $\Delta(v_i) = 1$, then no redundant variables are needed at all for vertex v_i as the only edge incident to v_i has to be chosen in the edge cover set so $1 - \sum_{e_{i,j} \in I(i)} x_{i,j}$ has to be 0. The argument that will be used here to show the correctness of this formulation is quite similar to the proof in the previous subsection.

Theorem 2. *The objective function (4) is correct.*

Proof. First, we show that a solution from $x^* = \min_x F(x)$ will always be an edge cover. Suppose we have $x^* = \min_x F(x)$ and $C(x^*)$ is not an edge cover. Then there must exist a set of vertices $\{u_1, u_2, \ldots, u_l\}$ such that $I(u_i) \cap C(x^*) = \emptyset$ for all $1 \leq i \leq l$. That is, for each u_i, none of the edges incident to u_i is in $C(x^*)$. 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 $F(x)$ by at least $A - 1$.

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

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 QUBO 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, constant and linear terms have to be replaced just like in the Dominating Set problem. Doing so would give us

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

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

$$x = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1] \text{ and } C(x) = \{(0,i) \mid 1 \leq i \leq n\}$$

which can be verified quite easily the only minimum edge cover for S_{15}.

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 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.

3.1 Weighted Dominating Set

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

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

where

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

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

Theorem 3. *The QUBO formulation in (6) is correct.*

Proof. First, we show that it is always possible to transform a non-dominating set into a dominating set which will have a smaller value in the objective function. Suppose we have $x^* = \min_x F(x)$ and $D(x^*)$ 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 $D(x^*)$. 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 $F(x^*)$ will be reduced by $A - w_i > 0$ by choice of A. Hence the solution from $x^* = \min_x F(x)$ 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 smallest value in the objective function. It is trivial as if $D(x)$ 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:

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

The encoded QUBO matrix is shown in Table 3. Solving $x^* = \min_x \sum_{i \leq j} x_i Q_{(i,j)} x_j$ this time gives two different solutions with identical value objective function (6). The two solutions are $x_1 = [1,0,0,0,0,0,0,0,0,0,0,0,0,0]$ and $x_2 = [0,1,1,1,1,1,0,0,1,0,0,0,0,0]$.

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

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

The number of vertices in these two dominating sets is different, $D(x_1)$ has only one vertex while $D(x_2)$ has five vertices.

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

3.3 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 $O(n^3)$ [19, 14][1]. Similar to the weighted dominating set formulation in the previous subsection, we only need to do some small modification to the Edge Cover problem to obtain a QUBO formulation for the weighted version.

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

where

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

Once again, we need to have $A > \max\{w_{i,j} \mid \forall e_{i,j} \in E\}$. Although the argument may seem almost identical to the unweighted version, for the sake of completeness, we will present the theorem and proof formally below.

Theorem 4. *The QUBO formulation in (7) is correct.*

Proof. First, we show that a solution from $x^* = \min_x F(x)$ will always be an edge cover. Suppose we have $x^* = \min_x F(x)$ and $C(x^*)$ is not an edge cover. Then there must exist a set of vertices $\{u_1, u_2, \ldots, u_l\}$ such that $I(u_i) \cap C(x^*) = \emptyset$ for $1 \le i \le l$. That is, for each u_i, none of the edges incident to u_i is in $C(x^*)$. 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 $F(x)$ will always be an edge cover set.

Now since $C(x^*)$ where $x^* = \min_x F(x)$ has to be an edge cover as shown in the previous paragraph, it also has to be the smallest edge cover. When $C(x)$ is an edge cover, each P_i

[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).

in $F(x)$ has to be 0 and therefore the value of $F(x)$ is completely dependent on the weights of the edges chosen to be in $C(x)$. $\square$

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 vertices into a cycle of length n. Taking $n=5$, we have $V=\{0,1,2,3,4,5\}$ and $E=\{(0,i) \mid 1 \leq i \leq n\} \cup \{(1,2),(2,3),(3,4),(4,5),(5,1)\}$. The weight function $W: E \to \mathbb{R}^+$ which assigns a weight to each edge is defined as follows:

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

Let $A=20$, the QUBO matrix encoded from objective function (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.

$$x_1 = [0,0,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]$$
$$x_2 = [1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]$$

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.

4 Experimental Results and Discussion

Experiments were conducted on the D-Wave quantum computer. The chip had 424 active qubits and could support problems with up to 424 variables in theory. In practice, before the QUBO formulations could be executed on the D-Wave computer, an embedding on the hardware must be first 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 a (induced subgraph of) specific Chimera host graph. A Python scripts which generates the QUBO instance of objective function for Dominating Set and Edge Cover are available from the authors. We used the NetworkX graph package in both scripts [12] in addition to the D-Wave library.

For the Dominating Set problem, with $A=2$, we did two trials on a set of small graphs. For the second trial, instead of encoding $F(x)$ exactly as in (1), a constant multiplier 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.

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

5 was applied. That is, $5F(x)$ was used instead of $F(x)$. This modification changes the value of the objective function by a factor of 5 but does not affect the optimal solutions of $x^* = \min_x F(x)$. The aim of this modification was to determine if there were any links between the quality of the solutions found by the computer and the energy level set by the objective function. Each QUBO instance corresponding to each graph was ran one hundred times in both trials. However, nothing conclusive could be drawn from the result with respect to the energy level and the solution quality, the solutions obtained from the two experiments were indistinguishable from one another. In general, very small differences were observed on the same graph. Therefore we decided to only list the better result in Table 5 for each graph, that is, each row of the table is taken from the better trial. One trial is considered better if the size of the smallest dominating set was smaller. One interesting thing to note is that all solutions returned by D-Wave are dominating sets, not necessarily good ones in terms of the number of vertices, but they do cover the graph correctly.

The first three columns of Table 5 contains standard information about the graphs; the *order* and *size* are with respect to the input graphs rather than the QUBO formulations. The next three columns contains information on the embeddings. *Logical qubits* is the number of variables of the formulation and *physical qubits* is the number of hardware qubits required after embedding the QUBO instance into the 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 the guest graph 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 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 [5] and [3]. Another thing to note here is that we did not try to minimize the number of variables nor the density of the QUBO matrix when developing the objective functions given in Section 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.

Finally in Table 5, the *best answer* column is the best solution the D-Wave machine was able to find and the *optimal answer* column is the true optimal solution of the particular graph. The optimal solution was computed by first computing an Integer Programming formulation of the Dominating Set problem and then Sage Maths [20] software was used to compute the solution. As mentioned before, each QUBO instance was executed one hundred times, 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 one hundred times. As can be seen in the table, the overall result was not very satisfying. In a lot of the cases, the D-Wave quantum computer was not able to find the true optimal solution. Furthermore, even when it did, the probability of at successfully finding the optimal solution is very low in most cases. We did do software simulations (e.g. conventional evolutionary search) on our QUBO matrices to verify optimal dominating sets were possible.

Google has recently published a paper that had some experimental result on the new D-Wave 2X [6] 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 in such a big difference, the authors did considered several factors that could have lead to this observation. First of all, the size of the problems, as in the number of physical qubits used, we tested here are relatively small compare to [6] which had problems sizes vary from 200 to around 1000. Google had found the best suited annealing time for each case, in terms of solution quality and problem size, while we used an annealing time of 20 microseconds for all of our test cases as we did not have direct access to the machine so it was difficult for us to run multiple fine-tuned tests. Secondly, the problem that Google used in their benchmark paper is a hand crafted problem that fits directly on the actual hardware they have, in other words, the underlying graph is a subgraph of the Chimera graph, so the minor containment problem did not need to be considered. In general, the bigger the mapping size is, the more extra constraints the objective function needs [5] hence it might reduce the solution quality.

The authors plan to experiment with a newer D-Wave machine which has 1100 active qubits and less noise issues. Hopefully bigger problems can now be solved because of a larger host graph and better connectivity (e.g. embeddings with smaller chains). We anticipate the solution quality will improve substantially. We are also in the process of testing our QUBO formulations for the Edge Cover problem and the other weighted covering problems described in this paper.

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 that help us improve this report.

References

[1] 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.

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

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

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

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

[6] 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.

[7] S. Even and O. 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

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

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

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

[17] 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.

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

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

[20] 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 Hamiltonian.

Graph	Order	Size	Logical Qubits	Physical Qubits	Embedding Max Chain	Best Answer	Optimal Answer	% with Best
BidiakisCube	12	18	36	217	13	8	4	1
Bull	5	5	13	38	5	3	2	4
Butterfly	5	6	16	62	8	2	1	1
C4	4	4	12	48	6	2	2	8
C5	5	5	15	50	6	2	2	1
C6	6	6	18	62	6	3	2	9
C7	7	7	21	88	7	5	3	5
C8	8	8	24	90	9	4	3	1
C9	9	9	27	94	7	5	3	4
C10	10	10	30	100	7	6	4	1
C11	11	11	33	125	9	7	4	1
C12	12	12	36	157	11	8	4	2
Diamond	4	5	12	48	6	2	1	26
Dinneen	9	21	36	297	25	6	2	2
Durer	12	18	36	270	22	8	4	1
Frucht	12	18	36	196	14	8	3	1
Grid2x3	6	7	18	88	9	4	2	2
Grid3x3	9	12	28	172	15	6	3	1
Grid3x4	12	17	38	250	16	7	4	2
Grid4x4	16	24	52	309	20	10	4	1
Grotzsch	11	20	39	301	30	8	3	1
Heawood	14	21	42	293	25	9	4	4
Herschel	11	18	36	260	18	8	3	3
Hexahedral	8	12	24	130	12	4	2	7
House	5	6	15	55	6	2	2	2
K2	2	1	4	5	2	1	1	91
K3	3	3	9	23	4	1	1	3
K4	4	6	12	42	5	2	1	10
K5	5	10	20	127	15	4	1	5
K6	6	15	24	199	17	4	1	2
K7	7	21	28	307	25	4	1	1
K8	8	28	32	333	29	4	1	1
K2,3	5	6	15	54	8	2	2	1
K3,3	6	9	18	87	8	3	2	1
K3,4	7	12	24	151	15	5	2	2
K4,4	8	16	32	197	13	6	2	5
K4,5	9	20	36	278	20	7	2	2
K5,5	10	25	40	338	22	7	2	1
Krackhardt	10	18	34	237	22	7	2	1
Octahedral	6	12	24	152	15	4	2	1
Petersen	10	15	30	227	20	7	3	1
Q3	8	12	24	153	10	4	2	2
S2	3	2	7	13	2	1	1	50
S3	4	3	9	22	4	1	1	3
S4	5	4	12	44	6	1	1	1
S5	6	5	14	42	5	3	1	12
S6	7	6	16	50	6	3	1	2
S7	8	7	18	65	6	4	1	5
S8	9	8	21	102	11	4	1	1
S9	10	9	23	108	9	5	1	2
S10	11	10	25	153	13	8	1	1
Tietze	12	18	36	230	16	8	3	3
Wagner	8	12	24	133	9	4	3	1

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

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

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

```python
import networkx as nx
import sys, math
from dwave_sapi import solve_qubo, local_connection
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 = {}

    # map each edge of G to some index in 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

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

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

    # main program
    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

```
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