Journal of Membrane Computing
https://doi.org/10.1007/s41965-019-00030-1

REGULAR PAPER

Quantum solutions for densest *k*-subgraph problems

Cristian S. Calude[1] · **Michael J. Dinneen**[1] · **Richard Hua**[1]

Received: 11 November 2019 / Accepted: 24 December 2019

Abstract
In this paper, we present, for the first time, quantum annealing solutions for densest *k*-subgraph problems which have many applications in computational biology. Our solutions are formulated as solutions for quadratic unconstrained binary optimization (QUBO) and integer programming problems, proved to be equivalent with the densest *k*-subgraph problems and then tested on an D-Wave 2X machine. The QUBO formulations are optimal in terms of the number of logical qubits, but require the highest number of physical qubits after embedding. Experimental work reported here suggests that the D-Wave 2X model cannot handle problems of this difficulty. The next generation of D-wave hardware architecture—the Pegasus architecture—is much denser than the current one, so dense QUBOs will be easier to embed. The experimental work also suggests that the current built-in post-processing optimization method does not work very well for some problems and the default setting (post-processing optimization on) should be avoided (or at least tested before being turned on).

Keywords Densest *k*-subgraph problem · Quadratic unconstrained binary optimization and integer programming problems · Quantum annealing · D-Wave 2X

1 Introduction

Information in computational biology is increasingly represented with graphs, e.g., protein interactions, metabolic pathways, gene regulation, gene annotation, RNA-seq reads, etc. This means that many problems in processing biological information can be represented and solved using graph theory, see, for example, [9, 17, 30]. Also, many graph problems have been studied in membrane computing [28].

Adiabatic quantum computing (AQC) is a model of quantum computing based on the propensity of physical systems—classical or quantum—to minimize their free energy, specifically the free energy minimization in a quantum system. This model is implemented by the D-Wave series of quantum machines [6, 11, 26, 27].

A D-Wave quantum machine solves a generic quadratic unconstrained binary optimization (QUBO) problem, an NP-hard mathematical problem consisting in the minimization of a quadratic objective function:

$$z = \mathbf{x}^T Q \mathbf{x},$$

where $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$ is a n-vector of binary variables and Q is an upper triangular $n \times n$ matrix:

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

In this paper, we propose QUBO efficient formulations for problems involving finding various dense subgraphs of a given graph which have many applications in computational biology [9, 17, 30].

2 Notation

The set of positive reals is denoted by $\mathbb{R}^+$. Let $G = (V, E)$ be a simple undirected graph with vertices V and edges E. The number of vertices of a graph G is called the order of G and the number of edges is called the size of G. The order and size of a graph G are typically denoted by n and m, respectively.

✉ Cristian S. Calude
cristian@cs.auckland.ac.nz

Michael J. Dinneen
mjd@cs.auckland.ac.nz

Richard Hua
rwan074@aucklanduni.ac.nz

[1] School of Computer Science, University of Auckland, Auckland, New Zealand

Published online: 04 February 2020

Given a graph of order n, we denote the vertices by $V = \{v_0, v_1, \ldots, v_{n-1}\}$. An edge between vertices v_i and v_j is denoted by $\{v_i, v_j\}$ and for convenience, we assume without loss of generality that $i < j$. A vertex u in V is adjacent to a vertex v in V whenever $\{u, v\} \in E$. The set of adjacent vertices of u, denoted by $\mathrm{N}(u)$, is called the neighbors of u. In this paper, we only consider simple graphs which are graphs without self-loops (i.e., for all $v \in V$, $\{v, v\} \notin E$) or parallel edges.

Let $G = (V, E)$ be a graph and C a subset of V. The subgraph induced by C, denoted $G(C) = (V(C), E(C))$, is the graph having C as vertices and all the edges connecting pairs of vertices in C in E (i.e., $\{v_i, v_j\} \in E(C)$ if $\{v_i, v_j\} \in E$ and $\{v_i, v_j\} \subseteq C$). The set C is a clique if the induced graph $G(C)$ is complete [i.e., $G(C)$ has size $(|C| \cdot (|C| - 1)/2)$].

The density of the graph G is defined by:

$$\mathrm{DENS}(G) = \frac{|E|}{|V| \cdot (|V| - 1)/2}.$$

Given a constant $\gamma \in (0, 1]$, a subset C of vertices is called a γ-quasi-clique or, simply, a γ-clique, if C induces a subgraph with the edge density of at least γ. A γ-clique C is maximal if there is no other γ-clique C' which strictly contains C. A γ-clique C is maximum if there is no other γ-clique C' with higher density. Finding the maximum clique of a graph G is an NP-hard problem (the decision version is NP-complete) [4].

3 QUBO formulations for the (edge-weighted) densest *k*-subgraph problems

We first state formally the problem.

Densest k-subgraph problem:

Instance: A graph $G = (V, E)$, $n = |V|$, $m = |E|$, and positive integer $k \leq n$.

Question: Find a $V' \subseteq V$ with $|V'| = k$, such that $\mathrm{DENS}(G(V')) = \max\{\mathrm{DENS}(G(V'')) \mid V'' \subseteq V \text{ and } |V''| = k\}$.

The densest k-subgraph problem is a well-studied problem in complexity theory [24] which is related to several other important computational problems such as the maximum clique problem, which asks for the largest completely connected subgraph. The densest k-subgraph problem is at least at hard as the maximum clique problem and thus NP-hard for general graphs (i.e., the output is a k-clique of density 1 if it exists). Furthermore, the densest k-subgraph problem remains NP-hard even when the input is restricted to bipartite graphs [10] and planar graphs [23].

The edge-weighted densest k-subgraph problem, which is a generalization of the densest k-subgraph problem, has many applications in computational biology [9, 17, 30]. Here, every edge in the graph is assigned a real weight value by an edge-weight function W. Without loss of generality, we assume that the weights are normalized, so $W : E \to [-1, 1]$. The goal is to find a subset of vertices $V' \subseteq V$ that maximizes not the density of $G(V')$, but the sum of edge weights instead.

Edge-weighted densest k-subgraph problem:

Instance: A graph $G = (V, E)$, $n = |V|$, $m = |E|$, positive integer $k \leq n$ and an edge weight function $W : E \to [-1, 1]$.

Question: Find a $V' \subseteq V$ with $|V'| = k$, such that $\sum_{\{v_i, v_j\} \in E(V')} W(\{v_i, v_j\}) = \max\{\sum_{\{v_i, v_j\} \in E(V'')} W(\{v_i, v_j\}) \mid V'' \subseteq V \text{ and } |V''| = k\}$.

Note that the above two problems do not have unique solutions.

Now, we present a QUBO formulation for this edge-weighted problem. Given a graph $G = (V, E)$ with $|V| = n$ vertices, $|E| = m$ edges and a positive integer $k \leq n$, the QUBO formulation requires n binary variables labeled x_i for $0 \leq i < n$, one for each vertex $v_i \in V$. The binary variables are denoted by the binary vector $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$.

The objective function to be minimized is:

$$F(\mathbf{x}) = \alpha\left(\sum_{i=0}^{n-1} x_i - k\right)^2 - \sum_{\{i,j\} \in E} W(\{i,j\}) x_i x_j, \tag{1}$$

for sufficiently large $\alpha \in \mathbb{R}^+$.

Assume that x^* is an optimal solution with $\mathbf{x}^*$ and its corresponding variable assignment and take $V' = \{v_i \mid x_i = 1, 0 \leq i < n - 1\}$ as a solution to the densest k-subgraph problem. We define a 'decoder' function $D(\mathbf{x}) : \mathbb{Z}_2^n \to 2^V$ and $D(\mathbf{x}) = \{v_i \mid x_i = 1, 0 \leq i \leq n - 1\}$. Note $|D(x)| = k$.

The basic idea is that the term $\alpha\left(\sum_{i=0}^{n-1} x_i - k\right)^2$ in (1) ensures that exactly k vertices are chosen at the end and $-\sum_{\{i,j\} \in E} W(\{i,j\}) x_i x_j$ maximizes the weight sum of edges in the induced subgraph. We get the following:

Theorem 1 *Suppose $G = (V, E)$ is a graph of order n, size m, and a normalized edge-weight function $W : E \to [-1, 1]$. Assume that x^* and $\mathbf{x}^*$ are an optimal solution and its corresponding variable assignment of objective function (1), respectively. If $V' = D(\mathbf{x}^*)$, then $G(V')$ is the induced*

subgraph of k vertices with maximum $\sum_{\{i,j\}\in E(V')} W(\{i,j\})$. The total edge-weight sum of $G(V')$ is exactly $-x^*$.

Proof Let x^* and $\mathbf{x}^*$ be the optimal solution and its corresponding variable assignment in (1).

We first show that $\alpha\left(\sum_{i=0}^{n-1} x_i - k\right)^2 = 0$. Assume for the sake of a contradiction that $\left(\sum_{i=0}^{n-1} x_i - k\right)^2 \neq 0$. Then, we must have $\sum_{i=0}^{n-1} x_i \neq k$. Based on this, we show that one can construct a new variable assignment $\mathbf{y}$, such that $F(\mathbf{y}) < F(\mathbf{x}^*)$, which contradicts the optimality of $\mathbf{x}^*$.

There are two cases to analyze: (1) $\sum_{i=0}^{n-1} x_i = c > k$, and (2) $\sum_{i=0}^{n-1} x_i = c < k$.

In the former case, let $b = c - k$. As there are exactly c variables in $\mathbf{x}^*$ with value 1, we select any b of them, say $x_0, x_1, \ldots, x_{b-1}$ (note that we can always relabel all variables as we can permute the vertices of G if necessary). If we set all of these variables to 0, then $\sum_{i=0}^{n-1} x_i - k = 0$ which means that the value of $\alpha\left(\sum_{i=0}^{n-1} x_i - k\right)^2$ will be reduced by αb^2, so it will be possible to construct a new variable vector $\mathbf{y}$ with $F(\mathbf{y}) < F(\mathbf{x}^*)$. Now, for each vertex v_i where $0 \leq i < b$, the vertex v_i is connected to at most $c-1$ vertices in the induced subgraph $G(D(\mathbf{x}^*))$. Since $W(\{i,j\}) \leq 1$, setting the corresponding variables $x_0, \ldots, x_{b-1}$ to 0 will reduce the value of $\sum_{\{i,j\}\in E} W(\{i,j\})x_i x_j$ by at most $b(c-1)$. Accordingly, if $\alpha b^2 > b(c-1)$, then the overall value of $F(\mathbf{x})$ will decrease. Note that $\alpha b^2 > b(c-1)$ implies that $\alpha b > c - 1$ and since $c \leq n$ and $b > 1$, setting $\alpha = n$ is sufficient for this condition. By symmetry, we can also construct a new variable vector $\mathbf{y}$ for the latter case.

Finally, assume that $\alpha\left(\sum_{i=0}^{n-1} x_i - k\right)^2 = 0$, and hence, in this case, $|D(\mathbf{x}^*)| = k$ (i.e., the induced subgraph $G(D(\mathbf{x}^*))$ has exactly k vertices). Consequently, $-\sum_{\{i,j\}\in E} W(\{i,j\})x_i x_j$ maximizes the sum of edge weights in the induced subgraph $G(D(\mathbf{x}^*))$. □

Since the edge-weighted densest k-subgraph problem is the generalized version of the unweighted version, formula (1) can be used to solve the unweighted problem, as well. For fixed k, $k \cdot (k-1)/2$ is a constant, and hence, maximizing $\text{DENS}(G(V'))$ is the same as maximizing the number of edges in $G(V')$. Therefore, if $W(\{i,j\})$ is a constant function, then formula (1) solves the densest k-subgraph problem. Note that a negative edge weight in this case does not make much sense in terms of the problem, and therefore, we assume the edge-weight function is strictly positive. Consequently, we have:

Corollary 1 *Suppose that $G = (V, E)$ is a graph with order n, size m, and a constant edge-weight function $W : E \to (0, 1]$. Assume that x^* and $\mathbf{x}^*$ are an optimal solution and its corresponding variable assignment of objective function (1), respectively. If $V' = D(\mathbf{x}^*)$, then $G(V')$ is the induced subgraph of k vertices with maximum* $\text{DENS}(G(V'))$.

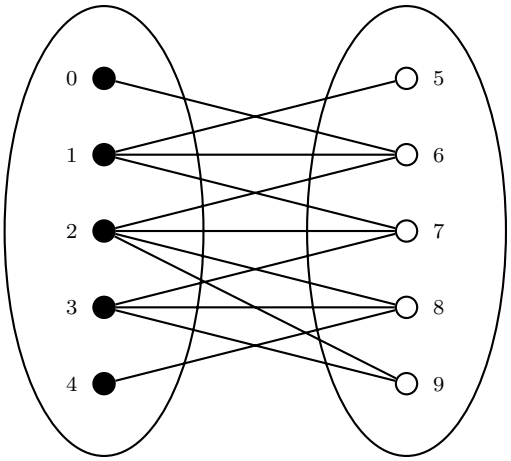


Fig. 1 Example of a bipartite graph; this graph was also used as an example in [30]

Proof Note that the proof of Theorem 1 only requires $W(\{i,j\}) \leq 1$, and hence, the result holds. □

3.1 An example

In this subsection, we give an example of a QUBO constructed by our method. Let $G = (V, E)$ be the graph shown in Fig. 1, where $V = \{0, 1, \ldots, 9\}$ and:

$$E = \{\{0,6\},\{1,5\},\{1,6\},\{1,7\},\{2,6\},\{2,7\},\{2,8\},\{2,9\},\{3,7\},\{3,8\},\{3,9\},\{4,8\}\}.$$

Suppose that $W : E \to (0, 1]$ a constant function where $W(e) = 1$ for all $e \in E$ and $k = 5$. Based on the objective function (1), with α set to $n = 10$, the complete QUBO is given in Table 1. A Python program to compute the QUBO can be found in Appendix 2. This QUBO has a unique optimal solution $\mathbf{x}^* = [0, 0, 1, 1, 0, 0, 0, 1, 1, 1]$ which corresponds to the vertex subset $S = \{2, 3, 7, 8, 9\}$ and it can be verified (e.g., by brute-force computation) that $G(S)$ is a subgraph of order 5 with maximum density.

4 Integer programming solution for the densest k-subgraph problem

In this section, we present a $0-1$ integer programming (IP) formulation for the densest k-subgraph problem. Various different formulations exist for (slight variations) of the problem and our solution is quite similar to the mixed

Table 1 QUBO matrix for graph of Fig. 1

Variables	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
x_0	− 90	20	20	20	20	20	19	20	20	20
x_1	0	− 90	20	20	20	19	19	19	20	20
x_2	0	0	− 90	20	20	20	19	19	19	19
x_3	0	0	0	− 90	20	20	20	19	19	19
x_4	0	0	0	0	− 90	20	20	20	19	20
x_5	0	0	0	0	0	− 90	20	20	20	20
x_6	0	0	0	0	0	0	− 90	20	20	20
x_7	0	0	0	0	0	0	0	− 90	20	20
x_8	0	0	0	0	0	0	0	0	− 90	20
x_9	0	0	0	0	0	0	0	0	0	− 90

integer programming formulation given in [2][1]. The difference is that the formulation in [2] assumes all edge weights are non-negative (hence, it does not compute the correct answer with negative edge weights).

Our formulation requires exactly $n + m$ binary variables: one for each vertex in the graph and one for each edge in the graph. We define x_i for each $v_i \in V$ and $y_{i,j}$ for each $\{i,j\} \in E$ where $x_i \in \{0,1\}$ and $y_{i,j} \in \{0,1\}$. If $x_i = 1$, then we choose v_i to be in the subset V'. For convenience, we denote the collection of variables x_i by $\mathbf{x}$. The complete formulation is as follows:

$$\text{Maximize} \sum_{\{i,j\} \in E} W(\{i,j\})\, y_{i,j}, \tag{2}$$

subject to

$$\sum_{0 \le i < n} x_i = k, \tag{3}$$

$$y_{i,j} \le x_i \text{ for each } \{i,j\} \in E, \tag{4}$$

$$y_{i,j} \le x_j \text{ for each } \{i,j\} \in E, \tag{5}$$

$$y_{i,j} + 1 - x_i - x_j \ge 0 \text{ for each } \{i,j\} \in E. \tag{6}$$

Note that the terms $W(\{i,j\})$ are constants, so the objective function (2) satisfies the conditions of a $0-1$ integer programming formulation.

Theorem 2 *Suppose that $G = (V,E)$ is a graph of order n, size m, and an edge-weight function $W : E \to [-1,1]$. Assume that x^* and $\mathbf{x}^*$ are an optimal solution and its corresponding variable assignment of objective function (2), respectively. If $V' = D(\mathbf{x}^*)$, then $G(V')$ is the induced subgraph of k vertices with maximum $\sum_{\{i,j\} \in E(V')} W(\{i,j\})$. The total edge-weight sum of $G(V')$ is exactly $-x^*$.*

Proof The constraint (3) ensures that exactly, k vertices are chosen in V', so it is sufficient to prove the theorem by showing that $y_{i,j} = 1$ if and only if $x_i = 1$ and $x_j = 1$. This is indeed the case, because the constraints (4) and (5) ensure the value of $y_{i,j}$ cannot be 1 unless both x_i and x_j have value of 1 and the constraints (6) enforce the condition that if both x_i and x_j have value 1, then $y_{i,j}$ must have value 1 as well.

To conclude, for every $V' \subseteq V$ with $|V'| = k$, for each edge $\{i,j\}$ in $E(G(V'))$, the corresponding variable $y_{i,j}$ will have a value of 1, and therefore, (2) will maximize the sum of edge weights in the induced subgraph. □

5 Specified subset edge-weighted densest *k*-subgraph problem

In gene annotation networks [30], the vertices and edges of the graphs represent DNA genes and features of some (group of) organisms; if the goal is to identify a specific set of genes or to find the set of genes related to a specified set of features, a general solution of the edge-weighted densest k-subgraph problem might not provide the relevant solution. These cases can be modeled by the following problem:

Specified subset edge-weighted densest k-subgraph problem:

Instance: A graph $G = (V,E)$, $n = |V|$, $m = |E|$, positive integer $k \le n$, a subset of vertices $S \subseteq V$, and an edge-weight function $W : E \to [-1,1]$.

Question: Find a $V' \subseteq V$ with $S \subseteq V'$ and $|V'| = k$, such that $\sum_{\{v_i,v_j\} \in E(V')} W(\{v_i,v_j\}) = \max\{\sum_{\{v_i,v_j\} \in E(V'')} W(\{v_i,v_j\}) \mid S \subseteq V'', V'' \subseteq V \text{ and } |V''| = k\}$.

[1] This particular formulation is actually a $0-1$ IP formulation as well.

Table 2 QUBO matrix for sample graph with fixed $S = \{0, 1, 2, 3\}$

Variables	x_4	x_5	x_6	x_7	x_8	x_9
x_4	− 10	20	20	20	19	20
x_5	0	− 11	20	20	20	20
x_6	0	0	− 13	20	20	20
x_7	0	0	0	− 13	20	20
x_8	0	0	0	0	− 12	20
x_9	0	0	0	0	0	− 12

In this variation of the edge-weighted densest k-subgraph problem, the input includes a subset S which the output V' must contain. The QUBO formulation given in Sect. 3 can be modified slightly to solve this problem. Assume that $S = \{v_0, v_1, \ldots, v_{c-1}\}$ (we can relabel vertices if needed). By setting all variables corresponding to S, that is, $\{x_0, x_1, \ldots, x_{c-1}\}$, to 1, we can guarantee that S is contained in the output subset V'. Note that the term k in Formula (1) has to be replaced by $k - c$ since we are now only choosing $k - c$ vertices from $V \setminus S$. The complete objective function is:

$$F(\mathbf{x}) = \alpha\left(\sum_{i=c}^{n-1} x_i - (k-c)\right)^2 - \sum_{\{i,j\}\in E \text{ and } i,j\notin S} W(\{i,j\})x_i x_j - \sum_{\{i,j\}\in E \text{ and } i\in S,\, j\notin S} W(\{i,j\})x_j. \tag{7}$$

Note that since we label the vertices in S by the first c integers, we will not have an edge $\{i,j\}$ where $i \notin S$ and $j \in S$ (recall that we assume that $i < j$). Equation (7) does not consider edges between vertices in S, that is, edges $\{i,j\}$ where $i,j \in S$. For a fixed set S, the edges between vertices in S are also fixed and so $\sum_{\{i,j\}\in E,\, i,j\in S} W(\{i,j\})$ will be constant, so it will have no effect on the minimization of $F(\mathbf{x})$.

Theorem 3 *Suppose that $G = (V, E)$ is a graph of order n, size m, S a subset of V, where $|S| = c < k$, a normalized edge-weight function $W : E \to [-1, 1]$. Assume that x^* and $\mathbf{x}^*$ are an optimal solution and its corresponding variable assignment of objective function (7), respectively. If $V' = D(\mathbf{x}^*) \cup S$, then $G(V')$ is the induced subgraph of k vertices with maximum $\sum_{\{i,j\}\in E(V')} W(\{i,j\})$. The total edge-weight sum of $G(V')$ is $-x^* + \sum_{\{i,j\}\in E,\, i,j\in S} W(\{i,j\})$.*

Theorem 3 follows directly from Theorem 1 and the construction of Equation (7) and so the proof is omitted.

5.1 The example revisited

Recall the example given in Sect. 3.1; let us consider the same graph here and let $S = \{0, 1, 2, 3\}$. Once again, the goal is to find a maximum edge-weighted induced subgraph of order 5, and the subgraph has to include vertices 0, 1, 2 and 3. All other parameters are the same as the example given in Sect. 3.1. Based on Formula (7), the complete QUBO is given in Table 2. Note that S only contains vertices from one partition of the bipartite graph and we are only selecting one additional vertex, so it is quite obvious that the new vertex has to be chosen from the other partition (i.e., $\{5, 6, 7, 8, 9\}$). Choosing either vertex 6 or 7 in this case would produce the maximal number of edges and it can be verified computationally that both $\mathbf{x}_1^* = [0, 0, 1, 0, 0, 0]$ and $\mathbf{x}_2^* = [0, 0, -0, 1, -0, 0]$, which correspond to the selection of vertices 6 and 7, respectively, are optimal solutions of the QUBO given in Table 2.

6 Experimental work

We conducted several experiments to empirically measure the viability of the QUBO formulation of the densest k-subgraph problem given in Sect. 3 on a D-Wave 2X quantum annealer. We will first provide an overview of how the experiments were set up followed by a presentation and discussion of the results. Some of our programs and scripts given in the appendix use external packages such as Sage Mathematics [31], NetworkX [18] and the D-Wave System API [15]. We also assume that all graphs are represented using the standard adjacency list format (for example, see Sect. 3).

6.1 Experiment setup and test cases

As mentioned in Sect. 3, the densest k-subgraph problem remains NP-hard even when the input is restricted to bipartite graphs. Because of the importance of bipartite graphs in computational biology (e.g., see [30]), we generated random test cases which are all bipartite graphs. There are several different ways to generate random graphs: in this paper, we used the random model from [3]. In this model, each edge is chosen independently with probability p for $0 < p < 1$. This means that if $G = (V, E)$ is a bipartite graph and $V = V_1 \cup V_2$ is the vertex partition, then for $v_i \in V_1$ and $v_2 \in V_2$, the probability of $\{v_i, v_j\} \in E$ is p.

The NetworkX software package provides a function to generate these test cases, see Script 1. The function takes three parameters: the number of vertices in both partitions and an edge probability p. Both partitions have the same size, so if the order of the graph is n, each partition has exactly $n/2$ vertices (assuming n is even). The edge probability p is set to 0.8, since denser instances of the problem are harder in general. We also require the graph to be connected as otherwise it can be reduced to multiple smaller instances of the densest k-subgraph problem.

We randomly generated 20 bipartite graphs of order 30. See Sect. 5 for an example drawing of graph 1 and the auxiliary data of [8] for further details of all test cases. These graphs have about the largest instances that can be solved on a D-Wave 2X. We also set k to $n/2 = 15$, because these are the hardest cases for the densest k-subgraph problem, since $\binom{n}{k}$ yields the highest number of different combinations when $k = n/2$. Each graph is also associated with a set of randomly generated edge weights (in range [0, 1)) using the built-in pseudo-random number generator of Python (see Script 1). The weights are rounded to 2 decimal place due to the limited precision of the D-Wave hardware.

Since the quantum annealers manufactured by D-Wave do not support arbitrary qubit interactions, a given QUBO instance has to be 'embedded' in the hardware structure of the quantum device before the device can be used to solve it. This embedding process is highly dependent on the hardware structure. The family of D-Wave computers uses the Chimera graphs which are $M \times N$ blocks of interconnected complete bipartite graphs $K_{L,L}$. The specific model that we have used (D-Wave 2X) has $M = N = 12$ and $L = 4$; more details of the hardware structure and the embedding process can be found in [1]. In general, the denser the QUBO instance is, the more physical qubits are required to embed the instance. Previous experimental work such as [1, 7, 16] and theoretical results [5] suggest that the embedding of a QUBO instance of n variables requires $O(n^2)$ physical qubits. This is especially true here, since formula (1) requires every pair of variables to interact, so the variable interactions form a complete graph of order n (K_n) which is the hardest case possible (from the perspective of embedding). In theory, one can embed complete graphs up to order 48 in the D-Wave 2X model and a polynomial time algorithm to generate such embeddings is given in [5]. Moreover, the particular D-Wave 2X which we worked with has 54 inactive qubits and extra faulty couplers in-between active qubits, so the aforementioned algorithm [5] is not applicable in this case here. Consequently, we have used the heuristic algorithm provided by the D-Wave software package [15] to compute the embeddings for our test cases. Since all of our test cases have the same interaction structure, the same embedding was used for all test cases—this also saved considerable classical computer time, since computing different embeddings is time consuming. The QUBO instance has to be converted to its equivalent Ising form, since the D-Wave model works only with problems in Ising formulation; the D-Wave software package [15] has functions that support this transformation process. For more details, see [22] and [11].

6.2 Parameter settings

After embedding the QUBO instance in the D-Wave hardware, the annealing cycle begins; that is, the quantum annealing is used to solve the embedded problem instance. The D-Wave 2X system provides many different options and settings (see [12]) and it is difficult to determine the optimal settings for a particular problem. For our experiments, we used the same parameter settings as described in [20] which follow the standard guidelines in the manuals such as [11] and [12]. One difference between our settings and the ones used in [20] is that we repeated the experiment twice: the first time with the post-processing optimization option turned on (default) and the second time with the option turned off, see Sect. 6.4 for details. The process of quantum annealing is probabilistic, and in general, it is not possible to calculate the probability of obtaining an optimal solution, so experimental work is used to empirically approximate this probability via repetitions, see [1]). Finally, we set the annealing cycle to repeat a total of 1000 times for each test case which generates 1000 samples (not all samples are unique). Since the D-Wave hardware can sometimes be inconsistent (e.g., due to the machine maintenance and recalibration cycles), we repeated the entire experiment three times and all the results are given in Tables 5 and 6. See Sect. 6.4 for more details.

6.3 Classical solvers

We have also implemented several classical algorithms for the densest k-subgraph problem to empirically compare the performance of the D-Wave 2X. These includes a Sage program that solves the IP formulation (2) as well as a straightforward randomized algorithm. The Sage program is given in Appendices 3 and 4. The Sage program uses the GLPK solver which is an exact solver for IP. The IP formulation (2) requires $O(n^2)$ variables, since there are at most $O(n^2)$ edges in a graph. The largest instance of the problem solved with the script in Appendix 3 was a graph with 40 vertices and more than 1000 edges which took more than 30 min, which is quite a long time, on a 8-core Intel(R) Core(TM) i7-4790K CPU running Fedora 29 Another reason for choosing test cases of order 30 was the difficulty to verify the correctness quantum solutions with classical computations.

Appendix 4 describes the simple randomized algorithm. At each iteration, we generated a random subset of V of size 15 and we computed the total edge-weight sum of the

induced subgraph. The final returned value is the vertex subset that yields the highest total edge-weight sum.

6.4 Experiment results and discussion

We also used the Sage solver to verify that all test cases had unique optimal solutions. In theory, the probability of a test case having more than one optimal solutions is very low but not impossible (i.e., for example a test case with at least two subgraphs of order 15 with the same edge-weight sum). Since having a unique optimal solution would make the analysis in Sect. 6.4 a lot simpler, we modified that the Sage Program 3 to verify the optimal solution of each test case is unique. The GLPK solver itself does not support the function of enumerating all optimal solutions and so we modified the program. Let $G = (V, E)$ be a graph and suppose that we have used Script 3 to find (one of) its optimal solution; let W' and $V' = \{v_0, v_1, \ldots, v_{14}\}$ be the optimal objective value (edge-weight sum of the induced subgraph) and set of vertices of the optimal solution. If we add another constraint of the form:

$$\sum_{v_i \in V'} x_{v_i} \leq k - 1$$

to the IP formula (2), then the GLPK solver is now not allowed to select V' as the vertex set again. Hence, if we now compute the optimal objective value W'' and compare with W', the optimal solution is unique if and only if $W'' \neq W'$. We verified via this approach that all test cases had unique optimal solutions.

Table 3 is the optimal solution of the IP formulations computed by Sage. Table 4 presents the best results obtained with the randomized algorithm (Program 4) after 1000 iterations. We have also measured the quality of the best answer from the randomized algorithm by two additional metrics. Let $G' = (V', E')$ and $G'' = (V'', E'')$ be the optimal solution computed by Sage (i.e., induced subgraph with maximum edge-weight sum) and the best answer produced by the randomized algorithm, respectively. The optimal solution ratio is calculated as $\frac{\sum_{e \in E''} W(e)}{\sum_{e \in E'} W(e)}$, so it shows how close the weighted edge sum produced by the random algorithm is to the actual optimal answer. The other metric is the Jaccard similarity index, a commonly used measurement in statistics that measures how close (or similar) two given sets are. In our case, the Jaccard index is calculated as $\frac{|V' \cap V''|}{|V' \cup V''|}$ and so it measures how close the set of vertices found by the randomized algorithm is to the actual optimal set.

As can be seen in this table, the randomized algorithm consistently computes solutions that are about 85% of an optimal value, a rather surprising result considering the simplicity the algorithm. In theory, the expected number of vertices in a random subgraph of k vertices is $\binom{k}{2}\frac{2m}{n(n-1)}$, where

Table 3 Sage optimal solutions

Test cases	Maximum weight sum
1	33.89
2	34.22
3	31.41
4	31.49
5	30.07
6	34.61
7	33.24
8	34.19
9	35.61
10	31.75
11	30.12
12	31.27
13	33.95
14	32.17
15	32.65
16	31.69
17	34.91
18	31.21
19	32.39
20	33.07

Table 4 Randomized algorithm solutions

Test cases	Best answer	Ratio over optimal solution	Jaccard index
1	28.83	0.85	0.43
2	29.76	0.87	0.58
3	27.25	0.87	0.67
4	27.73	0.88	0.5
5	26.2	0.87	0.25
6	30.16	0.87	0.58
7	29.48	0.89	0.5
8	32.02	0.94	0.67
9	29.68	0.83	0.5
10	27.17	0.86	0.5
11	26.72	0.89	0.43
12	27.44	0.88	0.36
13	29.29	0.86	0.58
14	28.2	0.88	0.36
15	27.94	0.86	0.58
16	27.17	0.86	0.67
17	29.76	0.85	0.58
18	26.31	0.84	0.5
19	29.17	0.9	0.5
20	30.02	0.91	0.58

m and n are the size and order of the graph respectively. Let i and j be the ith and jth random vertex choices. Since there are two ways to pick an edge $\{u, v\} \in E$ (i.e., $i = u$ and $j = v$ or $i = v$ and $j = u$), the chance of an edge being chosen is $2m/n(n-1)$. Furthermore, since there are $\binom{k}{2}$ different combinations, the assertion follows. For the test graphs (Program 1), the expected number of edges in each graph is exactly $15 \times 15 \times 0.8 = 180$ (since there are 15 vertices in each partition and edge probability is 0.8). Together with the fact that we have used normalized edge weights in range (0, 1], the expected edge-weight sum in a random graph is $\binom{15}{2}\frac{180}{30 \cdot 29} \approx 21.72$ which is already about 60% of the true optimal values. The average Jaccard index value for all the test cases is 52% so the best vertex set is somewhat different from the optimal set. This is expected as well, since the edge weights are uniformly distributed so there should be many induced subgraphs with similar edge-weight sum.

Recall that for the D-Wave 2X experiment, we repeat the annealing cycle 1000 times for each test case and, therefore, generating 1000 samples per test case. In general, there is no guarantee that all 1000 samples correspond to valid solutions, so we discarded samples which do not correspond to exactly 15 vertices. The valid answer with the highest edge-weight sum for each test case is given in Tables 5 and 6. We have also computed the optimal solution rate and the Jaccard similarity value for the best answer, and we repeated the entire experiment three times. For example, the test case labeled 1.2 in Table 5 is the result on graph 1 (see Sect. 5 for its structure) of the second repetition of the experiment with post-processing optimization turned off.

As can be seen in Tables 5 and 6, the answers obtained with quantum annealing are significantly worse than the answers generated by the randomized algorithm: the average optimal solution ratio is 41% and 71% with and without post-processing optimization respectively. There is also a decrease in the Jaccard similarity value when compared with the randomized algorithm; the average is 25% and 37% with and without post-processing settings, respectively.

We suspect that the main reason for the poor performance of D-Wave is due to the quality of the embedding used. To test this hypothesis, we define the average map size of an embedding as $\frac{\text{no. of physical qubits}}{\text{no. of logical qubits}}$ (i.e., how many physical qubits does each logical qubit corresponds to), a commonly used metric in measuring the quality of embeddings. Previous experimental studies such as [19, 29][2] have shown that embeddings with shorter and uniform map size provide a better solution quality. The particular embedding that we have computed uses 440 physical qubits, so the average map

[2] These publications refer to the map size as 'chain length' which could be somewhat misleading since the set of physical qubits do not necessarily form a path.

Table 5 D-Wave solutions with no post-processing

Test cases	Best answer	Ratio over optimal solution	Jaccard index
1.1	20.55	0.61	0.25
1.2	24.07	0.71	0.5
1.3	28.2	0.83	0.5
2.1	25.37	0.74	0.43
2.2	26.25	0.77	0.5
2.3	23.66	0.69	0.43
3.1	21.59	0.69	0.36
3.2	23.33	0.74	0.36
3.3	23.73	0.76	0.3
4.1	25.51	0.81	0.3
4.2	24.29	0.77	0.25
4.3	24.03	0.76	0.3
5.1	22.05	0.73	0.25
5.2	23.5	0.78	0.25
5.3	22.64	0.75	0.3
6.1	24.46	0.71	0.43
6.2	24.53	0.71	0.43
6.3	22.75	0.66	0.43
7.1	20.43	0.61	0.3
7.2	22.65	0.68	0.36
7.3	24.13	0.73	0.43
8.1	21.59	0.63	0.3
8.2	25.45	0.74	0.43
8.3	25.89	0.76	0.3
9.1	24.25	0.68	0.25
9.2	27.89	0.78	0.5
9.3	28.29	0.79	0.43
10.1	24.29	0.77	0.3
10.2	24.98	0.79	0.43
10.3	22.35	0.7	0.36
11.1	0.0	0.0	0.0
11.2	20.43	0.68	0.5
11.3	23.35	0.78	0.5
12.1	22.61	0.72	0.25
12.2	23.07	0.74	0.43
12.3	23.52	0.75	0.43
13.1	26.39	0.78	0.3
13.2	28.44	0.84	0.43
13.3	27.05	0.8	0.58
14.1	0.0	0.0	0.0
14.2	24.94	0.78	0.58
14.3	26.64	0.83	0.3
15.1	23.27	0.71	0.36
15.2	26.51	0.81	0.43
15.3	26.35	0.81	0.36
16.1	0.0	0.0	0.0
16.2	26.22	0.83	0.5
16.3	24.88	0.79	0.5
17.1	19.95	0.57	0.36

Table 5 (continued)

Test cases	Best answer	Ratio over optimal solution	Jaccard index
17.2	27.9	0.8	0.5
17.3	27.69	0.79	0.5
18.1	22.59	0.72	0.43
18.2	21.84	0.7	0.43
18.3	22.54	0.72	0.43
19.1	24.62	0.76	0.36
19.2	28.09	0.87	0.43
19.3	27.39	0.85	0.2
20.1	24.63	0.74	0.43
20.2	24.83	0.75	0.36
20.3	24.66	0.75	0.3

size is $\frac{440}{30} \approx 14.7$, a relatively high value. Furthermore, the minimum and maximum map sizes are 8 and 21, respectively, which is far from uniform. The study in [29] benchmarked the effects of large map size (long chains) on a D-Wave 2 quantum computer (the predecessor of the D-Wave 2X), and experimentally estimated the probability of obtaining the correct answer with map size of 14 to be less than 60%: in our test cases, we have 30 'chains', and hence, the probability of all chains working correctly is less than 0.0000221% (if we assume that each chain operates independently).

Another observation is the fact that the quality of the results is clearly worse when the post-processing optimization option is turned on. There is no test case where turning on the post-processing optimization option actually improved the optimal solution rate. Since the samples (with optimization on and off) are from two different runs, it is possible that the samples from the runs when the optimization option is turned on were so bad that it was beyond "repair" by the classical algorithm (due to an unknown reason): however, his hypothesis is highly unlikely, since the same trend exists in every test case. It seems odd that the post-processing optimization produces worse results for our test cases. Indeed, the post-processing optimization algorithm is just a classical algorithm running after the annealing cycles to improve the results, so one should expect that the final results to be at least as good as the results with the option turned off. One possible explanation is that our test cases are the hardest cases for this classical algorithm. The exact details of the post-processing optimization algorithm are not available, but a high-level (rather vague) pseudocode of the algorithm is given in [13]. The user manual [13] states that the algorithm is based on [25] and [21]. Both [13] and [25] assume and rely on the fact that the logical interaction graph (i.e., QUBO matrix) is rather sparse, so it

Table 6 D-Wave solutions with post-processing

Test cases	Best answer	Ratio over optimal solution	Jaccard index
1.1	0.0	0.0	0.0
1.2	22.55	0.67	0.43
1.3	28.2	0.83	0.5
2.1	0.0	0.0	0.0
2.2	22.19	0.65	0.25
2.3	23.66	0.69	0.43
3.1	0.0	0.0	0.0
3.2	21.36	0.68	0.36
3.3	23.73	0.76	0.3
4.1	0.0	0.0	0.0
4.2	22.51	0.71	0.3
4.3	24.03	0.76	0.3
5.1	0.0	0.0	0.0
5.2	17.97	0.6	0.36
5.3	22.54	0.75	0.36
6.1	0.0	0.0	0.0
6.2	24.53	0.71	0.43
6.3	22.75	0.66	0.43
7.1	0.0	0.0	0.0
7.2	21.65	0.65	0.36
7.3	22.3	0.67	0.43
8.1	0.0	0.0	0.0
8.2	25.45	0.74	0.43
8.3	23.48	0.69	0.3
9.1	0.0	0.0	0.0
9.2	25.38	0.71	0.43
9.3	27.64	0.78	0.36
10.1	0.0	0.0	0.0
10.2	16.36	0.52	0.36
10.3	22.28	0.7	0.36
11.1	0.0	0.0	0.0
11.2	19.75	0.66	0.5
11.3	25.23	0.84	0.58
12.1	0.0	0.0	0.0
12.2	0.0	0.0	0.0
12.3	23.52	0.75	0.43
13.1	0.0	0.0	0.0
13.2	28.44	0.84	0.43
13.3	23.1	0.68	0.36
14.1	0.0	0.0	0.0
14.2	23.95	0.74	0.58
14.3	26.64	0.83	0.3
15.1	0.0	0.0	0.0
15.2	22.79	0.7	0.25
15.3	21.21	0.65	0.3
16.1	0.0	0.0	0.0
16.2	24.54	0.77	0.5
16.3	24.88	0.79	0.5
17.1	0.0	0.0	0.0

Table 6 (continued)

Test cases	Best answer	Ratio over optimal solution	Jaccard index
17.2	24.83	0.71	0.5
17.3	23.43	0.67	0.36
18.1	0.0	0.0	0.0
18.2	20.58	0.66	0.36
18.3	21.19	0.68	0.36
19.1	0.0	0.0	0.0
19.2	24.43	0.75	0.2
19.3	27.39	0.85	0.2
20.1	0.0	0.0	0.0
20.2	24.51	0.74	0.43
20.3	23.95	0.72	0.3

has low tree-width, a condition which is not satisfied by our test cases. Based on formula (1), our QUBO are all complete graphs and so always have tree-width $n-1$. Our test cases seem to be somewhat of the type of an 'edge case' to the post-processing optimization algorithm which is causing this poor performance (or maybe even working incorrectly). Finally, we stress that we could not determine the exact reason, since the exact algorithm is not available.

7 Conclusion and related problems

In this paper, we have presented for the first time QUBO and IP formulations for the (edge-weighted) densest k-subgraph problem as well as a QUBO formulation for the specified subset edge-weighted densest k-subgraph problem. Given a graph $G = (V, E)$ of order n, there are 2^n possible subgraphs and so one needs at least $O(\lg 2^n) = O(n)$ binary variables to represent all possible solutions. Our QUBO formulations are optimal in terms of the number of logical qubits required, which is not typical for QUBO formulations (for example, see [1, 7]). However, the performance of D-Wave is heavily affected by the high map size of logical qubits and our constructions require the highest number of physical qubits after embedding: it seems that the D-Wave 2X model cannot handle problems of this difficulty. D-Wave next generation of hardware architecture, known as the Pegasus architecture [14], is much denser (the degree of each vertex is more than doubled) than the current Chimera graph, so dense QUBOs will be easier to embed. This feature should mitigate some of the negative effects which we are seeing here with the embedding.

We have also shown (via experimental work) that the current built-in post-processing optimization method does not work very well for some problems and the default setting (post-processing optimization on) should be avoided (or at least tested before being turned on).

Finally, we note that there is another variation of the densest k-subgraph problem called the distance restricted densest k-subgraph problem [30]. The input of the problem has an additional distance restriction function $D : V \times V \to \mathbb{N}$ and a new constraint that for all pairs of vertices u and v in the selected subset S, the distance between u and v in $G(V)$ has to be at most $D((u, v))$. This distance restricted version of the problem is obviously NP-hard if an arbitrary function D is allowed. It is more difficult to develop practical QUBO or IP formulations for this distance restriction version of the problem, since the standard methods do not seem to easily encode the distance metric constraint. Indeed, the distance between u and v can only be determined after the set S is fixed and it is not practical to consider all possible paths between u and v in G in general. We plan to study this problem in more detail in the future.

Acknowledgements This work was supported in part by the Quantum Computing Research Initiatives at Lockheed Martin. We thank the anonymous referees for useful comments which improved the presentation.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix 1: Python script to generate random bipartite graphs

```
#!usr/bin/env python2

import sys, random
from networkx.algorithms import bipartite as bi
import networkx as nx

n = int(sys.argv[1].strip())
n_1 = n/2
n_2 = n - n_1

while True:
    G = bi.random_graph(n_1, n_2, 0.8)
    if nx.is_connected(G):
        break

print G.order()
for i in G.nodes():
    for j in G.neighbors(i):
        print j,
    print

edge_weight_dict = {}
for (u,v) in G.edges():
    w = round(random.random(),2)
    edge_weight_dict[u,v] = w
    edge_weight_dict[v,u] = w
print edge_weight_dict
```

generate_test_cases.tex

Appendix 2: Python script to generate QUBOs

```
#!/usr/bin/env python2
#usage: generate_QUBO.py < alist

import sys, math
import networkx as nx

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

def generate_qubo(G, W, k):
    n = G.order()
    a = n
    Q = {}
    e = {}

    for i in range(n):
        for j in range(n):
            Q[i,j] = a

    for i in range(n):
        Q[i,i] -= 2*a*k

    for (i,j) in G.edges():
        Q[i,j] -= W[i,j]

    # upper-triangulization
    for i in range(n):
        for j in range(n):
            if i > j:
                Q[j,i] += Q[i,j]
                Q[i,j] = 0

    # output QUBO
    for i in range(n):
        for j in range(n):
            print Q[i,j],
        print

G = read_graph()
n = G.order()
W = eval(sys.stdin.readline().strip())
generate_qubo(G, W, n/2)
```

generate_QUBO.tex

Appendix 3: Sage script to solve the edge-weighted densest *k*-subgraph problem

```
#!/usr/bin/env sage
#usage: sage dense_subgraph_IP.py < alist

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()
k = n/2

p=MixedIntegerLinearProgram(solver="GLPK", maximization=True)
x=p.new_variable(binary=True)
#p.set_binary(x)
c = x[0]
for i in range(1,n):
    c = c + x[i]
p.add_constraint(c == k)
for (i,j) in G.edges():
    c = x[i,j] + 1 - x[i] - x[j]
    p.add_constraint(c >= 0)
    p.add_constraint(x[i,j] <= x[i])
    p.add_constraint(x[i,j] <= x[j])

W = eval(sys.stdin.readline().strip())
#print W
p.set_objective(sum(W[i,j] * x[i,j] for (i,j) in G.edges()))
#p.show()
try:
    sz=p.solve()
except sage.numerical.mip.MIPSolverException as e:
    print e
else:
    pass
    opt_weight = p.get_objective_value()
    var_value = p.get_values(x)

print opt_weight
print var_value
```

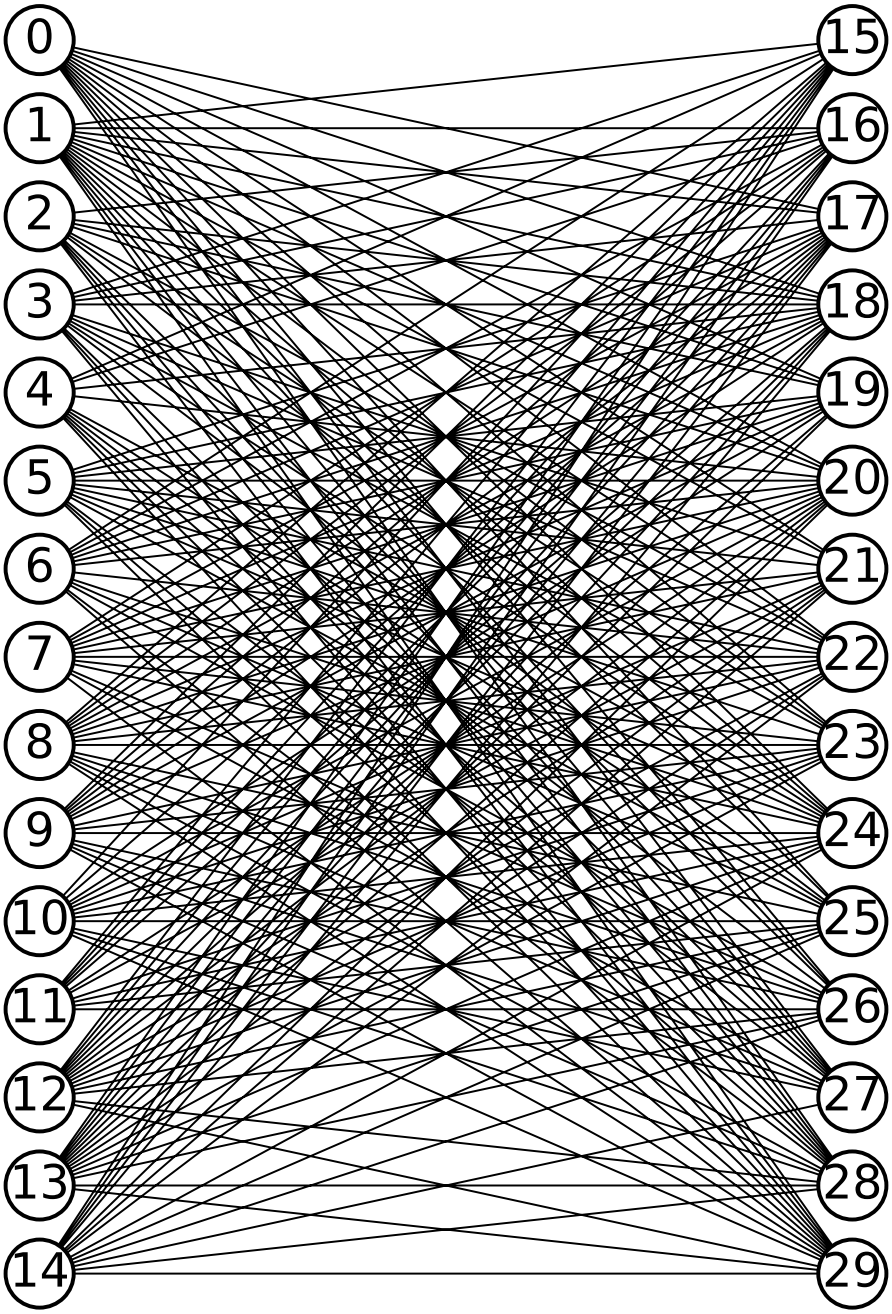
dense_subgraph_IP.tex

Appendix 4: Randomized algorithm to solve the edge-weighted densest *k*-subgraph problem

```
#!/usr/bin/env python2

import random, sys
import 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()

number_of_repeat = 1000
best_weight_sum = 0
best_vertex_set = []
n = G.order()
W = eval(sys.stdin.readline().strip())

for i in range(number_of_repeat):
    S = random.sample(range(n),n/2)
    induced_subgraph = G.subgraph(S)
    weight_sum = 0
    for (j,k) in induced_subgraph.edges():
        weight_sum += W[j,k]
    if weight_sum > best_weight_sum:
        best_weight_sum = weight_sum
        best_vertex_set = S

print best_vertex_set
print best_weight_sum
```

dense_subgraph_random.tex

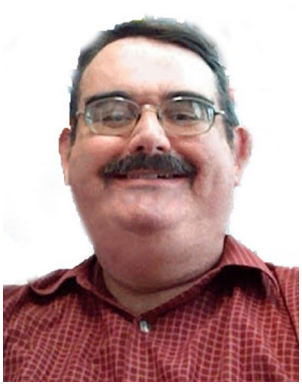
Appendix 5: Drawing of a sample test graph



References

1. Abbott, A. A., Calude, C. S., Dinneen, M. J., & Hua, R. (2018). A hybrid quantum-classical paradigm to mitigate embedding costs in quantum annealing. *CDMTCS Report Series* 520.
2. Billionnet, A. (2005). Different formulations for solving the heaviest k-subgraph problem. *INFOR: Information Systems and Operational Research*, *43*(3), 171–186.
3. Bollobás, B. (1981). Degree sequences of random graphs. *Discrete Mathematics*, *33*(1), 1–19.
4. Bomze, I. M., Budinich, M., Pardalos, P., & Pelillo, M. (1999). Handbook of combinatorial optimization. In D.-Z. Du & P. M. Pardalos (Eds.), *chap. The maximum clique problem* (pp. 1–74). Dordrecht: Kluwer Academic Publishers.
5. Boothby, T., King, A. D., & Roy, A. (2016). Fast clique minor generation in Chimera quit connectivity graphs. *Quantum Information Processing*, *15*(1), 495–508.
6. Calude, C. S., Calude, E., & Dinneen, M. J. (2015). Adiabatic quantum computing challenges. *ACM SIGACT News*, *46*(1), 40–61. https://doi.org/10.1145/2744447.2744459.
7. Calude, C. S., Dinneen, M. J., & Hua, R. (2017). QUBO formulations for the graph isomorphism problem and related problems. *Theoretical Computer Science*. https://doi.org/10.1016/j.tcs.2017.04.016.
8. Calude, C. S., Dinneen, M. J., & Hua, R. (2019). Quantum solutions for densest *k*-subgraph problems. Report CDMTCS-540, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand. https://www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports. Accessed Dec 2019.
9. Canzar, S., Andreotti, S., Weese, D., Reinert, K., & Klau, G. W. (2016). CIDANE: Comprehensive isoform discovery and abundance estimation. *Genome Biology*, *17*, 16.
10. Corneil, D. G., & Perl, Y. (1984). Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, *9*(1), 27–39.
11. D-Wave Systems. (2018). D-Wave Problem-Solving Handbook. User Manual. https://docs.dwavesys.com/docs/latest/doc_handbook.html. Accessed Dec 2019.
12. D-Wave Systems. (2018). D-Wave Solver Properties and Parameters Reference. User Manual. https://docs.dwavesys.com/docs/latest/doc_solver_ref.html. Accessed Dec 2019.
13. D-Wave Systems. (2018). Postprocessing Methods on D-Wave Systems. User Manual. https://docs.dwavesys.com/docs/latest/doc_post-processing.html. Accessed Dec 2019.
14. D-Wave Systems. (2019). Next-Generation Topology of D-Wave Quantum Processors. https://www.dwavesys.com/sites/default/files/14-1026A- C_Next-Generation-Topology-of-DW-Quantum-Processors.pdf. Accessed Dec 2019.
15. D-Wave Systems, Inc. (2018). Developer guide for Python. Technical Report Release 2.4 09-1024A-K.
16. Dinneen, M. J., & Hua, R. (2017). Formulating graph covering problems for adiabatic quantum computers. In: *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '17 (pp. 18:1–18:10). New York, NY, USA: ACM. https://doi.org/10.1145/3014812.3014830.
17. Fratkin, E., Naughton, B. T., Brutlag, D. L., & Batzoglou, S. (2006). Motifcut: Regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, *22*(14), e150–1e57.
18. Hagberg, A., Schult, D., & Swart, P. (2019). *NetworkX. Software for Complex Networks*
19. Hamerly, R., Inagaki, T., McMahon, P. L., Venturelli, D., Marandi, A., Onodera, T., et al. (2019). Experimental investigation of performance differences between coherent ising machines and a quantum annealer. *Science Advances*, *5*(5), eaau0823.
20. Hua, R., & Dinneen, M. J. (2020). Improved QUBO formulation of the graph isomorphism problem. *SN Computer Science*, *1*(1), 19.
21. Jensen, F. V., Lauritzen, S. L., & Olesen, K. G. (1990). Bayesian updating incausal probabilistic networks by local computations. *Computational Statistics Quarterly*, *4*, 269–282.
22. Kadowaki, T., & Nishimori, H. (1998). Quantum annealing in the transverse ising model. *Physical Review E*, *58*(5), 5355.
23. Keil, J. M., & Brecht, T. B. (1991). The complexity of clustering in planar graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, *9*, 155–159.
24. Khuller, S., Saha, B. (2009) On finding dense subgraphs. In: *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I, ICALP '09* (pp. 597–608). Berlin: Springer. https://doi.org/10.1007/978-3-642-02927-1_50.
25. Markowitz, H. M. (1957). The elimination form of the inverse and its application to linear programming. *Management Science*, *3*(3), 255–269.
26. McGeoch, C. C. (2014). *Adiabatic quantum computation and quantum annealing. Theory and practice*. San Rafael: Morgan & Claypool Publishers.
27. McGeoch, C. C., Harris, R., Reinhardt, S. P., & Bunyk, P. I. (2019). Practical annealing-based quantum computing. *Computer*, *52*, 38–46.
28. Paun, G., Rozenberg, G., & Salomaa, A. (2010). *The oxford handbook of membrane computing*. New York: Oxford University Press Inc.
29. Pudenz, K. L., Albash, T., & Lidar, D. A. (2014). Error-corrected quantum annealing with hundreds of qubits. *Nature Communications*, *5*, 3243.
30. Saha, B., Hoch, A., Khuller, S., Raschid, L., & Zhang, X. N. (2010). Dense subgraphs with restrictions and applications to gene annotation graphs. In B. Berger (Ed.), *Research in computational molecular biology* (pp. 456–472). Berlin: Springer.
31. Stein, W., et al. (2016). Sage mathematics software (version 7.0).

Cristian S. Calude has a personal chair at the University of Auckland. His areas of research include complexity theory, algorithmic information theory, quantum physics and computing, history and philosophy of mathematics and computing sciences. For more information see https://calude.net/cristianscalude/about/ and https://en.wikipedia.org/wiki/Cristian_S._Calude.



Michael J. Dinneen received his PhD from the University of Victoria (B.C. Canada) in 1996 and is currently a Senior Lecturer at the University of Auckland. Prior to that he was employed for several years at the Los Alamos National Laboratory (New Mexico, USA) working on grand-challenge combinatorial search and optimization problems using supercomputers, such as those developed by Cray Research. Besides his specialty of graph theory and algorithms, he does research on unconventional models of computation such as (adiabatic) quantum computing and membrane computing, culminating in over 100 research papers.

Richard Hua received his BSc and Master of Science with first class honours from the University of Auckland in 2012 and 2016, respectively. He is currently a PhD student in adiabatic quantum computing supported by a University of Auckland Doctoral Scholarship. His research interests also include graph theory, algorithms and other theoretical computer science topics.