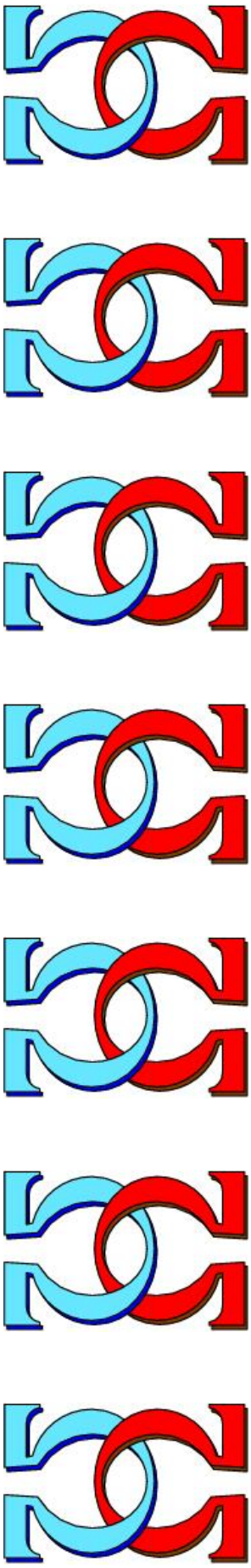


CDMTCS
Research
Report
Series

Solving the Bounded-Depth Steiner Tree Problem using an Adiabatic Quantum Computer

Kai Liu
Michael J. Dinneen

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

CDMTCS-532
Feb 2019 (updated Oct 2019)

Centre for Discrete Mathematics and
Theoretical Computer Science

Solving the Bounded-Depth Steiner Tree Problem using an Adiabatic Quantum Computer

Kai Liu and Michael J. Dinneen

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

Abstract. In this paper we propose quadratic unconstrained binary optimization (QUBO) formulation for the Bounded-Depth Steiner Tree problem suitable for adiabatic quantum computers. The numbers of qubits (dimension of QUBO matrices) required by our formulation is $\mathcal{O}(|V|^3)$, where $|V|$ represents the number of vertices. A D-Wave 2X computer consisting of 1098 active qubits is evaluated for a selection of known common graphs. Experimental results are shown and discussed. Our results are applicable to the Bounded-Depth Minimum Spanning Tree problem, as a special case.

Keywords: bounded-depth; Steiner tree; hop-constrained; QUBO formulation; adiabatic quantum computing

1 Introduction

In the last seven years, D-Wave Systems developed the first comercial adiabatic quantum computers from a 128-qubit model (announced in 2011) to a model operating on 2048 qubits (early 2017). As a comparison, Intels first 8-byte SRAM chip was released in 1969, and in 1976 the Intel 8048 was released which has a 64-byte internal (on-chip) RAM. While rapid development are observed in both cases, the Moore's law, which has been followed by classical computers for over fifty years, cannot continue indefinitely. It is then of great interest to explore the potential capabilities of the new technology.

A D-Wave adiabatic quantum computer can naturally solve either Ising or Quadratic Unconstrained Binary Optimization (QUBO) problems. The QUBO framework is a unifying model for many types of optimization problems. Despite the fact that the D-Wave machine is not a universal quantum computer, it has been capable of generating efficient and accurate solutions to several computational problems. Several research groups have reported on practical work to evaluate the performance of D-Wave systems [4, 6]. Several graph problems were converted into QUBO or similar optimization problems and tested on the D-Wave computer. Examples include graph isomorphism [6], vertex coloring [19, 16], broadcast-time [5] and spanning tree calculation [17] problems.

In this work, we solve the Bounded-Depth Steiner Tree problem using a D-Wave quantum computer. The problem is restated as a D-Wave feasible QUBO

formulation and is then tested on a D-Wave computer. The proposed method is also adapted for the Bounded-Depth Minimum Spanning Tree problem. Results of our experiment are presented and compared with classically obtained verifications. Accordingly, the paper is organized as follows: Section 1 contains the introduction. In Section 2, we present necessary definitions and describe the Bounded-Depth Steiner Tree problem. Section 2.1 contains the QUBO formulation with its proof of correctness. Section 3 talks about a special case of Bounded-Depth Steiner Tree problem that is known as the Bounded-Depth Minimum Spanning Tree problem. Section 4 contains our experimental results and Section 5 finishes with a conclusion.

2 Bounded-Depth Steiner Tree

If $G = (V, E)$ is a graph and a tree $T = (V_T, E_T)$ satisfies the conditions that $V_T \subseteq V$, $E_T \subseteq E$, and $u, v \in V_T$ for every $(u, v) \in E_T$, then, T is a **subtree** of G. The minimum spanning tree is a subtree of a (edge-) weighted graph $G = (V, E)$ with a minimum total cost that spans all vertices. The Steiner Tree problem specifies a set $U \subseteq V$ named terminal vertices that are to be connected by the subtree while the other vertices, called Steiner vertices, may or may not be contained in the subtree. A Steiner tree is a **rooted tree**, a tree with a specified root vertex. It is also called an **arborescence** with all its edges point away from the root. We add a depth bound to this problem which makes it a slight variant of Steiner Tree problem: the Bounded-Depth Steiner Tree problem.

Definition 1 *Considering an undirected weighted graph $G = (V, E)$ with non-negative weights $c_{u,v}$ associated with each edge $(u, v) \in E$. Given a set $U \subset V$ of terminal vertices with vertex $v_0 \in U$ specified as the root vertex and a natural number h, the **Bounded-Depth Steiner Tree** problem is to find a minimum cost subtree T of G such that there exists a path in T from v_0 to each vertex in terminal set U and the distance not exceeding h.*

As the Steiner tree problem arises in telecommunications networks design, the importance of bounded-depth Steiner trees become relevant when a maximum bound on transmission delay is a requirement [11]. More relevant real-world optimization problems such as the lot-sizing problems in production planning and phylogenetic trees in biology have been studied using this model [14, 2, 20].

2.1 QUBO formulation

Quadratic Unconstrained Binary Optimization (QUBO) is a mathematical problem of minimizing a quadratic objective function $F(\mathbf{x}) = \mathbf{x}^{\mathbf{T}} Q \mathbf{x}$, where Q is a real-valued symmetric matrix[1] and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a vector of binary variables. In this work, all QUBOs considered are of the following form

[1] Some times this problem is stated for upper-triangular matrices; it is straightforward to convert between these two equivalent forms.

$$x^* = \min_{\mathbf{x}} \sum q_{ij} x_i x_j, \quad \text{where } x_i, x_j \in \{0,1\}.$$

The problem is NP-hard in general. It has been extensively studied because it is a unifying model for many combinatorial optimization problems that can be formulated into QUBO form.
In this subsection, we present our QUBO formulation which requires $\mathcal{O}(h|V|^2)$ binary variables in worst case and produces a QUBO matrix with a density of $\mathcal{O}(h|V|^2)$.
The variables involved in this formulation are

- $x_{v_0,u,1}$ for each edge $(v_0, u) \in E$.
- $x_{u,v,i}$ and $x_{v,u,i}$ for each edge $(u, v) \in E$ with $u, v \neq v_0$ and $2 \leq i \leq h$

When $x_{u,v,i} = 1$, it means that edge (u, v) is contained in a directed optimal solution tree such that u is closer to the root than v and vertex v is in depth i. As v_0 is specified as root, we assume that the arcs of the arbor essence are directed away from v_0. Therefore, we use a single variable $x_{v_0,u,1}$ to represent each edge $(v_0, u) \in E$. In total, we need $2(h-1)(|E| - \deg_G(v_0)) + \deg_G(v_0)$ binary variables. In the worst case, $|E|$ is $\mathcal{O}(|V|^2)$, the complexity is then $\mathcal{O}(h|V|^2)$ in terms of variable size.

The objective function $F(\mathbf{x})$ to be minimized is stated as follows.

$$F(\mathbf{x}) = O(\mathbf{x}) + A \cdot P(\mathbf{x}) \tag{1}$$

where

$$P(\mathbf{x}) = |V| \cdot (P_1(\mathbf{x}) + P_2(\mathbf{x})) + P_3(\mathbf{x})$$

with

$$P_1(\mathbf{x}) = \sum_{v \in U \setminus \{v_0\}} \left(1 - \sum_{(u,v) \in E} \sum_{i=1}^{h} x_{u,v,i}\right)^2 \tag{2}$$

$$P_2(\mathbf{x}) = \sum_{v \in V \setminus U} \sum_{i=1}^{h} \left(\sum_{(u,v),(w,v) \in E} x_{u,v,i} x_{w,v,i} \right) \tag{3}$$

$$P_3(\mathbf{x}) = \sum_{v \in V \setminus \{v_0\}} \left(\sum_{(u,v) \in E} \sum_{i=2}^{h} x_{u,v,i} \left(1 - \sum_{(w,u) \in E} x_{w,u,i-1}\right) \right)$$

$$O(\mathbf{x}) = \sum_{(u,v) \in E} \sum_{i=2}^{h} c_{u,v} x_{u,v,i} + \sum_{(v_0,u) \in E} c_{v_0,u} x_{v_0,u,1}$$

and

$$A = E_T \max_{(u,v)\in E}(c_{u,v}) + 1$$

The terms are interpreted as follows.

- $P_1(\mathbf{x}) = 0$ when every terminal vertex excluding the root, that is $v \in U\backslash\{v_0\}$, has exactly one incoming arc and one assigned depth. Otherwise, $P_1(\mathbf{x})$ adds a positive penalty to objective function.
- $P_2(\mathbf{x}) = 0$ when every Steiner vertex $v \in V\backslash U$ has at most one incoming arc and at most one assigned depth. Otherwise, $P_2(\mathbf{x})$ adds a positive penalty to objective function.
- $P_3(\mathbf{x})$, on the other hand, penalize with a positive number if there exists an arc (u, v) in solution tree such that u is not parent of v. We also observe that $\sum_{(w,u)\in E} x_{w,u,i-1}$, which represents $\deg^-(u)$ in T, may exceed 1 and make $P_3(\mathbf{x})$ negative. However, we claim that the combined sum $|V| \cdot (P_1(\mathbf{x}) + P_2(\mathbf{x})) + P_3(\mathbf{x})$ is positive.
- The term $O(\mathbf{x})$ sums up the cost of each edge that is present in an optimal solution.
- A penalty scalar A is finally applied to $P(\mathbf{x})$ that is essential to make this formulation work correctly.

A **digraph** $D = (V, A)$ is a pair of non-empty sets—set V of vertices and set A of directed edges or arcs. The arc (u, v) is an outgoing arc of vertex u, and an incoming arc of vertex v. The **indegree** of a vertex $v \in V$, denoted by $\deg^-(v)$, counts the edges going into v, while the **outdegree** of a vertex $v \in V$ counts the edges going out of v and is denoted by $\deg^+(v)$. The notation $\deg(v)$ is defined as the sum of the indegree and the outdegreee of v:

$$\deg(v) = \deg^+(v) + \deg^-(v) \tag{4}$$

Once the optimal solution of the generated QUBO is obtained, $F(\mathbf{x}^*)$ gives the minimum cost and $\mathbf{x}^*$ encodes a corresponding minimum cost Steiner tree satisfying the depth constraint. Note that feasible solution may not exist for some instances of this problem. The scalar A can be used as a cut-off integer to decide whether a minimized value of the QUBO encodes a optimal solution or not. We will show that in due course.

2.2 Proof of correctness

Observe that a variable assignment $\mathbf{x} \in \mathbb{Z}_2^{2(h-1)(|E|-\deg_G(v_0))+\deg_G(v_0)}$ for the QUBO encodes a directed subgraph $S_{\mathbf{x}} = (V_{\mathbf{x}}, A_{\mathbf{x}}) \subseteq G$:

$$V_{\mathbf{x}} = \bigcup_{i=0}^{h} V_{\mathbf{x},i}$$
$$A_{\mathbf{x}} = \bigcup_{i=1}^{h} A_{\mathbf{x},i}$$

where

$$V_{\mathbf{x},i} = \{v_0\} \cup \{v \in V \mid \exists u \in V \backslash \{v\} \text{ such that } x_{u,v,i} = 1 \text{ or } x_{v,u,i} = 1\}$$
$$A_{\mathbf{x},i} = \{(u,v) \mid x_{u,v,i} = 1 \text{ or } x_{v,u,i} = 1\}$$

Lemma 1. *Let $\mathbf{x} \in \mathbb{Z}_2^{2(h-1)(|E|-\deg_G(v_0))+\deg_G(v_0)}$. Then $S_{\mathbf{x}}$ is a rooted Steiner tree for terminal vertices U in G with depth constraint h if and only if $P(\mathbf{x}) = 0$.*

Proof. Given a graph $G = (V, E)$, depth constraint h and v_0 specified as the root, we can represent a bounded-depth Steiner tree T for G as a sequence of vertex sets $\{V_0 = \{v_0\}, V_1, \ldots, V_h\} \subset P(V)$, which represents the vertices in depth i, and arc sets $A_1, A_2, \ldots, A_h$ such that for every $1 \leq i \leq h$:

1. Each arc in A_i is an oriented edge of E.
2. $V_i = \{v \mid (u,v) \in A_i, u \in V_{i-1}, v \in V_i\}$.
3. For any $(u,v) \in A_i, u \in V_{i-1}$ and $v \in V_i$.
4. A terminal vertex $v \in U$ only appears in one of the vertex sets.
5. A Steiner vertex $v \in V \backslash U$ appears in at most one of the vertex sets.
6. $U \subseteq (V_0 \cup V_2 \cup \cdots \cup V_h)$.

With a binary vector $\mathbf{x} \in \mathbb{Z}_2^{2(h-1)(|E|-\deg_G(v_0))+\deg_G(v_0)}$, we have

$$V_{\mathbf{x},i} = \{v \in V \mid x_{u,v,i} = 1\}, \text{ where } 1 \leq i \leq h$$
$$A_{\mathbf{x},i} = \{(u,v) \in E \mid x_{u,v,i} = 1\}, \text{ where } 1 \leq i \leq h$$

All the six criteria hold when $P(\mathbf{x}) = 0$:

1. *Each arc in $A_{\mathbf{x},i}$ is an oriented edge of E.*
   This holds by the definition of variable $x_{u,v,i}$.
2. $V_{\mathbf{x},i} = \{v \mid (u,v) \in A_{\mathbf{x},i}, u \in V_{\mathbf{x},i-1}, v \in V_{\mathbf{x},i}\}$.
   Suppose $v \in V_{\mathbf{x},i}$, which means that $x_{u,v,i} = 1$. If $u = v_0$, then $u \in V_{\mathbf{x},i-1}$ holds by the definition, $v_0 \in V_0$. If $u \neq v_0$, suppose a contradiction $u \notin V_{\mathbf{x},i-1}$. In such case, $1 - \sum_{(u,v)\in E} x_{w,u,i-1} = 1$ which leads to $P_2(\mathbf{x}) > 0$. But $P_2(\mathbf{x}) = 0$, so $u \in V_{\mathbf{x},i-1}$.
3. *For any $(u,v) \in A_{\mathbf{x},i}, u \in V_{\mathbf{x},i-1}$ and $v \in V_{\mathbf{x},i}$.*
   If $(u,v) \in A_{\mathbf{x},i}$, then we have $x_{u,v,i} = 1$. Thus, $u \in V_{\mathbf{x},i-1}, v \in V_{\mathbf{x},i}$ holds as we illustrated in previous criteria.

4. *A terminal vertex $v \in U$ only appears in one of the vertex sets.*
   For a contradiction suppose that we have a terminal vertex $v \in U$ appears in more than one of the vertex sets or appears in none of the vertex sets but $P_1(\mathbf{x}) = 0$. That is, $\sum_i x_{u,v,i} > 2$ or $\sum_i x_{u,v,i} = 0$, which implies $P_1(\mathbf{x}) > 0$, a contradiction.
5. *A Steiner vertex $v \in V \backslash U$ appears in at most one of the vertex sets.*
   For a contradiction suppose that we have a Steiner vertex $v \in V \backslash U$ appears in more than one of the vertex sets but $P_2(\mathbf{x}) = 0$. That is, $\sum_i x_{u,v,i} > 2$, which implies $P_2(\mathbf{x}) > 0$, a contradiction.
6. $U \subseteq (V_0 \cup V_{\mathbf{x},1} \cup \cdots \cup V_{\mathbf{x},h})$.
   Suppose there exists a vertex v such that $v \in U$ and $v \notin (V_0 \cup V_{\mathbf{x},1} \cup \cdots \cup V_{\mathbf{x},h})$. This suggests $\sum_{(u,v)\in E} \sum_i x_{u,v,i} = 0$ which leads to $P_1(\mathbf{x}) > 0$.

As noted in the term $P_3(\mathbf{x})$, if $\deg^-(u) \geq 2$ for some $w \in V$, $P_3(\mathbf{x})$ becomes negative. Suppose for some vertex $v \in V \backslash \{v_0\}$, there exists a vertex u where $\deg^-(u) \geq 1$ and $(u, v) \in A_{\mathbf{x}}$. Let $\deg^-(u)$ incremented by 1. As a consequence,

$$P_3(\mathbf{x}) = \sum_{u \in V \backslash \{v_0\}} \deg^+(u)(1 - \deg^-(u))$$

is decreased by $\deg^+(u) < |V|$. On the other hand, if $v \in U$,

$$|V| P_1(\mathbf{x}) = |V| \sum_{u \in U \backslash \{v_0\}} \left(1 - \deg^-(u)\right)^2$$

will increase by $|V|$ at least. Otherwise,

$$|V| P_2(\mathbf{x}) = |V| \sum_{u \in V \backslash U} \binom{\deg^-(u)}{2}$$

will increase by at least $|V|$. Note that the third dimension variable i representing its depth is ignored in the calculation as this operation has not effect on the final result. Thus, the combined sum $P(\mathbf{x}) = |V|(P_1(\mathbf{x}) + P_2(\mathbf{x})) + P_3(\mathbf{x})$ is positive.

Corollary 1. *Let $\mathbf{x} \in \mathbb{Z}_2^{2(h-1)(|E|-\deg_G(v_0))+\deg_G(v_0)}$, then $F(\mathbf{x}) \geq A$ if and only if $S_{\mathbf{x}}$ is not an h-constrained Steiner tree of G.*

Proof. If $S_{\mathbf{x}}$ is not an h-constrained Steiner tree, then by Lemma 1, $P(\mathbf{x}) \neq 0$. This implies $P(\mathbf{x}) \geq 1$ since $P(\mathbf{x})$ is a non-negative integer. Therefore, $F(\mathbf{x}) = O(\mathbf{x}) + A \cdot P(\mathbf{x}) \geq A$. On the other hand, if $S_{\mathbf{x}}$ is a h-constrained Steiner tree of G, we have $P(\mathbf{x}) = 0$ and $F(\mathbf{x}) = O(\mathbf{x})$. Let $T = (V_T, E_T)$ be a Steiner tree, grounded on $c(u, v) \geq 0$ with $(u, v) \in E_T$ and $|V| - 1 = |E_T|$, we have:

$$F(\mathbf{x}) \leq |E_T| \cdot \max_{(u,v)\in E}(c_{u,v}) + 1 < A.$$

Theorem 1. *The QUBO formulation in (1) is correct.*

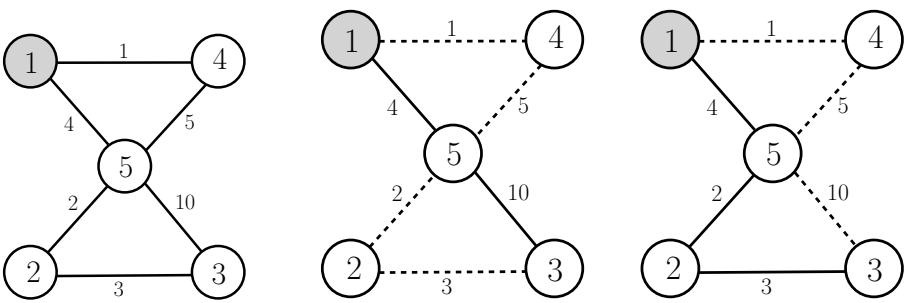
Proof. If a graph $G = (V, E)$ has a h-constrained minimum cost Steiner tree, there exists an assignment $\mathbf{x} \in \mathbb{Z}_2^{2(h-1)(|E|-\deg_G(v_0))+\deg_G(v_0)} < A$ such that $S_{\mathbf{x}}$ is a Steiner tree. Assume that $\mathbf{x}^*$ is the optimal variable assignment of $F(\mathbf{x})$, then by Lemma 1,

$$F(\mathbf{x}^*) = \min O_{\mathbf{x}} = \min C(S_{\mathbf{x}}).$$

and $S_{\mathbf{x}^*}$ is a minimum cost Steiner tree of graph G. Therefore, (1) is a QUBO formulation of bounded-depth Steiner tree problem.

2.3 Example: the graph Butterfly

Consider the weighted Butterfly graph shown in Figure 1a. The vertex set is $V = \{1, 2, 3, 4, 5\}$, the edge set is $E = \{(1, 4), (1, 5), (2, 3), (2, 5), (3, 5), (4, 5)\}$ and a cost for each edge: $c_{1,4} = 1, c_{1,5} = 4, c_{2,3} = 3, c_{2,5} = 2, c_{3,5} = 10, c_{4,5} = 5$. With $v_0 = 1$ specified as root and a terminal set $U = \{1, 3, 5\}$, we wish to find the minimum spanning tree with different bounded-depth 2 and 3.
Figure 1b and Figure 1c depict the bounded-depth Steiner tree of Butterfly for $h = 2$ and $h = 3$ respectively. Note that the vertex filled with gray color is root vertex and dashed-lines denote the edges that are not in solution tree.



$v_0 = 1, U = \{1, 3, 5\}$ (a) graph Butterfly

$h = 2$ (b) 2-constrained Steiner tree

$h = 3$ (c) 3-constrained Steiner tree

Fig. 1: A weighted Butterfly and solutions with different bounded-depth

The number of variables required for the two instances are $2(2-1)(6-2)+2 = 10$ and $2(3-1)(6-2)+2 = 18$. After processing the constants and linear terms, we obtain the diagonal symmetric matrix Q_1 and Q_2 shown in Table 1 and Table 2.
The optimal solutions for the two QUBO instances are:

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

Table 1: QUBO matrix Q_1 for graph Butterfly with $h = 2$

variables	$x_{1,4,1}$	$x_{1,5,1}$	$x_{2,3,2}$	$x_{2,5,2}$	$x_{3,2,2}$	$x_{3,5,2}$	$x_{4,5,2}$	$x_{5,2,2}$	$x_{5,3,2}$	$x_{5,4,2}$
$x_{1,4,1}$	1	0	0	0	0	0	-20.5	0	0	102.5
$x_{1,5,1}$	0	-201	0	205	0	205	205	-20.5	-20.5	-20.5
$x_{2,3,2}$	0	0	-161	0	0	0	0	0	205	0
$x_{2,5,2}$	0	205	0	-162	0	205	205	0	0	0
$x_{3,2,2}$	0	0	0	0	44	0	0	102.5	0	0
$x_{3,5,2}$	0	205	0	205	0	-154	205	0	0	0
$x_{4,5,2}$	-20.5	205	0	205	0	205	-159	0	0	0
$x_{5,2,2}$	0	-20.5	0	0	102.5	0	0	43	0	0
$x_{5,3,2}$	0	-20.5	205	0	0	0	0	0	-154	0
$x_{5,4,2}$	102.5	-20.5	0	0	0	0	0	0	0	46

Table 2: QUBO matrix Q_2 for graph Butterfly with $h = 3$

variables	$x_{1,4,1}$	$x_{1,5,1}$	$x_{2,3,2}$	$x_{2,3,3}$	$x_{2,5,2}$	$x_{2,5,3}$	$x_{3,2,2}$	$x_{3,2,3}$	$x_{3,5,2}$	$x_{3,5,3}$	$x_{4,5,2}$	$x_{4,5,3}$	$x_{5,2,2}$	$x_{5,2,3}$	$x_{5,3,2}$	$x_{5,3,3}$	$x_{5,4,2}$	$x_{5,4,3}$
$x_{1,4,1}$	1	0	0	0	0	0	0	0	0	0	-20.5	0	0	0	0	0	102.5	102.5
$x_{1,5,1}$	0	-201	0	0	205	205	0	0	205	205	205	205	-20.5	0	-20.5	0	-20.5	0
$x_{2,3,2}$	0	0	-161	205	0	0	0	-20.5	0	-20.5	0	0	0	0	205	205	0	0
$x_{2,3,3}$	0	0	205	-161	0	0	-20.5	0	0	0	0	0	-20.5	0	205	205	0	0
$x_{2,5,2}$	0	205	0	0	-162	205	0	0	205	205	205	205	0	-20.5	0	-20.5	0	-20.5
$x_{2,5,3}$	0	205	0	0	205	-162	-20.5	0	205	205	205	205	-20.5	0	0	0	0	0
$x_{3,2,2}$	0	0	0	-20.5	0	-20.5	44	0	0	0	0	0	102.5	102.5	0	0	0	0
$x_{3,2,3}$	0	0	-20.5	0	0	0	0	44	0	0	0	0	102.5	102.5	-20.5	0	0	0
$x_{3,5,2}$	0	205	0	0	205	205	0	0	-154	205	205	205	0	-20.5	0	-20.5	0	-20.5
$x_{3,5,3}$	0	205	-20.5	0	205	205	0	0	205	-154	205	205	0	0	-20.5	0	0	0
$x_{4,5,2}$	-20.5	205	0	0	205	205	0	0	205	205	-159	205	0	-20.5	0	-20.5	0	-20.5
$x_{4,5,3}$	0	205	0	0	205	205	0	0	205	205	205	-159	0	0	0	0	-20.5	0
$x_{5,2,2}$	0	-20.5	0	-20.5	0	205	102.5	102.5	0	0	0	0	43	0	0	0	0	0
$x_{5,2,3}$	0	0	0	0	-20.5	0	102.5	102.5	-20.5	0	-20.5	0	0	43	0	0	0	0
$x_{5,3,2}$	0	-20.5	205	205	0	0	0	-20.5	0	-20.5	0	0	0	0	-154	205	0	0
$x_{5,3,3}$	0	0	205	205	-20.5	0	0	0	-20.5	0	-20.5	0	0	0	205	-154	0	0
$x_{5,4,2}$	102.5	-20.5	0	0	0	0	0	0	0	0	0	-20.5	0	0	0	0	46	0
$x_{5,4,3}$	102.5	0	0	0	-20.5	0	0	0	-20.5	0	-20.5	0	0	0	0	0	0	46

As expected, each of these two assignment gives the optimal solution $S_{\mathbf{x}} = (V_{\mathbf{x}}, E_{\mathbf{x}})$ where $E_{\mathbf{x}} = \{(1,5),(5,3)\}$ for $h = 2$ and $E_{\mathbf{x}} = \{(1,5),(5,2),(2,3)\}$ for $h = 3$.

3 Bounded-Depth Minimum Spanning Tree

The concept of Minimum Spanning Tree (MST) is a core part of graph theory. Classic polynomial-time algorithms exist for the minimum spanning tree problem such as well-known Prim's algorithm and Kruskal's algorithm [13, 18]. However, with a depth constraint added, the problem becomes NP-hard [10, 9, 15].

The Bounded-Depth MST problem arises in the design of centralized telecommunication networks with quality of service constraints [21]. Various applications can be found in [1].

Definition 2 *Given an weighted graph $G = (V, E)$ with a cost function c: $E \mapsto \mathbb{R}$ and a natural number h, the* ***Bounded-Depth Minimum Spanning Tree*** *problem is to find a minimum cost spanning tree T of G, such that there exists a path in T from a specified root vertex v_0 to any other vertex with at most h edges.*

From the earlier Definition 1, it is quite clear that the Bounded-Depth Minimum Spanning Tree problem is the special case of the bounded-depth Steiner tree problem where the terminal set of Steiner tree is the set of all vertices (i.e., $U = V$). Based on this, we can adapt the formulation (1) for Bounded-Depth MST problem. First, the variables involved in this QUBO formulation are the same as these in 2.1. Then, we replace set U in Equation (2) by set V and remove the second penalty term (3) since $V \backslash U = \emptyset$. With other terms remained, we obtain the following:

$$F(\mathbf{x}) = O(\mathbf{x}) + A \cdot P(\mathbf{x}) \tag{5}$$

where

$$P(\mathbf{x}) = |V| \cdot P_1(\mathbf{x}) + P_2(\mathbf{x})$$

with

$$P_1(\mathbf{x}) = \sum_{v \in V \backslash \{v_0\}} \left(1 - \sum_{(u,v) \in E} \sum_{i=1}^{h} x_{u,v,i}\right)^2$$

$$P_2(\mathbf{x}) = \sum_{v \in V \backslash \{v_0\}} \left(\sum_{(u,v) \in E} \sum_{i=2}^{h} x_{u,v,i} \left(1 - \sum_{(w,u) \in E} x_{w,u,i-1}\right)\right)$$

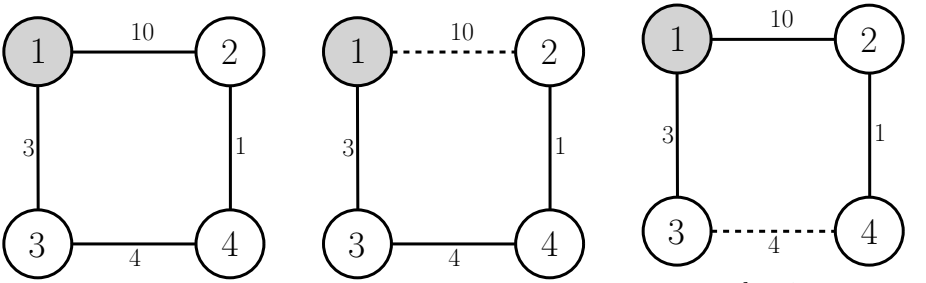
$$O(\mathbf{x}) = \sum_{(u,v) \in E} \sum_{i=2}^{h} c_{u,v} x_{u,v,i} + \sum_{(v_0,u) \in E} c_{v_0,u} x_{v_0,u,1}$$

and

$$A = (|V| - 1) \cdot \max_{(u,v) \in E} (c_{u,v}) + 1$$

3.1 Example: a weighted graph C_4

Consider the graph C_4 in Figure 2a where vertex set $V = \{1, 2, 3, 4\}$, edge set $E = \{(1, 2), (1, 3), (2, 4), (3, 4)\}$ and a cost for each edge: $c_{1,2} = 1, c_{1,3} = 3, c_{2,4} = 10, c_{3,4} = 4$. Given a depth bound $h = 2$ and a dedicated root $v_0 = 1$, we wish to find the bounded-depth minimum spanning tree. A solution tree $T = (V, E_T)$



(a) graph C_4 (b) a MST of C_4 (c) a bounded-depth MST of C_4

Fig. 2: A weighted C_4 and a solution

with $E_T = \{(1,2),(1,3),(2,4)\}$, as shown in Figure 2c, can be observed clearly. Note that we use dashed line to mark these edges that are not in the solution. The formulation requires $2 \cdot (2-1) \cdot (4-2) + 2 = 6$ variables:

$$\mathbf{x} = \{x_{1,2,1}, x_{1,3,1}, x_{2,4,2}, x_{3,4,2}, x_{4,2,2}, x_{4,3,2}\}$$

Then, we compute each term:

$$\begin{aligned}
O(\mathbf{x}) &= \sum_{(u,v)\in E} \sum_{i=2}^{h} c_{u,v} x_{u,v,i} + \sum_{(v_0,u)\in E} c_{v_0,u} x_{v_0,u,1} \\
&= 10x_{1,2,1} + 3x_{1,3,1} + (x_{2,4,2} + x_{4,2,2}) + 4(x_{3,4,2} + x_{4,3,2}) \\
P_1(\mathbf{x}) &= \sum_{v\in V\setminus\{v_0\}} \left(1 - \sum_{(u,v)\in E} \sum_{i=1}^{h} x_{u,v,i}\right)^2 \\
&= \begin{array}{l}(1-(x_{1,2,1}+x_{4,2,2}))^2 + (1-(x_{1,3,1}+x_{4,3,2}))^2 \\ \quad + (1-(x_{2,4,2}+x_{3,4,2}))^2\end{array}
\end{aligned}$$

$$\begin{aligned}
&\begin{array}{l} 1 - 2(x_{1,2,1}+x_{4,2,2}) + x_{1,2,1}^2 + x_{4,2,2}^2 + 2x_{1,2,1}x_{4,2,2} \\ = \; + 1 - 2(x_{1,3,1}+x_{4,3,2}) + x_{1,3,1}^2 + x_{4,3,2}^2 + 2x_{1,3,1}x_{4,3,2} \\ \quad + 1 - 2(x_{2,4,2}+x_{3,4,2}) + x_{2,4,2}^2 + x_{3,4,2}^2 + 2x_{2,4,2}x_{3,4,2}\end{array} \\
P_2(\mathbf{x}) &= \sum_{v\in V\setminus\{v_0\}} \left(\sum_{(u,v)\in E} \sum_{i=2}^{h} x_{u,v,i} \left(1 - \sum_{(w,u)\in E} x_{w,u,i-1}\right)\right) \\
&= x_{2,4,2}(1 - x_{1,2,1}) + x_{3,4,2}(1 - x_{1,3,1})
\end{aligned}$$

$$A = (4-1) \cdot 10 + 1 = 31$$

There are some linear terms as well as some constants that we need to process before we can encode them in a QUBO instance. As QUBO problem only allows quadratic terms, all linear terms x in $F(\mathbf{x})$ are replaced by x^2 grounding on the fact $x = x^2$. Furthermore, all the constants are removed since it does not change the optimality of the solution. After the processing we obtain the following:

$$\begin{aligned} F(\mathbf{x}) = &- 114x1, 2, 1^2 - 121x_{1,3,1}^2 - 92x_{2,4,2}^2 - 89x_{3,4,2}^2 - 92x_{4,2,2}^2 - 120x_{4,3,2}^2 \\ &- 31x_{1,2,1}x_{2,4,2} + 248x_{1,2,1}x_{4,2,2} - 31x_{1,3,1}x_{3,4,2} + 248x_{1,3,1}x_{4,3,2} \\ &+ 248x_{2,4,2}x_{3,4,2} \end{aligned}$$

With the coefficients of each quadratic term in $F(\mathbf{x})$ set to the corresponding entry, the complete QUBO matrix is obtained, as shown in Table 3. An optimal solution $\mathbf{x}^*$ for this QUBO instance generated by a QUBO solver is

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

This encodes the optimal solution tree $S_{\mathbf{x}} = (V, E_{\mathbf{x}^*})$ where $V = \{1, 2, 3, 4\}$ and $E_{\mathbf{x}^*} = \{(1, 2), (1, 3), (2, 4)\}$.

Table 3: QUBO matrix for graph C_4 with $h = 2$

variables	$x_{1,2,1}$	$x_{1,3,1}$	$x_{2,4,2}$	$x_{3,4,2}$	$x_{4,2,2}$	$x_{4,3,2}$
$x_{1,2,1}$	-114	0	-31	0	248	0
$x_{1,3,1}$		-121	0	-31	0	248
$x_{2,4,2}$			-92	248	0	0
$x_{3,4,2}$				-89	0	0
$x_{4,2,2}$					-92	0
$x_{4,3,2}$						-120

4 Solving the Bounded-Depth Problems on a D-Wave Machine

There is an issue arising from the fact that *not* all pairwise interactions between qubits are physically connected on the D-Wave machine. As a consequence, a guest graph (e.g. the QUBO matrices are viewed as graph adjacency matrices) where computation takes place must be transformed into a subgraph of a Chimera host graph (D-Wave architecture). To ensure connectivity of all logical variables, a *minor embedding* of the guest graph into the host graph is necessary [22]. The decision problem whether a graph contains another one as a minor is a non-trivial task due to its NP-complete nature. However, if one fixes the host graph (or family of hosts) we have several-known polynomial-time methods for embedding cliques. For the Chimera graph architecture, any graph (including

cliques) of n vertices can be minor embedded onto a Chimera host graph if it has $O(n^2)$ nodes (see [23]). For the computational experiments in this paper a randomized heuristic minor embedding algorithm [3, 8] was employed to find minor embeddings.

In the procedure of graph embedding, a logical variable (guest vertex) may be mapped into one or more physical qubits (host vertices). The set of physical qubits representing a logical variable is called a *chain*. Obviously, the existence of big chains consumes additional physical qubits and hence reduce the maximum problem sizes that can be solved. In the following experiments, we will see that minimizing the chain size is an essential factor to the success probability as well.

4.1 Experimental results

In the initial experiment, we run two groups of test cases on D-Wave 2X for the Bounded-Depth Steiner Tree problem, which consist of test cases for several small weighted graphs (see Appendix A) with two different sets of depth constraints. Each weighted graph, together with a nominated root (red vertex), a terminal set (green vertices) and a depth constraint, comprise an instance of Bounded-Depth Steiner Tree problem. We sum up the experimental results in Table 4 and Table 5. Column h represents the depth constraints assigned to each case.

From the tables, we can see that the *success probability* decreases as the number of *physical qubits* required increases. In Table 4, the selections of depth constraint are relatively smaller comparing to the depth constraints in Table 5. Recall that the QUBO objective function (1) requires more variables when the depth constraint h becomes larger. In other words, more physical qubits are needed after embedding as h increases resulting in the lower success rates. For these QUBO instances whose embedding chain size is under three, we observe a success rate at 100 percent. This observation applies in the experimental results of Bounded-Degree Problems (MST and Steiner tree) as well, which again suggests a better accuracy of the D-Wave computer when the topology of a problem is closer to the host graph. The *density*, on the other hand, has no direct impact on the success probability, which makes a difference between empirical and theoretical result.

The second group of test cases is created for Bounded-Depth Minimum Spanning Tree problem. They consist of ten K_6 graphs with their edges labeled with different weights ranging from 1 to 10. For each weighted K_6 graph, the $v_0 = 0$ is specified as root (see Appendix B) and given depth constraints $h = 2$ and $h = 3$. A total of 15000 trials are done for each test case, and the results are shown in Table 6. The best valid solutions found are listed in column *Minimum D-Wave*.

We notice that the *physical qubits* required after embedding differs from one to another even for the cases having the same size of logical qubits. The differences are due to the heuristic minor embedding algorithm used. The algorithm uses a randomized method during its initial stage and results in the slight fluctuation of number of physical qubits required as well as the embedding chain size [3]. Then, we observe a significant decline in the *success probability* as problem size becomes large. With depth bound $h = 3$, the *physical qubits* required

exceed 400, and the numbers of chain size are over 10. Only one of ten test cases succeeds in finding a ground state energy among its 15000 trials.

Overall, the size of problems that can be solved is significantly limited by the chip capacity and the distinctive architecture of D-Wave 2X. Thus, a quantum advantage cannot be observed yet. In their experiment on finding maximum cliques on the D-Wave quantum annealer, Chapuis et al [7]. generated some large graphs that can be embedded into the hardware and reported a quantum speedup. Their average chain size is relatively small which leads to considerable success. Thus, an efficient graph minor embeddings into Chimera graphs algorithm plays an essential role in the success of D-Wave quantum annealer.

Table 4: Results for the Bounded-Depth Steiner Tree 1

				Logical	Physical	Embedding		Success	Optimal
Graph	Order	Size	h	Qubits	Qubits	Max Chain	Density	Probability	Answer
Bull	5	5	2	9	14	3	40	15000/15000	22
Butterfly	5	6	2	8	8	1	44.44	15000/15000	2
C_4	4	4	2	6	6	1	52.38	15000/15000	5
C_5	5	5	2	8	8	1	38.89	15000/15000	20
C_6	6	6	3	18	41	3	26.9	15000/15000	21
C_7	7	7	3	22	58	5	22.92	15000/15000	23
C_8	8	8	4	38	137	5	18.35	40/15000	41
C_9	9	9	4	44	141	5	15.25	254/15000	34
C_{10}	10	10	5	66	320	6	12.71	1/15000	33
C_{11}	11	11	5	74	390	7	11.6	0/15000	49
C_{12}	12	12	6	102	621	12	8.72	0/15000	12
Diamond	4	5	2	8	12	2	41.67	15000/15000	10
Grid2x3	6	7	3	22	65	5	27.67	3212/15000	7
Grid3x3	9	12	4	57	341	9	16.33	0/15000	16
Hexahedral	8	12	4	57	422	13	17.48	0/15000	17
House	5	6	2	10	12	2	32.73	15000/15000	8
$K_{2,3}$	5	6	2	9	10	2	33.33	15000/15000	12
$K_{3,3}$	6	9	3	27	120	6	26.46	18/15000	18
K_3	3	3	2	4	4	1	80	15000/15000	10
K_4	4	6	2	9	16	3	48.89	15000/15000	9
K_5	5	10	2	16	36	3	38.24	11136/15000	12
Q_3	8	12	4	57	404	11	17.42	0/15000	20
Wagner	8	12	4	57	400	9	17.12	0/15000	9

5 Conclusion

The main contribution of this paper is the development of QUBO formulations with small number of required variables for the Bounded-Depth Steiner Tree and Bounded-Depth Minimum Spanning Tree problems. We have accompanied

Table 5: Results for the Bounded-Depth Steiner Tree 2

Graph	Order	Size	h	Logical Qubits	Physical Qubits	Embedding Max Chain	Embedding Density	Success Probability	Optimal Answer
Bull	5	5	5	33	176	11	15.18	7/15000	15
Butterfly	5	6	5	20	43	3	4.72	10019/15000	2
C_4	4	4	4	14	39	4	4.42	11979/15000	5
C_5	5	5	5	26	93	6	8.11	1642/15000	18
C_6	6	6	6	42	229	8	15.12	13/15000	19
C_7	7	7	6	52	310	11	13.85	0/15000	20
C_8	8	8	5	50	245	7	13.79	5/15000	41
C_9	9	9	5	58	277	7	10.04	3/15000	34
C_{10}	10	10	6	82	687	19	9.2	0/15000	33
C_{11}	11	11	6	92	570	15	11.31	0/15000	49
C_{12}	12	12	6	102	753	18	6.59	0/15000	12
Diamond	4	5	4	20	102	7	7.8	88/15000	8
Grid2x3	6	7	6	52	409	14	16.7	1/15000	7
Grid3x3	9	12	5	75	612	16	12.46	0/15000	16
Hexahedral	8	12	5	75	712	15	14.7	0/15000	17
House	5	6	5	34	199	8	14.7	23/15000	8
$K_{2,3}$	5	6	5	27	132	8	12.34	729/15000	12
$K_{3,3}$	6	9	6	63	817	25	16.58	0/15000	18
K_3	3	3	3	6	9	2	1.09	15000/15000	10
K_4	4	6	4	21	116	7	8.35	734/15000	9
K_5	5	10	5	52	811	23	22.99	1/15000	12
Q_3	8	12	5	75	786	22	15.97	0/15000	20
Wagner	8	12	5	75	746	17	14.82	0/15000	9

our research into the testing and analysis for the performance of a D-Wave 2X adiabatic quantum computer.

The final computational results imply that the probability of finding the ground state energy is affected by the quality of graph embedding. Higher success rates are observed for the instances with smaller embedding chain size. The hardware structure of all current generations of D-Wave adiabatic quantum computers (including the latest generation D-Wave 2000Q) are all Chimera graphs. Accordingly, it becomes valuable to explore more effective QUBO formulations that reduce qubits required and find graph minor embedding algorithms that can minimize chain size to maximize the success probability of the quantum annealing. The experiment results suggest the potential for growth on the capability of D-Wave. The latest D-Wave computer doubles the number of available qubits of D-Wave 2X and is claimed to work at a lower temperature, which implies a better success probability. A model with more qubits will also increase the size of the maximal complete guest graphs that can be embedded by a factor of 2 but remain the same runtime on D-Wave. Since the May 2011 release of their first device, D-Wave is doubling the number of qubits every two years which may lead to a quantum version of Moore's Law. Grounded on this assumption, the efforts

Table 6: Results for Bounded-Depth MST

Graph	Order	Size	h	Logical Qubits	Physical Qubits	Embedding Max Chain	Density	Success Probability	Minimum D-Wave	Optimal Answer
K_{6-1}	6	15	2	25	90	4	23.08	145/15000	9	9
K_{6-2}	6	15	2	25	85	5	23.08	198/15000	25	25
K_{6-3}	6	15	2	25	84	5	23.08	799/15000	13	13
K_{6-4}	6	15	2	25	92	5	23.08	565/15000	17	17
K_{6-5}	6	15	2	25	71	5	23.08	288/15000	21	21
K_{6-6}	6	15	2	25	96	7	23.08	131/15000	23	23
K_{6-7}	6	15	2	25	89	9	23.08	672/15000	23	23
K_{6-8}	6	15	2	25	89	7	23.08	729/15000	18	18
K_{6-9}	6	15	2	25	96	5	23.08	333/15000	17	17
K_{6-10}	6	15	2	25	80	4	23.08	216/15000	18	18
K_{6-1}	6	15	3	45	413	14	25.6	0/15000	10	7
K_{6-2}	6	15	3	45	468	16	25.6	0/15000	27	24
K_{6-3}	6	15	3	45	438	14	25.6	0/15000	16	12
K_{6-4}	6	15	3	45	435	15	25.6	0/15000	14	13
K_{6-5}	6	15	3	45	414	12	25.6	1/15000	18	18
K_{6-6}	6	15	3	45	487	16	25.6	0/15000	24	21
K_{6-7}	6	15	3	45	479	16	25.6	0/15000	21	19
K_{6-8}	6	15	3	45	424	18	25.6	0/15000	21	14
K_{6-9}	6	15	3	45	459	15	25.6	0/15000	18	16
K_{6-10}	6	15	3	45	443	14	25.6	0/15000	20	14

on this quantum system are encouraging. Recently announced by D-Wave Systems, future models will use a denser Pegasus graph architecture, which should help with the embedding and reliability of QUBO computations [12].

Acknowledgement

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

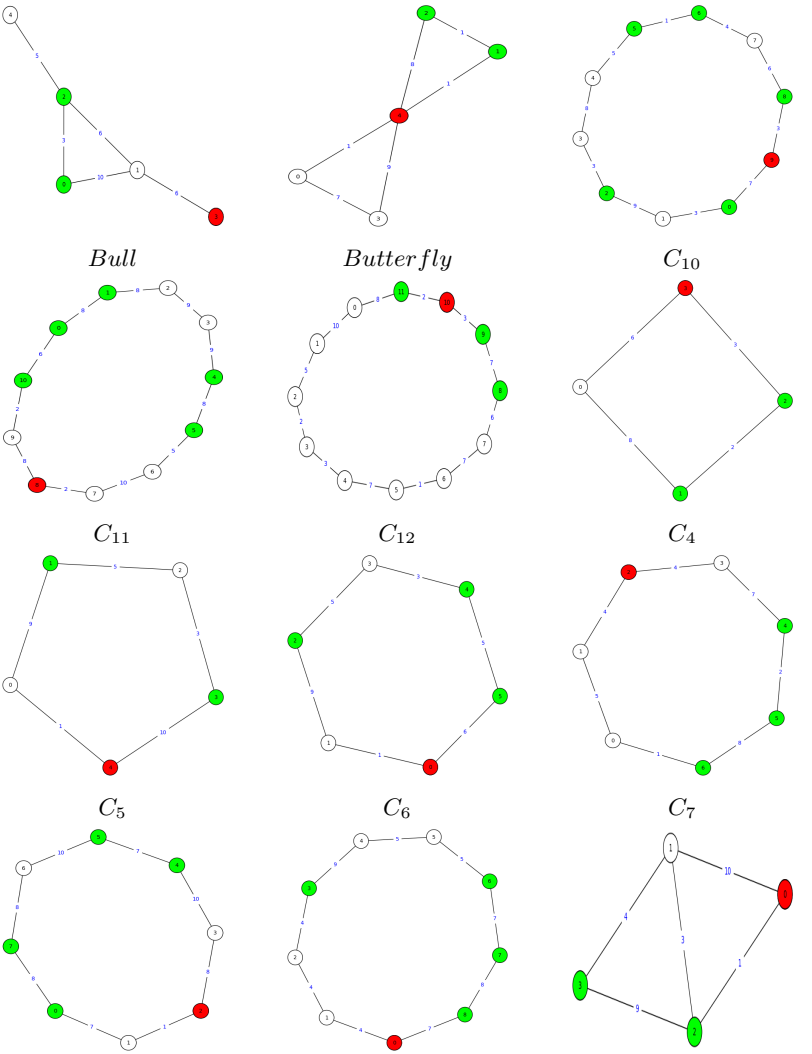
References

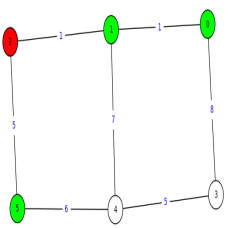
1. Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, et al. *Network flows: theory, algorithms, and applications*, volume 1. Prentice hall Englewood Cliffs, NJ, 1993.
2. Omer Angel, Abraham D Flaxman, and David B Wilson. A sharp threshold for minimum bounded-depth and bounded-diameter spanning trees and steiner trees in random networks. *Combinatorica*, 32(1):1–33, 2012.
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. *ACM SIGACT News*, 46(1):40–61, 2015.

5. Cristian S Calude and Michael J Dinneen. Solving the broadcast time problem using a D-Wave quantum computer. In *Advances in Unconventional Computing*, pages 439–453. Springer, 2017.
6. Cristian S Calude, Michael J Dinneen, and Richard Hua. QUBO formulations for the graph isomorphism problem and related problems. *Theoretical Computer Science*, 701:54–69, 2017.
7. Guillaume Chapuis, Hristo Djidjev, Georg Hahn, and Guillaume Rizk. Finding maximum cliques on the D-Wave quantum annealer. *Journal of Signal Processing Systems*, pages 1–15, 2018.
8. D-Wave. Programming with QUBOs. Technical Report 09-1002A-B, D-Wave Systems, Inc., 2013. Python Release 1.5.1-beta4 (for Mac/Linux).
9. Geir Dahl. The 2-hop spanning tree problem. *Operations Research Letters*, 23(1-2):21–26, 1998.
10. Luis Gouveia. Using the miller-tucker-zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research*, 22(9):959–970, 1995.
11. Luis Gouveia and Thomas L Magnanti. Network flow models for designing diameter-constrained minimum-spanning and steiner trees. *Networks*, 41(3):159–173, 2003.
12. Mark W Johnson. Future hardware directions of quantum annealing, 2018. Qubits Europe 2018 D-Wave Users Conveference, Munich, Germany https://www.dwavesys.com/sites/default/files/mwj_dwave_qubits2018.pdf.
13. Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.
14. Chin Lung Lu, Chuan Yi Tang, and Richard Chia-Tung Lee. The full steiner tree problem. *Theoretical Computer Science*, 306(1):55 – 67, 2003.
15. Prabhu Manyem and M Stallmann. Some approximation results in multicasting. *North Carolina State University*, 1996.
16. Anuradha Mahasinghe Michael J. Dinneen and Kai Liu. Finding the chromatic sums of graphs using a d-wave quantum computer. *The Journal of Supercomputing*, pages 1–18, 2019.
17. Mark A Novotny, Q L Hobl, J S Hall, and K Michielsen. Spanning tree calculations on D-Wave 2 machines. In *Journal of Physics: Conference Series*, volume 681(1), page 012005. IOP Publishing, 2016.
18. Robert Clay Prim. Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6):1389–1401, 1957.
19. Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.
20. Stefan Voß. The steiner tree problem with hop constraints. *Annals of Operations Research*, 86:321–345, 1999.
21. Kathleen A Woolston and Susan L Albin. The design of centralized networks with reliability and availability constraints. *Computers & Operations Research*, 15(3):207–217, 1988.
22. Zhongchen Yang and Michael J Dinneen. Graph minor embeddings for D-Wave computer architecture. Report CDMTCS-503, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, November 2016.
23. P. (Amanda) Yao and Richard Hua. Finding maximum-sized native clique embeddings: Implementing and extending the block clique embedding algorithm. Report

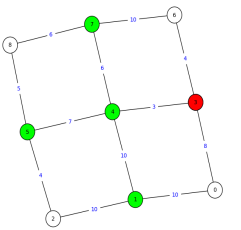
CDMTCS-523, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, March 2018.

A Test graphs for Bounded-Depth Steiner Tree

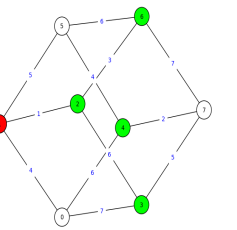




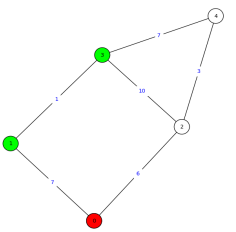
$Grid_{2\times 3}$



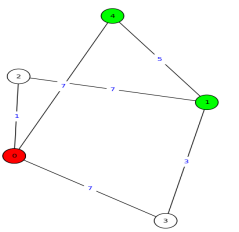
$Grid_{3\times 3}$



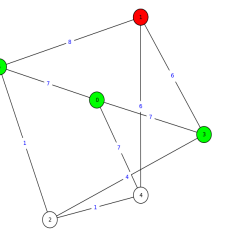
$Hexahedral$



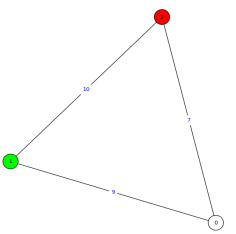
$House$



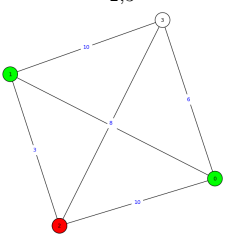
$K_{2,3}$



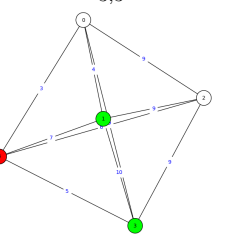
$K_{3,3}$



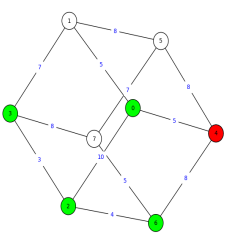
K_3



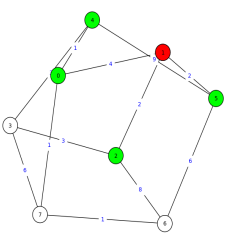
K_4



K_5

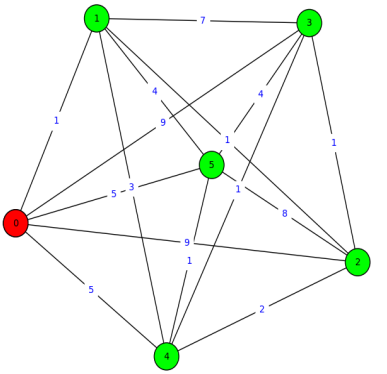


Q_3

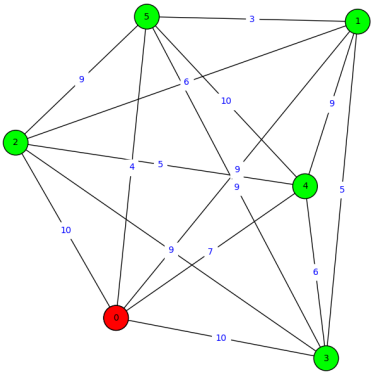


$Wagner$

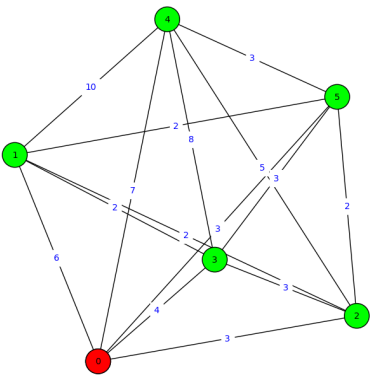
B Test graphs for Bounded-Depth MST



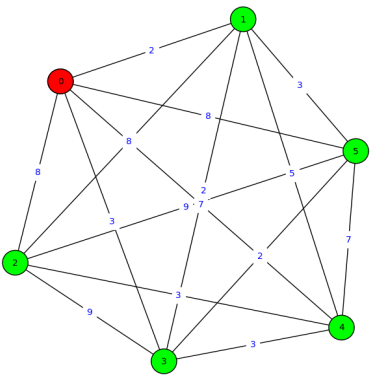
K_{6-1}



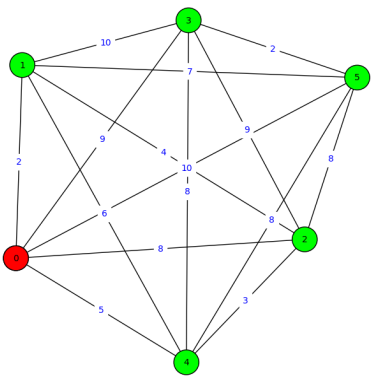
K_{6-2}



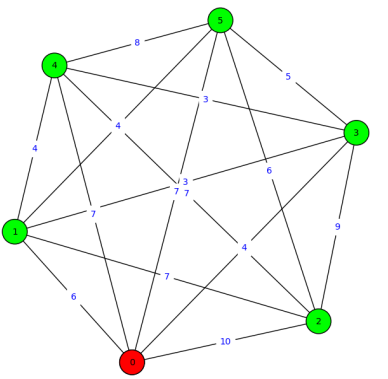
K_{6-3}



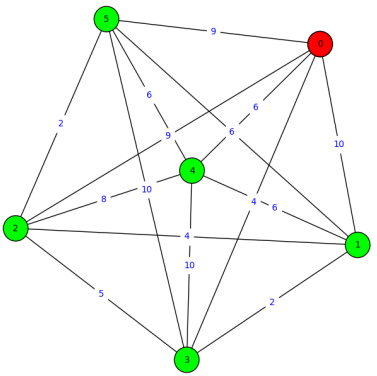
K_{6-4}



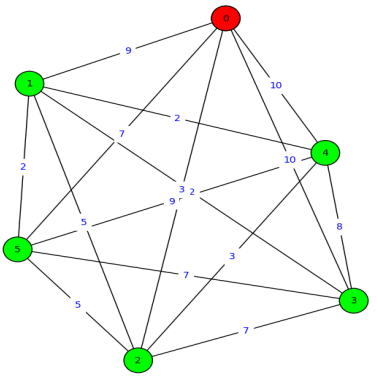
K_{6-5}



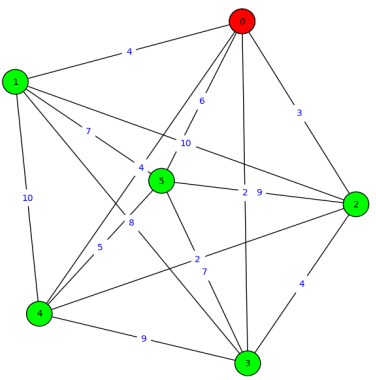
K_{6-6}



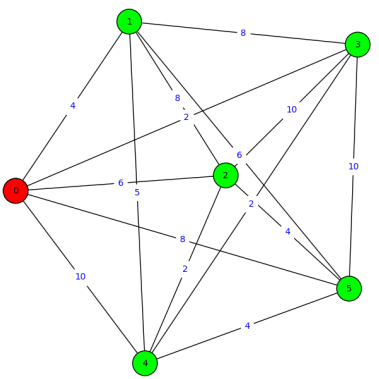
K_{6-7}



K_{6-8}



K_{6-9}



K_{6-10}

C C++ program to generate QUBO

```
// QUBO formulation for Bounded-Depth Steiner Tree
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <cmath>
#include <map>
#include <algorithm>

#define FIRST 0
#define MIDDLE 1
#define LAST 2

using namespace std;
void print_matrix(double **Q, int n);
void read_graph(int n, vector <pair<int,int> > &adjacent_list,
     vector <int> &setU, map<pair<int,int>,int> &weight);
const long generate_qubo(double **&Q, int node_size, vector<
    pair<int, int> > adjacent_list,
                         vector <int> setU, map<pair<int,int>,
    int> weight, int hop_constraint);

int main(int argc, char *argv[])
{
    int node_size=0,hop_constraint=0;
    double **Q;
    vector <pair<int,int> > adjacent_list;
    vector <int> setU;
    map<pair<int,int>,int> weight;
    if (argc != 2) cout << "Correct usage: " << argv[0] <<" <
    Hop constraint>" << endl;
    else hop_constraint = (int)strtol (argv[1], nullptr,10);
    cin>>node_size;

    read_graph(node_size,adjacent_list, setU, weight);

    const long N = generate_qubo(Q, node_size, adjacent_list,
    setU, weight, hop_constraint);
    cout<<N<<endl;
    print_matrix(Q,N);
    for (int i=0; i<N; i++)
        delete [] Q[i];
    delete [] Q;
    return 0;
}

```

```
const long generate_qubo(double **&Q, int node_size, vector<
    pair<int, int>> adjacent_list,
                         vector <int> setU, map<pair<int,int>,
    int> weight, int hop_constraint)
{
    // set nominated root as 0
    int nominated_root = setU[0];

    map<vector<int>,int> edge2matrix;

    // set V/U
    vector <int> setV_U;
    for(int i=0;i<node_size;i++){
        if(std::find(setU.begin(), setU.end(), i) == setU.end
    ()){
            setV_U.push_back(i);
        }
    }

    // init variables e_uv,i, and index of each variable in Q
    matrix
    int cnt=0;
    for (vector<pair<int,int> >::iterator iterator =
    adjacent_list.begin(); iterator != adjacent_list.end(); ++
    iterator) {
        if ((*iterator).second == nominated_root) continue;
        // variables x_{v0,u}_1
        if ((*iterator).first == nominated_root) {
            int myints[] = {iterator->first,iterator->second
    ,1};
            vector<int> var(myints, myints + sizeof(myints) /
    sizeof(int) );
            edge2matrix[var] = cnt;
            cnt++;
        } else // variables x_{u,v}_i, where 2 <= i <=
    hop_constraint
        {
            for(int i=2;i<=hop_constraint;i++)
            {
                int myints[] = {iterator->first,iterator->
    second,i};
                vector<int> var(myints, myints + sizeof(myints
    ) / sizeof(int) );
                edge2matrix[var] = cnt;
                cnt++;
            }
        }
    }

```

```
    // variables needed: N = 2(H-1)(|E| - Deg_G(v_0)) + Deg_G(
    v_0)
81  const long N = edge2matrix.size();

83  // initialize Q matrix
    Q = new double*[N];
85  for (int i=0; i<N; i++)
    {
87      Q[i] = new double[N];
        for (int j=0; j<N; j++) Q[i][j] = 0;
89  }

91  /********* F_{I,1} *********/
    for(int i=0; i<setU.size(); i++)
93  {
        int v = setU[i];
95      if (v == nominated_root) continue;
        for(map<vector<int>,int>::iterator iti=edge2matrix.
    begin(); iti!=edge2matrix.end(); ++iti)
97      {
            if(iti->first[MIDDLE] != v) continue;
99          int idx1 = iti->second;
            for(map<vector<int>,int>::iterator itj=edge2matrix
    .begin(); itj!=edge2matrix.end(); ++itj)
101         {
                if(itj->first[MIDDLE] != v) continue;
103             if(iti->second!=itj->second)
                {
105                 int idx2 = itj->second;
                    Q[idx1][idx2]=Q[idx1][idx2] + node_size;
107             }
                else
109             {
                    Q[idx1][idx1]=Q[idx1][idx1] - node_size;
111             }
            }
113     }
    }
115 if(DEBUG)
    {
117     printf("\n********* F_{I,1} *********\n");
        print_matrix(Q, N);
119 }

121 /********* F_{I,2} *********/
    int u,v;
123 for(int i = 0; i<setV_U.size(); i++){
        v=setV_U[i];
125     for(map<vector<int>,int>::iterator iti=edge2matrix.
    begin(); iti!=edge2matrix.end(); ++iti){
```

```
            if (iti->first[MIDDLE] != v ) continue;
            for(map<vector<int>,int>::iterator itj=edge2matrix.begin(); itj!=edge2matrix.end(); ++itj) {
                if (itj->first[MIDDLE] != v || iti->first[FIRST] >= itj->first[FIRST]) continue;
                Q[iti->second][itj->second]=Q[iti->second][itj->second] + node_size;
            }
        }
    }

    /********* F_{I,3} *********/
    for(v=0; v<node_size; v++){
        if (v == nominated_root) continue;
        for(map<vector<int>,int>::iterator iti=edge2matrix.begin(); iti!=edge2matrix.end(); ++iti)
        {
            if(iti->first[MIDDLE] != v) continue;
            if(iti->first[LAST] < 2) continue;
            u=iti->first[FIRST];
            int idx1 = iti->second;
            Q[idx1][idx1]++;
            for(map<vector<int>,int>::iterator itj=edge2matrix.begin(); itj!=edge2matrix.end(); ++itj)
            {
                if(itj->first[MIDDLE] != u) continue;
                if(itj->first[LAST] != (iti->first[LAST]-1)) continue;
                int idx2 = itj->second;
                Q[idx1][idx2]--;
            }
        }
    }

    /********* P_I *********/
    int maxWeight = 0;
    for(map<pair<int,int>,int>::iterator it=weight.begin(); it!=weight.end(); ++it)
        if (maxWeight < it->second) {maxWeight = it->second;}
    int P_I = (node_size-1) * maxWeight +1;
    for (int i=0; i<N; i++)
    {
        for (int j=0; j<N; j++)
            Q[i][j] = Q[i][j] * P_I;
    }

    /********* O_I *********/
    for(map<vector<int>,int>::iterator it=edge2matrix.begin(); it!=edge2matrix.end(); ++it)
    {
```

```
            u = it->first[FIRST];
            v = it->first[MIDDLE];
            Q[it->second][it->second] = Q[it->second][it->second]
        + weight[make_pair(u,v)];
        }
        return N;
}

void print_matrix(double **Q, const int n)
{
        for (int i=0; i<n; i++)
        {
            for (int j=0; j<n; j++)
                printf("%4d ", (int)Q[i][j]);
            printf("\n");
        }
}

void read_graph(const int n, vector <pair<int,int> > &
        adjacent_list, vector <int> &setU, map<pair<int,int>,int>
        &weight)
{
        vector <pair<int,int> > adjacent_tmp;
        map <pair<int,int>,int> adjacent;
        string line;
        int lineCnt=-1;
        for(int i=0;i<n+1;i++)
        {
            std::getline(cin, line);
            istringstream iss(line);
            int a;
            while (iss >> a)
            {
                adjacent[make_pair(lineCnt,a)] = 1;
                adjacent_tmp.push_back(make_pair(lineCnt,a));
            }
            lineCnt++;
        }

        lineCnt=0;
        for(int i=0;i<n;i++)
        {
            std::getline(cin, line);
            istringstream iss(line);
            int a;
            while (iss >> a) weight[adjacent_tmp[lineCnt++]] = a;
        }
        for(map <pair<int,int>,int>::iterator it=adjacent.begin();
         it!=adjacent.end(); ++it) {
            adjacent_list.push_back(it->first);
```

```
    }
    std::getline(cin, line);
    istringstream iss(line);
    int a;
    while (iss >> a) setU.push_back(a);
}
```

Listing 1.1: qubo_formulation.cpp