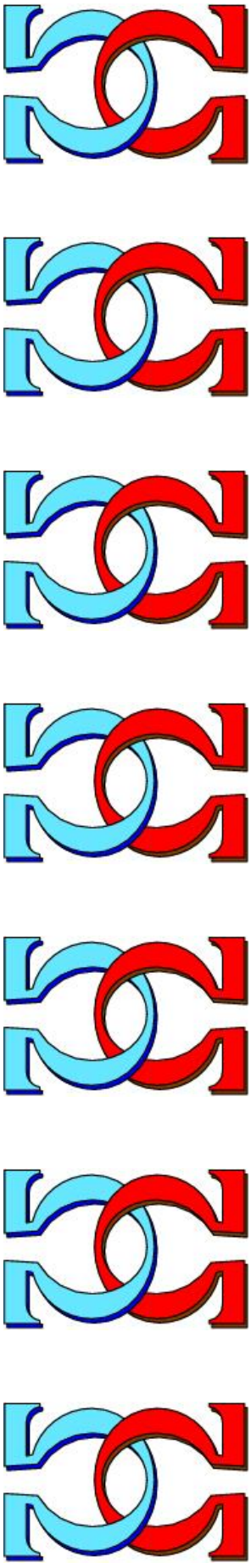

CDMTCS
Research
Report
Series

Formulating Mixed Dominating Set Problems for Adiabatic Quantum Computers

Michael J. Dinneen
Department of Computer Science,
University of Auckland, Auckland, New Zealand

Mohammad Reza Hooshmandasl
Department of Computer Science,
Yazd University, Yazd, Iran

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

CDMTCS-507
May 2017

Centre for Discrete Mathematics and
Theoretical Computer Science

Formulating Mixed Dominating Set Problems for Adiabatic Quantum Computers

Michael J. Dinneen[1], *Mohammad Reza Hooshmandasl*[2], *Richard Hua*[3]

[1,3]Department of Computer Science, The University of Auckland, Auckland, New Zealand.

[2]Department of Computer Science, Yazd University, Yazd, Iran.

e-mail:[1]mjd@cs.auckland.ac.nz, [2]hooshmandasl@yazd.ac.ir, [3]rwan074@aucklanduni.ac.nz.

Abstract

In this paper, we present efficient quadratic unconstrained binary optimization (QUBO) formulations for the mixed dominating set and the weighted mixed dominating set problems, which are both NP-hard. By using a D-Wave 2X quantum computer with 1098 active qubits, the QUBO formulation of the unweighted mixed dominating set is tested on several small graphs. In every test graph, 30K samples were taken on the D-Wave computer in two different execution modes (with and without post-processing optimization). The experimental results achieved optimal answers in the majority of the cases. The correctness of the formulations are proven, establishing empirical evidence that our formulation and their implementations are correct.

Keywords: Adiabatic quantum computing; Quadratic Unconstrained Binary Optimization; Ising/QUBO formulation; Mixed dominating set; Weighted mixed dominating set.

1 Introduction

Recently, adiabatic quantum computation (AQC) has attracted attention in the computing community. With respect to the standard quantum circuit model of quantum computation, the AQC model is equivalent with polynomial overhead [1, 15]. AQC is based on the adiabatic principle to approximate solutions of the Schrödinger's equation [6].

One of the pioneers in producing quantum computing computers is the company D-Wave, which has a general-purpose Ising (or QUBO, defined later) problem-solving hardware. These formulated problems correspond to finding the minimum energy state of a system modeled by Schrödinger's equation.

Since the Ising problem is an NP-hard problem [5], the standard way to solve a hard combinatorial optimization problem with D-Wave machine is to find an equivalent Ising/QUBO formulation (with polynomial-time reduction).

A set of vertices D in a graph $G = (V, E)$ is called a dominating set of G if every vertex in $V - D$ is adjacent to at lest one vertex in D. The dominating set problem is to find a dominating set with minimum cardinality. For general graphs, the decision version of the dominating set problem is NP-complete. Depending on applications, variations of the problem have been defined. One such problem is the *mixed dominating set* problem, also know as the total cover problem, it was introduced by Alavi et. al in 1977 [3]. For a given simple graph $G = (V, E)$, a vertex v of G is said to mixed dominates itself, all edges incident to v and all vertices adjacent to v. Similarly, an edge e of G is said to mixed dominates itself, the two end vertices of e, and all edges adjacent to e. A set $D \subseteq V \cup E$ is called a *mixed dominating set* if each element of $V \cup E$ is mixed dominated by some element of D. The *mixed domination number* of G, denoted by $\gamma_m(G)$, is the minimum size of a mixed dominating set in G. One of the known applications of Mixed Dominating Set Problem is the placement of phase measurement units (PMUs) in electrical power systems [17].

The Mixed Dominating Set Problem is NP-hard for general graphs [11] and remains NP-hard when instances are restricted to chordal graphs [9], planar bipartite graphs of maximum degree 4 [12] and split graphs [17, 10]. Finding a mixed dominating set with minimum cardinality is tractable for some family of graphs such as trees [11, 8] and cactus graphs [10]. Recently, a dynamic programming algorithm is proposed to solve the Mixed Dominating problem for graphs with bounded tree-width tw in $O(3^{tw^2} \times tw^2 \times |V|)$ time [13].

The focus of this paper is to use the mathematical QUBO model to solve the mixed Dominating Set Problems and its weighted version. The extension of the Mixed Dominating Set Problem to the Weighted Mixed Dominating Set Problem is proposed (for the first time). Since the problem of finding the optimal locations of phasor measurement units (PMUs) in a given power system is considered with weights [16, 4, 2], therefore the proposed method can be used in analysis for such systems. The rest of paper is organized as follows. In Section 2, we review some basic definition and notions. In Section 3, we present efficient QUBO formulations, along with proof of correctness, of the Mixed Dominating Set Problem and the Weighted Mixed Dominating Set Problem. The last section concludes with our experimental results.

2 Preliminaries

In this section, we introduce the necessary background knowledge and notations for what is to follow.

The cardinality of a set X is denoted by $|X|$. By lg we denote the logarithm function in base 2.

A graph $G = (V, E)$ consists a finite non-empty set of vertices V together with a set of edges E. The order of G, denoted by n, is the number of vertices. The vertices are labelled by $V = \{v_i \mid 0 \leq i < n\}$. The E consists of unordered pairs of vertices $u, v \in V$. We denote an edge by $e = uv$ or $e = \{u, v\}$, since uv and vu are considered the same edge, it will always be referred to with $u < v$. The number of edges, denoted by m, is called the size of G. Two vertices u and v are said to be adjacent (neighbors) if there is an edge $uv \in E$. The number

of neighbors a vertex v, denoted by $\Delta(v)$, is called the degree of v. Furthermore, if u and v are two vertices in V and uv is an edge in E, then we say that the edge uv is incident to the vertices u and v. Two edges uv and $u'v'$ are said to be edge-incident if they share a common vertex. The set of neighbors of a vertex v will be denoted by $N(v)$, and the set of edges incident to v will be denoted by $I(v)$. The set of edge-incident edges of an edge e will also be denoted by $I(e)$. For notational convenience, we define the following mixed-neighborhood function:

$$N^{md}(x) : V \cup E \to V \cup E$$

and

$$N^{md}(x) = \begin{cases} N(x) \cup I(x), & \text{if } x \in V \\ I(x) \cup \{u, v\}, & \text{if } x = uv \in E \end{cases}$$

Given a graph $G = (V, E)$, a mixed dominating set of G is a subset $MD \subseteq (V \cup E)$ such that for all $x \in V \cup E$, either $x \in MD$ or $MD \cap N^{md}(x) \neq \emptyset$. The Dominating Set Problem formally defined below involves finding the smallest of such set.

Mixed Dominating Set Problem:

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

3 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:

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

3.1 Mixed Dominating Set

We provide a simple QUBO formulation of the Mixed Dominating Set problem that is inspired by [7]. Given a graph $G = (V, E)$ with n vertices, let $V = \{v_0, v_1, \ldots, v_{n-1}\}$. We use one binary variable for each vertex, and one binary variable for each edge in the graph. Variables for both sets will be denoted by $x_{i,j}$, we follow the convention that $x_{i,j}$ corresponds to vertex v_i if $i = j$. Otherwise it corresponds to the edge ij if $i \neq j$. Since there is a natural correspondence between the binary variables $x_{i,j}$ and elements in the set $V \cup E$, to improve the readability of the objective function, we will use them interchangeably in what follows.

The objective function to be minimized is of the form:

$$F(\mathbf{x}) = \sum_{v_i \in V} x_{i,i} + \sum_{ij \in E} x_{i,j} + A \sum_{x_{i,j} \in (V \cup E)} P_{i,j} \tag{1}$$

where

$$P_{i,j} = \left(1 - \left(x_{i,j} + \sum_{x_{u,v} \in N^{md}(x_{i,j})} x_{u,v}\right) + \sum_{k=0}^{\lfloor \lg(|N^{md}(x_{i,j})|) \rfloor} 2^k y_{i,j,k}\right)^2$$

Note that in objective function (1), we have also introduced $\lfloor \lg(|N^{md}(x_{i,j})|) \rfloor + 1$ redundant variables for each variable $x_{i,j}$. These variables are necessary for counter balance unnecessary penalties when more than one element in the mixed-neighborhood of a vertex or edge is chosen to be in the mixed dominating set.

Assume $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ and $\mathbf{x}^*$ is the corresponding variable assignment. To obtain a solution of the Mixed Dominating Set Problem, we use an additional decoder function $D(\mathbf{x}) : \mathbb{Z}_2^{|\mathbf{x}|} \to 2^{V \cup E}$ and take $D(\mathbf{x}) = \{v_i \mid x_{i,i} = 1\} \cup \{e_{i,j} \mid x_{i,j} = 1\}$, a subset of $V \cup E$ as the mixed dominating set.

In the objective function, $A > 1$ is a positive real constant, the sum of $\sum_{v_i \in V} x_{i,i}$ and $\sum_{ij \in E} x_{i,j}$ represents a penalty for the size (number of elements) of the chosen set, and $P_{i,j}$ serves as a penalty if an incorrect set is chosen. If the assignment of the variables is a mixed dominating set, then for each $x_{i,j}$, we have $x_{i,j} + \sum_{x_{u,v} \in N^{md}(x_{i,j})} x_{u,v} \geq 1$. And therefore $1 - (x_{i,j} + \sum_{x_{u,v} \in N^{md}(x_{i,j})} x_{u,v}) \leq 0$. In the worst case, $1 - (x_{i,j} + \sum_{x_{u,v} \in N^{md}(x_{i,j})} x_{u,v}) = -|N^{md}(x_{i,j})|$ where the element $x_{i,j}$ corresponds to and all the other elements in its mixed-neighborhood are chosen, so a total number of $\lfloor \lg(|N^{md}(x_{i,j})|) \rfloor + 1$ redundant variables are needed to represent integers up to $|N^{md}(x_{i,j})|$.

Theorem 1. *Let x^* and $\mathbf{x}^*$ be the optimal value and its corresponding variable assignment of objective function (1). Then $D(\mathbf{x}^*)$ is a minimum mixed dominating set of G.*

Proof. First, we show that it is always possible to transform a variable assignment that does not map to a mixed dominating set under the decoder function D into an assignment, which does with a smaller value in objective function (1).

Suppose we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ and $D(\mathbf{x}^*)$ is not a mixed dominating set where $\mathbf{x}^*$ corresponds to the variable assignment yielding x^*. Then there must exist some elements such that these elements themselves nor any elements in their mixed-neighborhood are in $D(\mathbf{x}^*)$. Then the corresponding penalty $P_{i,j}$ for each of these elements will be 1 by the definition of $P_{i,j}$. Therefore, if we set the corresponding $x_{i,j}$ of these elements to 1, then for each one of them, a penalty of size 1 will be added to the term $\sum_{v_i \in V} x_{i,i}$ if the element is a vertex, or a penalty of size 1 will be added to the term $\sum_{ij \in E} x_{i,j}$ if the element is an edge. Furthermore, the corresponding $P_{i,j}$ 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 mixed 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 smaller value in the objective function. This is trivial as if $D(\mathbf{x})$ is a mixed dominating set, then each $P_{i,j}$ will have to be 0, so the value of the objective function solely depends on the sum of $\sum_{v_i \in V} x_{i,i}$ and $\sum_{ij \in E} x_{i,j}$. By the definition of the decoder function $D(\mathbf{x})$, an assignment with a smaller sum will be mapped to a smaller mixed dominating set. □

3.2 Mixed Dominating Set C_3 Example

In this subsection, we will provide an example of the QUBO formulation (1) on the graph C_3. Formally, The cycle graph C_3 is defined as follows. The vertices of C_3 are $V = \{0, 1, 2\}$ and the edges are $E = \{\{0, 1\}, \{0, 2\}, \{1, 2\}\}$. It can be visualized as a 2-dimensional triangle where the each corner of the square is a vertex.

Similar to what was presented in [7], objective function (1) will be modified as follows. Constant terms are ignored, and all linear terms are replaced by the square of that term, that is, we will replace all $x_{i,j}$ and $y_{i,j,k}$ by $x^2_{i,j}$ and $y^2_{i,j,k}$ respectively. These changes will not impact the optimality of any optimal variable assignments of the objective function (see [7] for more details).

After applying the two steps described in the paragraph above and summing up similar terms, with $A = 2$, we compute the coefficients of the each quadratic term in the new expression and present its matrix form in Table 1.

Table 1: Mixed Dominating Set QUBO matrix for C_3

variables	$x_{0,0}$	$x_{1,1}$	$x_{2,2}$	$x_{0,1}$	$x_{0,2}$	$x_{1,2}$	$y_{0,0,0}$	$y_{0,0,1}$	$y_{0,0,2}$	$y_{1,1,0}$	$y_{1,1,1}$	$y_{1,1,2}$	$y_{2,2,0}$	$y_{2,2,1}$	$y_{2,2,2}$	$y_{0,1,0}$	$y_{0,1,1}$	$y_{0,1,2}$	$y_{0,2,0}$	$y_{0,2,1}$	$y_{0,2,2}$	$y_{1,2,0}$	$y_{1,2,1}$	$y_{1,2,2}$
$x_{0,0}$	**-9**	16	16	16	16	16	-4	-8	-16	-4	-8	-16	-4	-8	-16	-4	-8	-16	-4	-8	-16	0	0	0
$x_{1,1}$		**-9**	16	16	16	16	-4	-8	-16	-4	-8	-16	-4	-8	-16	-4	-8	-16	0	0	0	-4	-8	-16
$x_{2,2}$			**-9**	16	16	16	-4	-8	-16	-4	-8	-16	-4	-8	-16	0	0	0	-4	-8	-16	-4	-8	-16
$x_{0,1}$				**-9**	16	16	-4	-8	-16	-4	-8	-16	0	0	0	-4	-8	-16	-4	-8	-16	-4	-8	-16
$x_{0,2}$					**-9**	16	-4	-8	-16	0	0	0	-4	-8	-16	-4	-8	-16	-4	-8	-16	-4	-8	-16
$x_{1,2}$						**-9**	0	0	0	-4	-8	-16	-4	-8	-16	-4	-8	-16	-4	-8	-16	-4	-8	-16
$y_{0,0,0}$							**6**	8	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_{0,0,1}$								**16**	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_{0,0,2}$									**48**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_{1,1,0}$										**6**	8	16	0	0	0	0	0	0	0	0	0	0	0	0
$y_{1,1,1}$											**16**	32	0	0	0	0	0	0	0	0	0	0	0	0
$y_{1,1,2}$												**48**	0	0	0	0	0	0	0	0	0	0	0	0
$y_{2,2,0}$													**6**	8	16	0	0	0	0	0	0	0	0	0
$y_{2,2,1}$														**16**	32	0	0	0	0	0	0	0	0	0
$y_{2,2,2}$															**48**	0	0	0	0	0	0	0	0	0
$y_{0,1,0}$																**6**	8	16	0	0	0	0	0	0
$y_{0,1,1}$																	**16**	32	0	0	0	0	0	0
$y_{0,1,2}$																		**48**	0	0	0	0	0	0
$y_{0,2,0}$																			**6**	8	16	0	0	0
$y_{0,2,1}$																				**16**	32	0	0	0
$y_{0,2,2}$																					**48**	0	0	0
$y_{1,2,0}$																						**6**	8	16
$y_{1,2,1}$																							**16**	32
$y_{1,2,2}$																								**48**

It is fairly easy to verify that any two vertices/edges from $V \cup E$ is a minimum mixed dominating set for C_3 so we have a total of $\binom{6}{2} = 15$ optimal solutions. For instance, we have

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

as one of the optimal solutions and $D(\mathbf{x}) = \{0, 1\}$. Note that the variables $y_{2,2,0}$ and $y_{0,1,0}$ are also set to 1 to counter balance the fact that vertex 2 and edge $\{0, 1\}$ are being covered by two elements.

3.3 Weighted Mixed Dominating Set

The formulation, which was provided in the previous section, can be modified quite easily to adapt to weighted graphs. The definition of the Mixed Dominating Set Problem is slightly different in weighted graphs. For the weighted problem, each element in $V \cup E$ is assigned a real positive weight $w_{i,j}$. Once again, we follow the convention that $w_{i,j}$ is the weight assigned to vertex v_i if $i = j$, otherwise, it is the weight assigned to the edge ij if $i \neq j$. The goal is to find a mixed dominating set that has a minimum sum of the weights. Formally, we have the following definitions.

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

We restrict to positive weights in the problem defined above. Any elements associated with a non-positive weight would always be added to a minimum solution and we could reduce to a strictly positive subproblem.

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

$$F(\mathbf{x}) = \sum_{v_i \in V} w_{i,i} x_{i,i} + \sum_{ij \in E} w_{i,j} x_{i,j} + A \sum_{x_{i,j} \in (V \cup E)} P_{i,j} \tag{2}$$

where

$$P_{i,j} = \left(1 - \left(x_{i,j} + \sum_{x_{u,v} \in N^{md}(x_{i,j})} x_{u,v} \right) + \sum_{k=0}^{\lfloor \lg(|N^{md}(x_{i,j})|) \rfloor} 2^k y_{i,j,k} \right)^2$$

Every term serves the same purpose here except that A has to be picked with the property that $A > \max\{w_{i,j} \mid x_{i,j} \in V \cup E\}$. Finally, we take $D(\mathbf{x}) = \{v_i \mid x_{i,i} = 1\} \cup \{e_{i,j} \mid x_{i,j} = 1\}$ 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 2. *Let x^* and $\mathbf{x}^*$ be the optimal value and its corresponding variable assignment of objective function (2). Then $D(\mathbf{x}^*)$ is a minimum weighted mixed dominating set of G.*

Proof. First, we show that it is always possible to transform a variable assignment that does not map to a mixed dominating set under the decoder function D into an assignment which does with a smaller value in objective function (2).

Suppose we have $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ and $D(\mathbf{x}^*)$ is not a mixed dominating set where $\mathbf{x}^*$ corresponds to the variable assignment yielding x^*. Then there must exist some elements such that these elements themselves nor any elements in their mixed-neighborhood are in $D(\mathbf{x}^*)$. Then the corresponding penalty $P_{i,j}$ for each of these elements will be 1 by the definition of $P_{i,j}$. Therefore, if we set the corresponding $x_{i,j}$ of these elements to 1, then for each one of them, a penalty of size $w_{i,i}$ will be added to the term $\sum_{v_i \in V} x_{i,i}$ if the element is a

vertex, or a penalty of size $w_{i,j}$ will be added to the term $\sum_{ij \in E} x_{i,j}$ if the element is an edge. Furthermore, the corresponding $P_{i,j}$ will be reduced to 0 and so $F(\mathbf{x}^*)$ will be reduced by at least $A - w_{i,j}$. Hence the solution from $x^* = \min_{\mathbf{x}} F(\mathbf{x})$ will always be a mixed dominating set.

The second part of the proof is to show that an assignment of $\mathbf{x}$ that produces a smaller weighted dominating set will have a smaller value in the objective function. This is trivial as if $D(\mathbf{x})$ is a mixed dominating set, then each $P_{i,j}$ will have to be 0, so the value of the objective function solely depends on the sum of $\sum_{v_i \in V} w_{i,i} x_{i,i}$ and $\sum_{ij \in E} w_{i,j} x_{i,j}$. By the definition of the decoder function $D(\mathbf{x})$ and the weighted sum function $S(MD)$, an assignment with a smaller value in objective function (2) will be mapped to a mixed dominating set with a smaller weighted sum. □

4 Computation Results

We tested our unweighted mixed dominating set QUBO formulation for several small graphs on a D-Wave 2X computer with 1098 active qubits. See [7] for more information about this computer. The results are highlighted in Table 2 using the programs listed in Appendices A–D. We took 30K samples for each test case in two different modes, with and without post-processing optimization. The later mode does a little bit of deterministic search to find local optimal answers, starting at the sample state. Only slightly better answers were obtained for the second mode, which is reflected in the majority of the "Minimum D-Wave" column of the table. The number of *logical qubits* represents the size of our QUBO matrices and the *physical qubits* represents the total size, after doing a minor embedding of the QUBO graph structure onto the physical D-Wave machine. The *maximum chain size* indicates how good the embedding is with a lower number generally better. The last column of the table is the mixed dominating number (optimal answer) computed by an Integer Program solver using Sage mathematical system [14] (see Appendix E). As can be observed in the table, we obtained the expected optimal answer from the D-Wave computer in most instances.

The total running time using default API parameters for about 10K samples on the D-Wave is about 3770000μs, with the bulk of the time about 3700000μs reading samples. Only about 20000μs was used for programming/loading the QUBO problem and about 117000μs and 404000μs for postprocessing (non-optimized and optimized, respectively). The total wall/real-time, which includes substantial network connection and queue time, for taking all 60K samples was about 45 minutes. For comparison, our sage exact solver could find all graphs' optimal answer was about 30 seconds.

Acknowledgement

This article has been written while the second author was in a sabbatical visit to the University of Auckland. He would like to express his gratitude to Prof. Cristian S. Calude and Department of Computer Science at the University of Auckland for the nice and friendly hospitality.

Table 2: Results for some small graphs families for Mixed Dominating Set.

Graph	Order	Size	Logical Qubits	Physical Qubits	Maximum Chain Size	Minimum PostProc Default	Minimum PostProc Optimize	Minimum D-Wave	Optimal Answer
Bull	5	5	38	264	17	2	2	2	2
Butterfly	5	6	45	390	20	3	3	3	3
C4	4	4	32	179	12	2	2	2	2
C5	5	5	40	240	14	2	3	2	2
C6	6	6	48	352	20	3	3	3	3
C7	7	7	56	329	14	3	4	3	3
C8	8	8	64	374	14	5	4	4	4
C9	9	9	72	546	25	5	6	5	4
C10	10	10	80	478	15	7	7	7	4
C11	11	11	88	541	16	6	7	6	5
C12	12	12	96	602	17	8	7	7	5
Diamond	4	5	36	332	19	2	2	2	2
Grid2x3	6	7	52	485	26	3	3	3	3
Grid3x3	9	12	85	871	36	5	5	5	4
Hexahedral	8	12	80	925	37	4	4	4	4
House	5	6	44	387	23	2	2	2	2
K2	2	1	9	28	6	1	1	1	1
K3	3	3	24	119	12	2	2	2	2
K4	4	6	40	316	17	2	2	2	2
K2x3	5	6	44	353	18	2	2	2	2
K3x3	6	9	60	669	30	3	3	3	3
S2	3	2	16	59	8	1	1	1	1
S3	4	3	25	135	10	1	1	1	1
S4	5	4	33	203	12	1	1	1	1
S5	6	5	40	337	18	1	1	1	1
S6	7	6	47	468	20	2	2	2	1
S7	8	7	61	790	40	2	2	2	1
S8	9	8	70	879	45	2	1	1	1

References

[1] D. Aharonov, W. Van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM Journal on Computing*, 37(1):166–194, 2007.

[2] A. Ahmadi, Y. Alinejad-Beromi, and M. Moradi. Optimal pmu placement for power system observability using binary particle swarm optimization and considering measurement redundancy. *Expert Systems with Applications*, 38(6):7263–7269, 2011.

[3] Y. Alavi, M. Behzad, L. M. Lesniak-Foster, and E. Nordhaus. Total matchings and total coverings of graphs. *Journal of Graph Theory*, 1(2):135–140, 1977.

[4] S. Chakrabarti, E. Kyriakides, and D. G. Eliades. Placement of synchronized measurements for power system observability. *IEEE Transactions on Power Delivery*, 24(1):12–19, 2009.

[5] B. A. Cipra. The ising model is NP-complete. *SIAM News*, 33(6):1–3, 2000.

[6] W. Cruz-Santos and G. Morales-Luna. *Approximability of Optimization Problems through Adiabatic Quantum Computation*. Synthesis Lectures on Quantum Computing. Morgan & Claypool, 2014.

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

[8] P. Hatami. An approximation algorithm for the total covering problem. *Discussiones Mathematicae Graph Theory*, 27(3):553–558, 2007.

[9] S. M. Hedetniemi, S. T. Hedetniemi, R. Laskar, A. McRae, and A. Majumdar. Domination, independence and irredundance in total graphs: a brief survey. In *Graph Theory, Combinatorics and Applications: Proceedings of the 7th Quadrennial International Conference on the Theory and Applications of Graphs*, volume 2, pages 671–683, 1995.

[10] J. K. Lan and G. J. Chang. On the mixed domination problem in graphs. *Theoretical Computer Science*, 476:84–93, 2013.

[11] A. Majumdar. *Neighborhood hypergraphs: a framework for covering and packing parameters in a graph*. PhD thesis, Department of Mathematical Sciences, Clemson University, South Carolina, 1992.

[12] D. F. Manlove. On the algorithmic complexity of twelve covering and independence parameters of graphs. *Discrete Applied Mathematics*, 91(1-3):155–175, 1999.

[13] M. Rajaati, M. Hooshmandasl, M. Dinneen, and A. Shakiba. On fixed-parameter tractability of the mixed domination problem for graphs with bounded tree-width. *arXiv preprint arXiv:1612.08234*, 2016.

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

[15] W. Van Dam, M. Mosca, and U. Vazirani. How powerful is adiabatic quantum computation? In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 279–287. IEEE, 2001.

[16] B. Xu and A. Abur. Observability analysis and measurement placement for systems with pmus. In *Power Systems Conference and Exposition, 2004. IEEE PES*, pages 943–946. IEEE, 2004.

[17] Y. Zhao, L. Kang, and M. Y. Sohn. The algorithmic complexity of mixed domination in graphs. *Theoretical Computer Science*, 412(22):2387–2392, 2011.

A Python utilities for mixed domination number

```
#!/usr/bin/python3
#
# utilites for MDS scripts

import sys
import networkx as nx

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

# output graph in adjacency list format
#
def print_graph(G):
    n=G.order()
    print(n)
    for u in range(n):
        for v in G[u]: print(v,end=' ')
        print()

# return mixed neighborhood for vertices (u,u) or edges (u,v)
#
def mixed_neighborhood( G, x ):
    u,v=x[0],x[1]

    result = []
    if u == v:
        for i in G.neighbors(u):
            result.append((i,i))
        for edge in G.edges(u):
            temp = (min(edge), max(edge))
            result.append(temp)
    else:
        result.append((u,u))
        result.append((v,v))
        u_edges = G.edges(u)
```

```
        v_edges = G.edges(v)
        for edge in u_edges:
            temp = (min(edge), max(edge))
            if temp != (u,v):
                result.append(temp)
        for edge in v_edges:
            temp = (min(edge), max(edge))
            if temp != (u,v):
                result.append(temp)

    return result
```

listings/MDS_utilities.py

B Python program to generate QUBO

```
#!/usr/bin/python3

import networkx as nx
import sys, math
from MDS_utilities import *

def compute_pij(G, Q, x, var_index, red_var_index, red_var_num)
    :
    v = 0
    for (i,j) in var_index:
        index = var_index[i,j]
        mixed_neigh = mixed_neighborhood(G, (i,j))
        v = 1 - x[index]
        for (a,b) in mixed_neigh:
            temp = var_index[(a,b)]
            v -= x[temp]
        red_index = red_var_index[i,j]
        red_num = red_var_num[i,j]
        for k in range(red_num):
            v += (2**k * x[red_index+k])
        print('P', (i,j), '=', v)

def generateQUBO(G):
    Q = {}
    size = G.size()
    order = G.order()
    total_num_vars = order + size
```

```
# compute/index num of slack vars for vertices and edges
var_index = {}
red_var_index = {}
num_red_var = {}
i = 0
j = 0
for v in G.nodes():
    var_index[v,v] = i
    i += 1
    num = int(math.log(2*nx.degree(G,v),2))+1
    num_red_var[v,v] = num
    red_var_index[v,v] = j + total_num_vars
    j += num

for (u,v) in G.edges():
    var_index[u,v] = i
    i += 1
    md_neigh = nx.degree(G,u) + nx.degree(G,v)

    num = int(math.log(md_neigh,2))+1
    num_red_var[u,v] = num
    red_var_index[u,v] = j + total_num_vars
    j += num

total_num_vars += j

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

for (u,v) in var_index:
    i = var_index[u,v]
    Q[i,i] += 1

A = 2

# encode P_i,j
for (u,v) in var_index:
    # -x_i,j
    i = var_index[(u,v)]
    Q[i,i] -= (1 * A)

    red_index = red_var_index[u,v]
    red_num = num_red_var[u,v]
```

```
mixed_neigh = mixed_neighborhood(G, (u,v))

# -2 sum x_u,v
for (a,b) in mixed_neigh:
    i = var_index[(a,b)]
    Q[i,i] -= (2 * A)

# + 2 * sum 2^k * y_i,j,k
for k in range(red_num):
    i = k + red_index
    Q[i, i] += (2**(k+1) * A)

# +2 x_i,j * sum x_u,v
for (a,b) in mixed_neigh:
    i = var_index[u,v]
    j = var_index[a,b]
    Q[i,j] += (2 * A)

# -2 x_i,j * sum 2^k * y_i,j,k
for k in range(red_num):
    i = k + red_index
    j = var_index[u,v]
    Q[i,j] -= (2**(k+1) * A)

# sum x_u,v * sum x_u,v
for (a,b) in mixed_neigh:
    i = var_index[(a,b)]
    for (c,d) in mixed_neigh:
        j = var_index[(c,d)]
        Q[i,j] += (1 * A)

# -2 sum x_u,v * sum 2^k * y_i,j,k
for (a,b) in mixed_neigh:
    j = var_index[(a,b)]
    for k in range(red_num):
        i = k + red_index
        Q[i,j] -= (2**(k+1) * A)

# sum 2^k * y_i,j,k * sum 2^k * y_i,j,k
for k1 in range(red_num):
    i = k1 + red_index
    for k2 in range(red_num):
        j = k2 + red_index
        Q[i,j] += (2**(k1+k2) * A)
```

```
    # making Q upper-triangular
    for i in range(total_num_vars):
        for j in range(i+1, total_num_vars):
            Q[i,j] += Q[j,i]
            Q[j,i] = 0

    return Q,total_num_vars

## main program ##

G=read_graph(sys.stdin)

Q,nvars=generateQUBO(G)

print(nvars, G.order(), G.size())
for i in range(nvars):
    for j in range(nvars):
        print(Q[i,j],end=' ')
    print()
```

listings/MDS_generate_QUBO.py

C Python program to solve QUBO on D-Wave

```
#!/usr/bin/env python2
# MDS of graph QUBO (with embedding) -> 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

from sys import exc_info

# coupler streingth for embedded qubits of same variable
s,s2=1.0,1.0
print 'Embed scale=',s,s2

assert len(sys.argv)==1
```

```
# read input

line=sys.stdin.readline().strip().split()
n=int(line[0])
order=int(line[1])
size= int(line[2])
print'Logical qubits used=', n, 'order=', order, 'size=', size

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

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

# create a remote connection and connect to solver
#
url = "https://dwave.machine.com"
token = "mytoken"
solver_name = "DW2X"

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

A = get_hardware_adjacency(solver)

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

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

```
for i in range(n):
    if len(H)>i:
        H[i]=H[i]/maxV
    for j in range(n):
        if j>=i and (i,j) in J:
            J[(i,j)]=J[(i,j)]/maxV

# Embed problem into hardware
(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)

# call the solver

result = solve_ising(solver, h1, j1, num_reads=10000,
   postprocess='optimization')
print 'result:', result

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

for i, sol in enumerate(newresult):
  print "solution", i, 'size=', \
     sum(x==1 for x in [sol[j] for j in range(order+size)]), \
     [(1+sol[j])/2 for j in range(n)] # Boolean/QUBO variables
```

listings/MDS_run_dwave.py

D Python program to verify solutions from D-Wave

```
#!/usr/bin/python3
# template for checking dwave answers
# usage: checkMDS.py graph.alist < graph.d.out

import sys
import networkx as nx
from MDS_utilities import *
```

```
def testsol(X):
    ans=sum(X[:n+m])
    MV=set((v,v) for v in range(n) if X[v]==1)
    ME=set(E[e-n] for e in range(n,n+m) if X[e]==1)
    MDS=MV | ME

    status=True

    # test vertices are covered
    for v in range(n):
        if (v,v) in MV: continue
        if set(mixed_neighborhood( G, (v,v) )) & MDS: continue
        status=False
        break

    # test edges are covered
    for (u,v) in G.edges():
        assert u < v
        if (u,v) in ME: continue
        if set(mixed_neighborhood( G, (u,v) )) & MDS: continue
        status=False
        break

    if status==True: return ans
    else:            return -1


### main check program ###

assert len(sys.argv)==2

gfile=open(sys.argv[1],'r')
G=read_graph(gfile)

n=G.order()
m=G.size()
E=G.edges()

minsol=999
mincnt=0

for line in sys.stdin:

    if line.find('solution ')==0:
       X=eval(line[line.find('['):])
```

```
        s=float(line[line.find('size=')+5:line.find('[')-1])
        if s>minsol: continue

        ans=testsol(X)
        if ans<0: continue

        if ans < minsol:
           minsol=ans
           mincnt=0
           print('new best:',minsol)
        if ans == minsol:
           mincnt += 1
           print(X)

print(minsol,mincnt)
```

listings/MDS_check.py

E Sage program for mixed domination number

```
#!/usr/bin/env sage

import sys, networkx as nx

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

G=read_graph()
n=G.order()
#print 'Graph', [(u,v) for (u,v) in G.edges()]

p=MixedIntegerLinearProgram(maximization=False, solver="GLPK")
x=p.new_variable(binary=True)
#p.set_binary(x)

var_index = {}
counter = 0
for v in G.nodes():
```

```
    var_index[v] = counter
    counter += 1
for (u, v) in G.edges():
    var_index[u,v] = counter
    var_index[v,u] = counter
    counter += 1

for u in G.nodes():
    p.add_constraint(x[var_index[u]]
      + sum(x[var_index[v]] for v in G[u])
      + sum(x[var_index[e]] for e in G.edges(u)), min = 1)

for (u,v) in G.edges():
    p.add_constraint(x[var_index[u]]+x[var_index[v]]
      + sum(x[var_index[e]] for e in G.edges(u))
      + sum(x[var_index[e]] for e in G.edges(v)), min = 1)

p.set_objective(sum(x[var_index[v]] for v in G.nodes())
   + sum(x[var_index[e]] for e in G.edges() ))

try:
    sz=p.solve()

except sage.numerical.mip.MIPSolverException as e:
    pass
else:
    print "Mixed domination number is", sz
```

listings/MDS_IP.sage