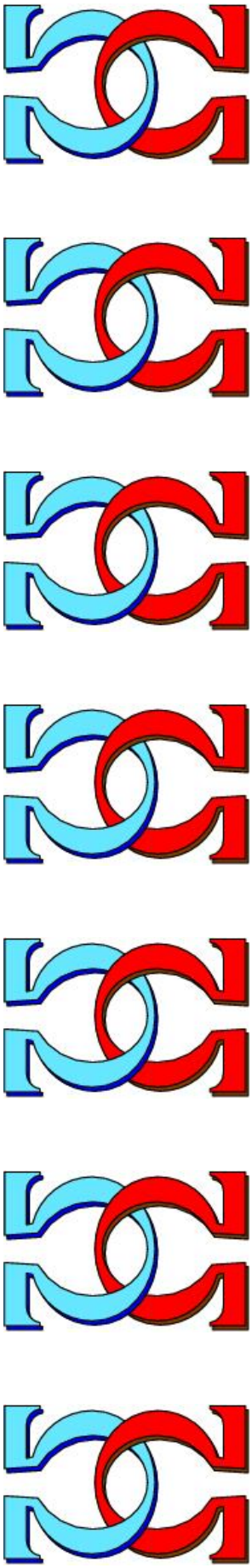


CDMTCS
Research
Report
Series

QUBO Formulations for Arithmetic Progression Graph Labeling Problems

C. S. Calude, M. J. Dinneen and Y. Liu

School of Computer Science, University of Auckland, New Zealand

CDMTCS-579
July 2024

Centre for Discrete Mathematics and
Theoretical Computer Science

QUBO Formulations for Arithmetic Progression Graph Labeling Problems

Cristian S. Calude, Michael J. Dinneen, and Yitong Liu

School of Computer Science
University of Auckland, New Zealand

July 4, 2024

Abstract

The Arithmetic Progression Graph Labeling is an NP-complete problem with various applications, including optimizing scheduling problems. This paper presents Quadratic Unconstrained Boolean Optimization (QUBO) solutions for the version with fixed vertex labels of this problem, then the original general problem. We use and compare standard (D-Wave Advantage and Advantage 2 Prototype) and hybrid (D-Wave Leap Hybrid Solver Service) methods to solve these problems with D-Wave quantum machines. Our experiments suggest that the hybrid methods outperform the standard ones.

1 Introduction

The Arithmetic Progression Graph Labeling Problem (APGLP) consists of labeling the edges of a graph with positive integers such that the sequence of the sums of incident edges of each vertex makes a finite arithmetic progression. The APGLP is an NP-complete problem [13, 10, 11], so the time needed to find a solution increases exponentially as the problem's size expands.

The Quadratic Unconstrained Binary Optimization (QUBO) problem is a combinatorial NP-hard problem that encodes optimization problems from various areas. The solutions of the QUBO problems are restricted to binary values of zero or one, which simplifies the decision space while still allowing the representation of a wide array of problems, including machine learning tasks [8], scheduling problems to optimize resource allocation [14, 4], and many graph theory problems [23]. The binary aspect makes QUBO particularly suitable for computational techniques such as quantum annealing and other heuristic algorithms.

The D-Wave quantum computer uses a quantum annealing approach to generate the solutions for QUBO problems. The first Advantage2 Prototype uses the Z4 Zephyr topology

and 500+ qubits [19]. Recently, D-Wave announced its new 1200+ qubits Advantage2 Prototype with the Z6 Zephyr topology that uses a new lower-noise fabrication stack [7]. We used the Advantage system and the new 1200+ qubits Advantage2 Prototype to solve two problems: the Fixed Vertex Labeled APGLP and the general APGLP. Our experiments were conducted on small graphs with various D-Wave solver parameters. Furthermore, we compared the performance of the D-Wave hybrid solver and the Python (Mixed Integer Programming) MIP solver for the APGLP. Advantage2 Prototype performed better on the APGLP, while Advantage excelled in Fixed Vertex Labeled APGLP. Advantage2 Prototype also required less QPU access time for processing and embedding both problems. Although the Python MIP solver outperformed the D-Wave hybrid solver in all cases, it is essential to note that the D-Wave hybrid solver, being a probabilistic solver, has a chance of returning an optimal result in significantly less time than the Python MIP solver typically requires when the problem gets larger.

In this paper, we formulate the APGLP as a QUBO problem, use classical algorithms and quantum annealing to solve it, and compare the efficiencies of these solutions.

2 D-Wave Systems

This section briefly presents the architecture and the computational models of the D-Wave Systems.

2.1 D-Wave architecture

Founded in 1999, D-Wave uses a quantum annealing process to search for optimal solutions to optimization problems [15]. The D-Wave quantum processing unit (QPU) operates at temperatures about or below 15 milli-Kelvins, maintained in isolation from the surrounding environment to behave quantum mechanically [16]. The qubits in the D-Wave quantum processor encode information in the form of magnetic flux quantum, the quantum of magnetic flux passing through a superconductor [9, 12].

The Advantage QPU [17] utilizes a P16 Pegasus graph topology for its lattice design, featuring 5 000+ qubits. The number of couplers per qubit is 15, resulting in 35 000+ couplers. The expanded count of qubits and couplers, combined with the improved connectivity offered by the Pegasus design, allows the solving of larger application problems directly on the Advantage QPUs. The next generation QPU Advantage2 uses the Zephyr topology [2]. The Zephyr topology achieves a degree of 20 (each qubit is connected to 20 different qubits through couplers). The full Advantage2 [6, 19], yet to be released, is expected to use a Z15 Zephyr graph topology with 7 000+ qubits and 60 000+ couplers. However, D-Wave recently announced a new Advantage2 Prototype that uses Z6 Zephyr graph topology with 1 200+ qubits and 10 000+ couplers [7]. The results in Section 5 have been obtained with the Advantage QPU and the 1 200+ qubits Advantage2 Prototype QPU; in this paper, we will refer to them as the Advantage and the Advantage2 Prototype, respectively.

2.2 D-Wave computations

To solve a problem with D-Wave, one must first formulate the problem as a QUBO where the lowest values are the optimal solutions to the problem the objective function represents.

A Quadratic Unconstrained Binary Optimization (QUBO) problem is defined using an upper-diagonal matrix Q, which is an $n \times n$ upper-triangular matrix of real weights, and a vector x of n binary variables. The problem is to minimize the following function:

$$F(x) = \min_{x \in \{0,1\}^n} \left(\sum_i Q_{i,i} x_i + \sum_i \sum_{j>i} Q_{i,j} x_i x_j + c \right),$$

where the variables x_i take values of 1 and 0, and c is an offset constant value.

The D-Wave system computation starts with a collection of initially uncoupled qubits in a superposition or ground state. Through the quantum annealing process, the qubits are subjected to magnetic fields and entangled via couplers. As the problem Hamiltonian is gradually introduced, excited states have the potential to approach the ground state. The closer these states become, the greater the probability of the system transitioning from the ground state to one of the excited states through tunneling. By the end of the quantum annealing process, each qubit settles into a classical state that reflects the problem's lowest energy state or one close to it.

This objective function is then submitted to one of the quadratic computational models that D-Wave solves.

2.3 The Leap Hybrid Solver Service

Hybrid solvers are used for large-scale problems that cannot be solved directly by the quantum system. The Hybrid Solver Service (HSS) is a cloud-based resource providing a collection of hybrid solvers that utilizes both classical and quantum computing methods tailored for various inputs and applications scenarios [18].

The HSS has two versions of portfolio solvers: version 1 employs the 2000Q, and version 2, used in this article, incorporates the Advantage. The front end of a portfolio solver selects and runs multiple hybrid solvers in parallel and returns the optimal solution. This approach frees users from guessing which solver will perform best for a given input.

The HSS Binary Quadratic Model (BQM) solver accepts a QUBO as input and an optional maximum time limit. If no time limit is given, the HSS assesses the size and structure of the input and then determines a minimum time limit to ensure that each solver has sufficient time to execute the process at least once; it receives at least one response from the Advantage. The time limit ranges from 3 seconds to 24 hours. Once the HSS receives the input, the front end selects one or more hybrid solvers for the specific input and runs them in parallel.

The hybrid solvers contain a query module (QM) communicating with the Advantage. The QM formulates partial representations of the given input and sends the smaller quantum queries, which the Advantage can directly solve. The QM gathers responses from the Advantage and transforms the replies into suggestions for the hybrid solvers, indicating promising areas of the solution space to be explored. Before reaching the time limit, the hybrid solvers send their results to the front end, forwarding the lowest-cost solution discovered to the user.

2.4 Minor embeddings

To solve a QUBO, represented as a matrix Q, we first need to map the qubit interactions (represented by non-zero entries of Q) onto actual QPU hardware architecture. The matrix Q can be viewed as an adjacency matrix of a graph, called the problem's logical graph. To solve a QUBO problem with quantum solvers, a minor embedding is essential for mapping the logical graph of the problem into the physical topology of a system's QPU. Nodes in the logical graph are mapped to one or more physical qubits, and the edges are mapped to physical couplers. Due to the sparse connectivity of QPU physical topologies, it is not feasible to fully map a connected graph directly onto specific qubits on a QPU. The challenge of sparse connectivity is mitigated by chaining qubits together and mapping some nodes into chains (connected sets, not necessarily a path) of physical qubits. Qubit chaining is achieved by setting the strength to sufficiently negative values for the couplers connecting the physical qubits. This ensures a strong correlation between the states of the qubits in the same chain. Therefore, chained qubits will likely result in the same classical state when measured after annealing, representing the same binary value: they collectively operate as if they were a single variable. (Note: if the physical qubits corresponding to a logical qubit do not end up in the same state after annealing, we often take the majority value of each chain for each logical qubit.) The general embedding function $find\_embedding()$ implements a heuristic algorithm [3] to find a minor logical source graph's embedding into a given target physical graph. One objective of the minor embedding algorithms is to minimize the chain sizes to minimize potential entanglement errors on the physical hardware, affecting the results' accuracy.

Note that some computational trade-offs need to be taken, as there is a limited set of values of coupler strengths supported by the current D-Wave hardware. While embedding a problem onto the QPU, the auto-scaling feature adjusts all the coupler strengths for the problem to ensure they fall within the $[-1, +1]$ range. The chain strengths, being the maximum value, are set to one. As chain strengths increase, the relative strength of couplers describing the logical problem diminishes. Consequently, with the continuous increase in chain strength, each chain begins to separate, and the physical model may no longer represent the original problem [5].

3 Arithmetic Progression Graphs

Given an undirected graph $G = (V, E)$ with n vertices and m edges, an *AP-labeling* of G is a labeling of the edges of G with positive integers that induce an arithmetic progression

of vertex labels.

A graph with such a labeling is called an *AP-graph*. An AP-labeling is defined with three parameters: X, a, d, where $X : E \to \mathbf{Z}^+$ ($\mathbf{Z}^+$ is the set of positive integers) is a total function to assign positive integers to edges, $a \in \mathbf{Z}^+$ is the initial value, and $d \in \mathbf{Z}^+$ is the constant difference of the arithmetic progression over the vertices labels. From an edge labeling X, the (induced) vertex labels are defined by the function $Y : V \to Z$ where for $v \in V$ we have $Y(v) = \sum_{u \in N(v)} X(uv)$. Here $N(v)$ denotes the vertex neighbors of a vertex v.

Formally, a graph is an AP-graph if there exist positive integer constants a and d, an edge labeling X, and a permutation σ over the set of vertex indices $\{0, 1, \ldots, n-1\}$, such that $Y(v_i) = a + \sigma_i d$, for $v_i \in V$.

The input for the *Arithmetic Progression Labeling Problem (APGLP)* consists of a given graph G with the initial value a and the constant difference d. The problem is whether an edge labeling X exists. To solve the APGLP, we must find (if possible) a set of edge labels X such that the induced vertex labels Y form an arithmetic progression.

Figure 1 gives an example of edge/vertex labeling of an AP-graph.

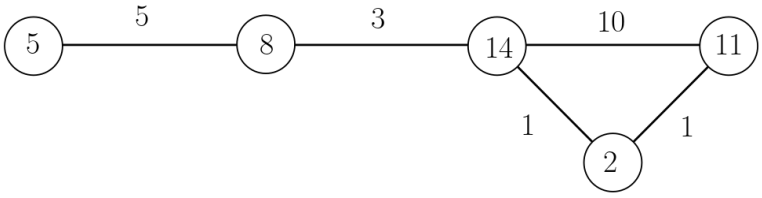


Figure 1: An AP-labeling of a graph with five nodes and five edges, start value $a = 2$, and a constant difference $d = 3$. The vertex labels form the arithmetic progression (2, 5, 8, 11, 14).

3.1 Fixed Vertex Labeled APGLP

We first solve an easier version of the APGLP by fixing the vertex labels and a and d. Given a graph $G = (V, E)$ with n vertices and m edges. If $V = \{v_0, v_1, ..., v_{n-1}\}$, and $E = \{e_0, e_1, ..., e_{m-1}\}$, we construct the $n \times m$ incidence matrix $M = M_{i,j}$ of G, such that $M_{i,j}$ equals to one if vertex v_i is an incident with edge e_j.

The highest possible label for an edge cannot be greater than $z = a + (n-2)d$, as the two incident vertices attached to this edge would have labels that violate the target arithmetic progression for a and d. Working with the upper bound for z we use the binary variables $y_{j,k}$ to represent and encode edge labels. The edge label for e_j is $\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k} + 1$. Note that the term $+1$ ensures that the edge labels must be positive integers. In this particular form of APGLP we fix the vertex values $Y(v_i)$ of an arithmetic progression.

The QUBO objective function for this problem of APGLP is:

$$F_1(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{m-1} \left(M_{i,j} (\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k} + 1) \right) - Y(v_i) \right)^2 ,$$

where $M_{i,j}$, and $Y(v_i)$ are given as inputs for the input variables $x = (y_{0,0}, \ldots, y_{m-1,\lceil \log z \rceil - 1})$.

As the function $F_1(x)$ is a sum of squares, the minimum value will be reached when the value inside the outermost braces is zero. The expression $\sum_{j=0}^{m-1}(M_{i,j}(\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k}+1))$ is the sum of all incidence edge labels for a vertex v_i. Since $Y(v_i)$ is a fixed label for vertex v_i, the function $F_1(x)$ is minimized when the sum and $Y(v_i)$ are equal. As this is the sole requirement that needs to be satisfied, no additional constraints are required. The degree of a vertex v_i, denoted as $\Delta(v_i)$, is also expressed through the summation as $\sum_{j=0}^{m-1} M_{i,j}$.

The offset ensures that the smallest value is zero, hence we can expand the objective function to the following:

$$F_1(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{m-1} (M_{i,j} (\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k})) + \Delta(v_i) - Y(v_i) \right)^2 ,$$

where the offset value is $\sum_{i=0}^{n-1}(\Delta(v_i) - Y(v_i))^2$.

3.2 An example

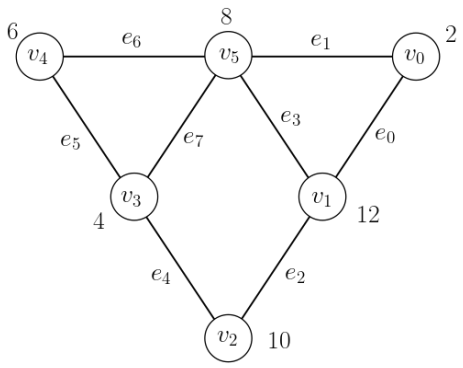


Figure 2: Graph G with six vertices and eight edges with the fixed vertex labels indicated.

For illustration, we consider the input graph $G = (V, E)$ with six nodes and eight edges in Figure 2. The start value a and the constant difference value d are set to 2. The upper bound for edge labels z is $a + (n-2)d = 2 + (6-2) \cdot 2 = 10$ and the vertex labels Y for $V = \{v_0, v_1, v_2, v_3, v_4, v_5\}$ are $\{2, 12, 10, 4, 6, 8\}$.

The following matrix is the incidence matrix of graph G of Figure 2:

$$
\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\
\left[\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}\right]
\end{array}
$$

By expanding the objective function, we get:

$$F_1(x) = \sum_{i=0}^{5} \left((\sum_{j=0}^{7} (M_{i,j} \sum_{k=0}^{\lceil \log 10 \rceil - 1} 2^k y_{j,k})) + \Delta(v_i) - Y(v_i) \right)^2 .$$

Below is a term of $F_1(x)$ for $i = 2$. Notice that, with the exception of $y_{j,k}$, all remaining terms are constants.

$$
\begin{aligned}
F_1(x,2) = {} & (y_{2,0} + 2y_{2,1} + 4y_{2,2} + 8y_{2,3} + y_{4,0} + 2y_{4,1} + 4y_{4,2} + 8y_{4,3})^2 + 16(y_{2,0} + 2y_{2,1} + \\
& 4y_{2,2} + 8y_{2,3} + y_{4,0} + 2y_{4,1} + 4y_{4,2} + 8y_{4,3}) + 64. \\
F_1(x,2) = {} & 17y_{2,0}^2 + 36y_{2,1}^2 + 80y_{2,2}^2 + 192y_{2,3}^2 + 17y_{4,0}^2 + 36y_{4,1}^2 + 80y_{4,2}^2 + 192y_{4,3}^2 + \\
& 4y_{2,0}y_{2,1} + 8y_{2,0}y_{2,2} + 16y_{2,0}y_{2,3} + 2y_{2,0}y_{4,0} + 4y_{2,0}y_{4,1} + 8y_{2,0}y_{4,2} + \\
& 16y_{2,0}y_{4,3} + 16y_{2,1}y_{2,2} + 32y_{2,1}y_{2,3} + 4y_{2,1}y_{4,0} + 8y_{2,1}y_{4,1} + 16y_{2,1}y_{4,2} + \\
& 32y_{2,1}y_{4,3} + 64y_{2,2}y_{2,3} + 8y_{2,2}y_{4,0} + 16y_{2,2}y_{4,1} + 32y_{2,2}y_{4,2} + 64y_{2,2}y_{4,3} + \\
& 16y_{2,3}y_{4,0} + 32y_{2,3}y_{4,1} + 64y_{2,3}y_{4,2} + 128y_{2,3}y_{4,3} + 4y_{4,0}y_{4,1} + 8y_{4,0}y_{4,2} + \\
& 16y_{4,0}y_{4,3} + 16y_{4,1}y_{4,2} + 32y_{4,1}y_{4,3} + 64y_{4,2}y_{4,3}.
\end{aligned}
$$

By repeating this process for each value of $i = 0, ..., 5$ we complete the entire QUBO matrix using $F_1(x)$. An optimal solution is shown in Table 1, where the coefficients for the binary variables $y_{2,3}, y_{3,0}, y_{5,0}, y_{7,0}, y_{7,1}$, highlighted in the table, have been assigned the value 1. The offset value is $\sum_{i=0}^{n-1} (\Delta(v_i) - Y(v_i))^2 = 178$. Optimal solutions are obtained when the value of the objective function is 0. An example of optimal solution satisfies is: $-144y_{2,3}^2 + 16y_{2,3}y_{3,0} - 24y_{3,0}^2 + 2y_{3,0}y_{7,0} + 4y_{3,0}y_{7,1} - 8y_{5,0}^2 + 2y_{5,0}y_{7,0} + 4y_{5,0}y_{7,1} - 14y_{7,0}^2 + 8y_{7,0}y_{7,1} - 24y_{7,1}^2$ equals to -178. The edge labels for $E = \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ are calculated from $\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k} + 1$ for $E = \{1, 1, 9, 2, 1, 2, 1, 4\}$.

The complete QUBO matrix is shown in Table 1.

Table 1: QUBO matrix for the example in Figure 2.

Variables	y0,0	y0,1	y0,2	y0,3	y1,0	y1,1	y1,2	y1,3	y2,0	y2,1	y2,2	y2,3	y3,0	y3,1	y3,2	y3,3	y40	y4,1	y4,2	y4,3	y5,0	y5,1	y5,2	y5,3	y6,0	y6,1	y6,2	y6,3	y7,0	y7,1	y7,2	y7,3
y0,0	-16	8	16	32	2	4	8	16	2	4	8	16	2	4	8	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y0,1	0	-28	32	64	4	8	16	32	4	8	16	32	4	8	16	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y0,2	0	0	-40	128	8	16	32	64	8	16	32	64	8	16	32	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y0,3	0	0	0	-16	16	32	64	128	16	32	64	128	16	32	64	128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y1,0	0	0	0	0	-6	8	16	32	0	0	0	0	2	4	8	16	0	0	0	0	0	0	0	0	2	4	8	16	2	4	8	16
y1,1	0	0	0	0	0	-8	32	64	0	0	0	0	4	8	16	32	0	0	0	0	0	0	0	0	4	8	16	32	4	8	16	32
y1,2	0	0	0	0	0	0	0	128	0	0	0	0	8	16	32	64	0	0	0	0	0	0	0	0	8	16	32	64	8	16	32	64
y1,3	0	0	0	0	0	0	0	64	0	0	0	0	16	32	64	128	0	0	0	0	0	0	0	0	16	32	64	128	16	32	64	128
y2,0	0	0	0	0	0	0	0	0	-32	8	16	32	2	4	8	16	2	4	8	16	0	0	0	0	0	0	0	0	0	0	0	0
y2,1	0	0	0	0	0	0	0	0	0	-60	32	64	4	8	16	32	4	8	16	32	0	0	0	0	0	0	0	0	0	0	0	0
y2,2	0	0	0	0	0	0	0	0	0	0	-104	128	8	16	32	64	8	16	32	64	0	0	0	0	0	0	0	0	0	0	0	0
y2,3	0	0	0	0	0	0	0	0	0	0	0	-144	16	32	64	128	16	32	64	128	0	0	0	0	0	0	0	0	0	0	0	0
y3,0	0	0	0	0	0	0	0	0	0	0	0	0	-24	8	16	32	0	0	0	0	0	0	0	0	2	4	8	16	2	4	8	16
y3,1	0	0	0	0	0	0	0	0	0	0	0	0	0	-44	32	64	0	0	0	0	0	0	0	0	4	8	16	32	4	8	16	32
y3,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-72	128	0	0	0	0	0	0	0	0	8	16	32	64	8	16	32	64
y3,3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-80	0	0	0	0	0	0	0	0	16	32	64	128	16	32	64	128
y4,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-16	8	16	32	2	4	8	16	2	4	8	16	0	0	0	0
y4,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-28	32	64	4	8	16	32	4	8	16	32	0	0	0	0
y4,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-40	128	8	16	32	64	8	16	32	64	0	0	0	0
y4,3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-16	16	32	64	128	16	32	64	128	0	0	0	0
y5,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-8	8	16	32	2	4	8	16	2	4	8	16
y5,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-12	32	64	4	8	16	32	4	8	16	32
y5,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-8	128	8	16	32	64	8	16	32	64
y5,3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	16	32	64	128	16	32	64	128
y6,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-8	8	16	32	2	4	8	16
y6,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-12	32	64	4	8	16	32
y6,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-8	128	8	16	32	64
y6,3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	16	32	64	128
y7,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-14	8	16	32
y7,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-24	32	64
y7,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-32	128
y7,3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.3 The APGL Problem

We now solve the general version of APGLP for a graph $G_2 = (V, E)$ with n vertices and m edges. $V = \{v_0, v_1, ..., v_{n-1}\}$ and $E = \{e_0, e_1, ..., e_{m-1}\}$. We construct the incidence matrix $M = M_{i,j}$ of G_2 and note that the upper bound for an edge label is $z = a+(n-2)d$. We use the binary variable $y_{j,k}$ to label the edge e_j as $\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k} + 1$, where $0 \leq k \leq \lceil log(z) \rceil - 1$.

The binary variables $T_{i,h}$ with $0 \leq h < n$ label the vertices: $T_{i,h}$=1 if and only if v_i= $a + hd$. The constraint $P(x)$ enforces that $hT_{i,h}$ in set $\{0, 1, ..., n-1\}$ represents a permutation. As a result, only one value in $\{a + 0b, a + 1b, ..., a + (n-1)b\}$ is assigned to each vertex, and each value can only be assigned once. The QUBO binary vector is $x = (y_{0,0}, ..., y_{m-1,\lceil \log z \rceil - 1}, T_{0,0}, ..., T_{n-1,n-1})$ and the objective function to be minimized is:

$$F_2(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{m-1} (M_{i,j} (\sum_{k=0}^{\lceil logz \rceil - 1} 2^k y_{j,k} + 1)) - a - \sum_{h=0}^{n-1} (hT_{i,h})d) \right)^2 + P(x),$$

subject to the constraint $P(x) = C_1(x) + C_2(x)$,

$$C_1(x) = \sum_{i=0}^{n-1} \left((\sum_{h=0}^{n-1} T_{i,h}) - 1 \right)^2 = 0, C_2(x) = \sum_{h=0}^{n-1} \left((\sum_{i=0}^{n-1} T_{i,h}) - 1 \right)^2 = 0.$$

The constraints $C_1(x)$ and $C_2(x)$ can be reformulated in the slightly more efficient quadratic penalty, which achieves the minimum value $-n$ (see [22]):

$$P(x) = \sum_{i=0}^{n-1} \sum_{h<h'} T_{i,h} T_{i,h'} + \sum_{h=0}^{n-1} \sum_{i<i'} T_{i,h} T_{i',h} - \sum_{i=0}^{n-1} \sum_{h=0}^{n-1} T_{i,h}.$$

Theorem 1. *The minimum value for the function $F_2(x)$ is $-n$, and any non-optimal (i.e. not a valid AP-labeling) assignment of the edge and vertex labels will result in a value greater than $-n$.*

Proof.

Let $G(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{m-1} (M_{i,j} (\sum_{k=0}^{\lceil logz \rceil - 1} 2^k y_{j,k} + 1)) - a - \sum_{h=0}^{n-1} (hT_{i,h})d) \right)^2$. Since $G(x)$ is a sum of squares, its minimum value will be shown to be 0 exclusively when the edge and vertex labels assigned are optimal. Given the knowledge that the constraint $P(x)$ possesses a minimum value of $-n$, the minimum value for the objective function $F_2(x)$ is $-n$.

We first consider the function $G(x)$. We now parametrize vertex v_i and let:

$$G_1(x,i) = \sum_{j=0}^{m-1}(M_{i,j}(\sum_{k=0}^{\lceil log_z \rceil - 1} 2^k y_{j,k} + 1)),$$
$$G_2(x,i) = a + \sum_{h=0}^{n-1}(hT_{i,h})d.$$

$G_1(x,i)$ be the sum of all its incident edge labels of v_i. The $+1$ is to ensure that no edge label equals zero. And $G_2(x,i)$ is the vertex label $a + \sigma_i d$ for some permutation σ represented by the $T_{i,h}$ variables.

Any non-optimal assignment would mean that $G_1(x,i)$ and $G_2(x,i)$ are unequal. And if not equal we get $(G_1(x,i) - G_2(x,i))^2$ bigger than 0. In this way, $G(x)$ enforces that for every vertex v_i in graph G, its vertex label equals the sum of all its incident edge labels.

We next come to the constraint $P(x)$ where we claim $P(x)$ is minimized to $-n$ only when the set of all vertices V is some permutation σ on $S = \{0, 1, \ldots, n-1\}$. We split $P(x)$ into three small functions where $P(x) = P_1(x) + P_2(x) + P_3(x)$:

$$P_1(x) = \sum_{i=0}^{n-1}\sum_{h<h'} T_{i,h}T_{i,h'}, \quad P_2(x) = \sum_{h=0}^{n-1}\sum_{i<i'} T_{i,h}T_{i',h}, \quad P_3(x) = -\sum_{i=0}^{n-1}\sum_{h=0}^{n-1} T_{i,h}.$$

$P_1(x)$ is minimized to zero only if, for each vertex v_i, $T_{i,h}$ and $T_{i,h'}$ are not both one for any pair of h and h'. $P_1(x)$ ensures that each vertex is assigned at most one value. $P_2(x)$ is minimized to zero only if, for each value h, $T_{i,h}$ and $T_{i',h}$ are not both one for any two of vertices v_i and $v_{i'}$. $P_2(x)$ ensures that each value is assigned to at most one vertex. $P_3(x)$ is minimized to $-n$ without violating $P_1(x)$ and $P_2(x)$. Any violation that reduces $P_3(x)$ by one will increase the combined value of $P_1(x)$ and $P_2(x)$ by two. Therefore, $P(x)$ is minimized to $-n$.

Moreover, the objective function of a QUBO can only have quadratic terms. Removing any constant term or simply squaring the linear terms so they are quadratic will not influence the optimal solutions generated. Therefore, after expanding, converting linear terms to quadratic terms by replacing all binary variables with their squares and deleting the constant term a^2. The minimized value for the objective function is $-\sum_{i=0}^{n-1}(\Delta(v_i) - a)^2 - n$. The offset is $\sum_{i=0}^{n-1}(\Delta(v_i) - a)^2 + n$. □

3.4 An example

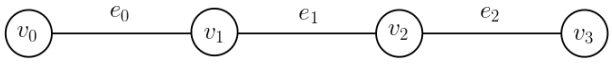


Figure 3: Graph G_2 with four nodes and three edges.

Figure 3 presents an example of an input graph $G_2 = (V, E)$ for the AP-Labeling problem, featuring four nodes and three edges. With the starting value a set to 2 and the difference d at 1, the value of z can be calculated as $a + (n-2)d = 4$.

Below is the incidence matrix for graph G_2:

$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} e_0 & e_1 & e_2 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

By expanding and simplifying the objective function, we get:

$$\begin{aligned} F_2(x) =& H(x) + P(x) \\ =& \sum_{i=0}^{3}(\sum_{j=0}^{2}(M_{i,j}(\sum_{k=0}^{1}2^k y_{j,k}+1)) - 2 - \sum_{h=0}^{3}(hT_{i,h})))^2 + \\ & \sum_{i=0}^{3}\sum_{h<h'}T_{i,h}T_{i,h'} + \sum_{h=0}^{3}\sum_{i<i'}T_{i,h}T_{i',h} - \sum_{i=0}^{3}\sum_{h=0}^{3}T_{i,h}, \end{aligned}$$

$$\begin{aligned} H(x) =& \sum_{i=0}^{3}(\sum_{j=0}^{2}(M_{i,j}(\sum_{k=0}^{1}2^k y_{j,k}+1)) - 2 - \sum_{h=0}^{3}(hT_{i,h})))^2 \\ =& \sum_{i=0}^{3}((\sum_{j=0}^{2}(M_{i,j}\sum_{k=0}^{1}2^k y_{j,k}))^2 - (4-2\Delta(v_i))\sum_{j=0}^{2}(M_{i,j}\sum_{k=0}^{1}2^k y_{j,k}) + (\sum_{h=0}^{3}(hT_{i,h}))^2 + \\ & (4-2\Delta(v_i))(\sum_{h=0}^{3}(hT_{i,h})) - 2(\sum_{j=0}^{2}(M_{i,j}\sum_{k=0}^{1}2^k y_{j,k}))(\sum_{h=0}^{3}(hT_{i,h})) - 4\Delta(v_i) + \\ & \Delta(v_i)^2). \end{aligned}$$

Note that $M_{i,j}$ is not a variable but a constant that can be calculated from the given input. For example, when $i = 0$:

$$\begin{aligned} H(x,0) =& ((y_{0,0} + 2y_{0,1} + 4y_{0,2} + 1) - 2 - (0T_{0,0} + T_{0,1} + 2T_{0,2} + 3T_{0,3}))^2 \\ =& (y_{0,0} + 2y_{0,1} + 4y_{0,2})^2 + (0T_{0,0} + T_{0,1} + 2T_{0,2} + 3T_{0,3})^2 - 2(y_{0,0} + 2y_{0,1} + \\ & 4y_{0,2})(0T_{0,0} + T_{0,1} + 2T_{0,2} + 3T_{0,3}) - 2(y_{0,0} + 2y_{0,1} + 4y_{0,2}) + 2(0T_{0,0} + \\ & T_{0,1} + 2T_{0,2} + 3T_{0,3}) - 3 \\ =& -y_{0,0}^2 + 8y_{0,2}^2 + 4y_{0,0}y_{0,1} + 8y_{0,0}y_{0,2} + 16y_{0,1}y_{0,2} + 3T_{0,1}^2 + 8T_{0,2}^2 + 15T_{0,3}^2 + \\ & 4T_{0,1}T_{0,2} + 6T_{0,1}T_{0,3} + 12T_{0,2}T_{0,3} - 2y_{0,0}T_{0,1} - 4y_{0,0}T_{0,2} - 6y_{0,0}T_{0,3} - \\ & 4y_{0,1}T_{0,1} - 8y_{0,1}T_{0,2} - 12y_{0,1}T_{0,3} - 8y_{0,2}T_{0,1} - 16y_{0,2}T_{0,2} - 24y_{0,2}T_{0,3} \end{aligned}$$

Repeating this process for each value of $i = 0, ..., 3$ and adding $P(x)$ we complete the QUBO matrix using $F_2(x)$. The binary variables are $y_{j,k}$ and $T_{i,h}$.

An optimal solution is shown above in Table 2: the coefficients for the binary variables $y_{0,0}, y_{2,0}, y_{2,1}, T_{0,0}, T_{1,1}, T_{2,3}, T_{3,2}$, highlighted in the table, have been assigned the value 1.

The offset value is $\sum_{i=0}^{n-1}(\Delta(v_i) - a)^2 + n = 6$. Optimal solutions are obtained when the value of the objective function is 0. An example of an optimal solution satisfies as: $-2y_{0,0}T_{1,1} + 8y_{2,0}y_{2,1} - 6y_{2,0}y_{2,3} - 4y_{2,0}T_{3,2} + 4y_{2,1}^2 - 12y_{2,1}T_{2,3} - 8y_{2,1}T_{3,2} - T_{0,0}^2 + 8T_{2,3}^2 + 7T_{3,2}^2$ equals to -6. The vertex labels for $V = \{v_0, v_1, v_2, v_3\}$ are calculated from $a + \sum_{h=0}^{n-1}(hT_{i,h})d$ for $V = \{2, 3, 5, 4\}$. And the edge labels for $E = \{e_0, e_1, e_2\}$ are calculated from $\sum_{k=0}^{\lceil \log z \rceil - 1} 2^k y_{j,k} + 1$ for $E = \{2, 1, 4\}$, respectively.

Table 2: QUBO matrix for the example in Figure 3.

variables	y00	y01	y02	y10	y11	y12	y20	y21	y22	T00	T01	T02	T03	T10	T11	T12	T13	T20	T21	T22	T23	T30	T31	T32	T33
y00	0	8	16	2	4	8	0	0	0	0	-2	-4	-6	0	-2	-4	-6	0	0	0	0	0	0	0	0
y01	0	4	32	4	8	16	0	0	0	0	-4	-8	-12	0	-4	-8	-12	0	0	0	0	0	0	0	0
y02	0	0	24	8	16	32	0	0	0	0	-8	-16	-24	0	-8	-16	-24	0	0	0	0	0	0	0	0
y10	0	0	0	2	8	16	2	4	8	0	0	0	0	0	-2	-4	-6	0	-2	-4	-6	0	0	0	0
y11	0	0	0	0	8	32	4	8	16	0	0	0	0	0	-4	-8	-12	0	-4	-8	-12	0	0	0	0
y12	0	0	0	0	0	32	8	16	32	0	0	0	0	0	-8	-16	-24	0	-8	-16	-24	0	0	0	0
y20	0	0	0	0	0	0	0	8	16	0	0	0	0	0	0	0	0	0	-2	-4	-6	0	-2	-4	-6
y21	0	0	0	0	0	0	0	4	32	0	0	0	0	0	0	0	0	0	-4	-8	-12	0	-4	-8	-12
y22	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	-8	-16	-24	0	-8	-16	-24
T00	0	0	0	0	0	0	0	0	0	-1	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0
T01	0	0	0	0	0	0	0	0	0	0	2	5	7	0	1	0	0	0	1	0	0	0	1	0	0
T02	0	0	0	0	0	0	0	0	0	0	0	7	13	0	0	1	0	0	0	1	0	0	0	1	0
T03	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	1	0	0	0	1	0	0	0	1
T10	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	1	1	1	0	0	0	1	0	0	0
T11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	7	0	1	0	0	0	1	0	0
T12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	13	0	0	1	0	0	0	1	0
T13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	1	0	0	0	1
T20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	1	1	1	0	0	0
T21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	7	0	1	0	0	0
T22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	13	0	0	1	0
T23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	1
T30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	1	1
T31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	5	7
T32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	13
T33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14

4 Methodology

To solve the Fixed Vertex Labeled APGLP, we translate the edge label variables into binary, subject to constraints, as represented by the objective function F_1, as given in Section 3.1. For the general APGLP, we construct the objective function F_2, as presented in Section 3.3.

The obtained QUBOs have been solved using a D-Wave quantum annealer with three samplers: two QPU solvers, `Advantage_system_6.3` and `Advantage2_Prototype2.2`, and one hybrid solver, `hybrid_binary_quadratic_model_version2` (HSS). We used the default parameters for HSS. Different numbers of reads and annealing times are applied to both the Advantage and Advantage2 Prototype and compared to assess the performance and accuracy of the solutions. The number of reads parameter specifies how many cycles the QPU should run a problem. Annealing time sets the duration of each quantum annealing cycle per sampling.

The examples of simple connected graphs have been taken from [20]: they have been generated by `geng` in g6 format from `nauty` [21]. The APGs for these graphs have been constructed with a starting value in the $\{2, 3\}$ and a constant difference in $\{1, 2, 3\}$. All connected graphs with four to six nodes and ten randomly chosen graphs from all connected graphs with seven to eight nodes have been used for the experiment set. The Mixed-Integer Programming (MIP) code for verifying their APGs is adapted from [10]. The total time required to solve a problem with a QPU solver includes the sum of the time spent formulating the QUBO from the input problem, the time spent embedding the problem on the physical topology of the solver, and the QPU access time reported by the solver. For a hybrid solver. This total time comprises the QUBO formulation time and the runtime set by the user. Performance has been evaluated and compared with four measures for the two QPU solvers: the percentage of optimal solutions, the total runtime, the embedding time, and the QPU access time.

The Python code is provided in Appendix B.

Both solvers Advantage and Advantage2 Prototype have a maximum QPU access time of $1,000,000$ microseconds. The problems submitted to the QPU with an estimated access time exceeding this time limit resulted in an error.

5 Results

In this section, we present and discuss the experimental results. Time is measured in microseconds (μs).

5.1 Fixed Vertex Labeled APGLP

The bar graph in Figure 4 compares the percentage of optimal solutions found by the Advantage and the Advantage2 Prototype. Both solvers achieved optimal solutions for problems with four nodes in over 90% of the cases with 1 000 reads and the default

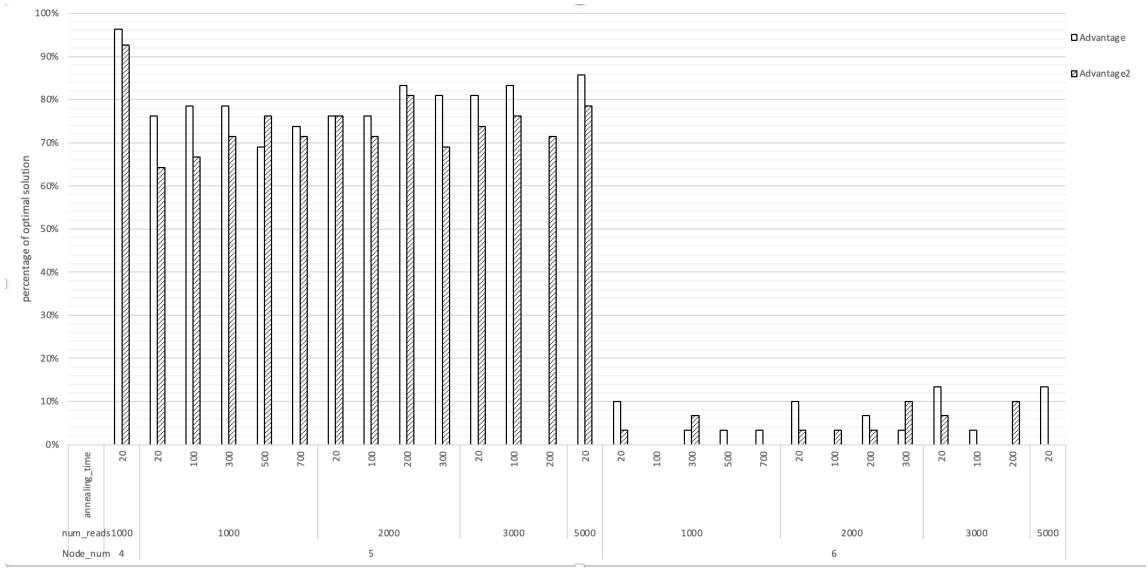


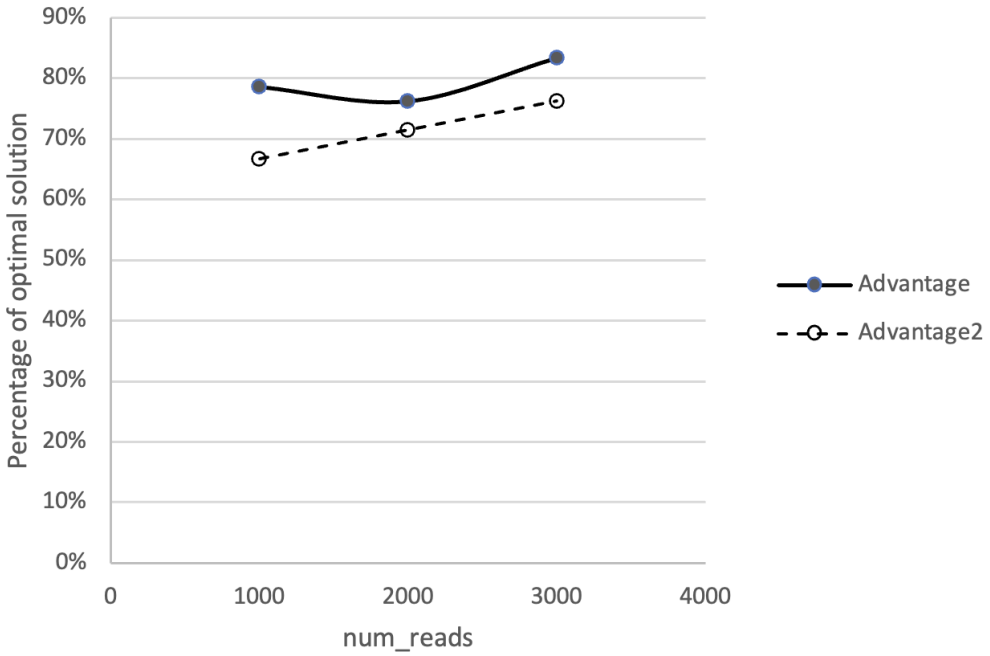
Figure 4: Comparison of optimal solution percentages for the fixed APGLP with four, five, and six nodes. The x-axis represents the number of reads and annealing times, and the y-axis indicates the proportion of optimal solutions obtained.

annealing time of $20\mu s$. With graphs of five nodes, the Advantage surpassed the Advantage2 Prototype nearly every time. The Advantage failed to solve some problems with graphs with five or six nodes, 3 000 reads, and an annealing time of $200\mu s$. Despite these limitations, the likelihood of obtaining optimal solutions for graphs with five nodes is mostly above 70%. However, for six node graphs, a noticeable and rapid decline in performance was recorded for both solvers, regardless of the sampling number or annealing time. Increasing the annealing time did not show an increased proportion of optimal solutions obtained.

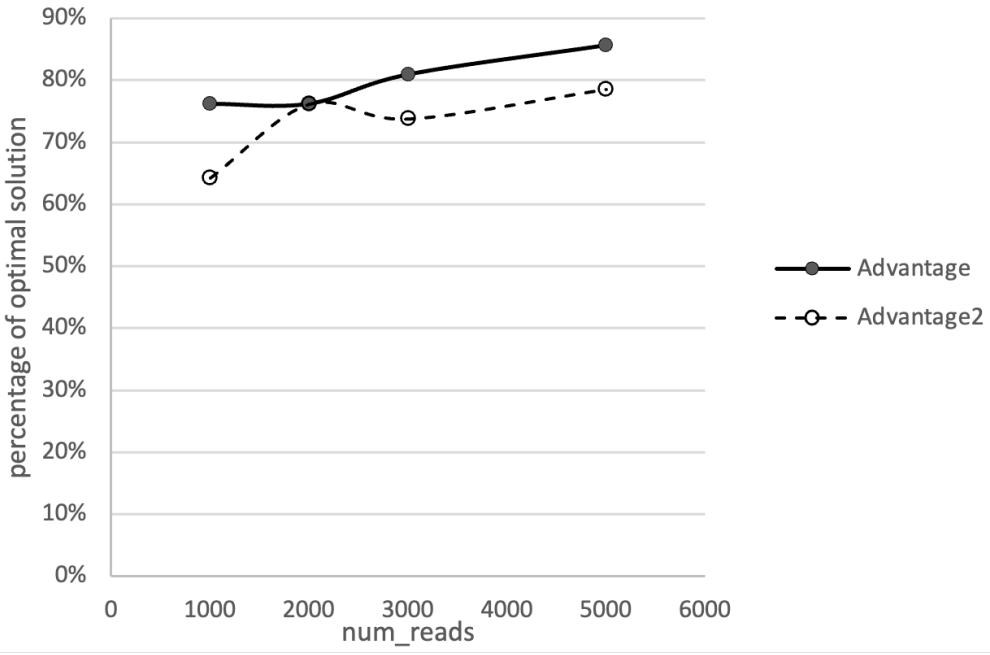
Figure 5 shows the performance trend of the two solvers for graphs with five nodes and constant annealing time. Both graphs show a gradual increase in the percentage of optimal solutions as the number of reads grows.

Figure 6 shows that the runtime increases as the annealing time increases and the number of reads remains constant. The upper graph indicates that Advantage2 Prototype generally achieves optimal results more quickly than Advantage for all tested conditions. The lower graph also shows similar patterns, with Advantage2 Prototype typically operating faster than Advantage when optimal solutions are not obtained. However, for larger graphs, the differences in runtime between these two are more significant when optimal solutions are not reached than when the optimal solutions are reached. It is also observed that non-optimal solutions tend to have a longer runtime than optimal ones.

In Figure 7, we compare average embedding times between the Advantage and the Advantage2 Prototype for graph order ranging from four to eight. For graphs with four nodes, the embedding times for both the Advantage and the Advantage2 Prototype are small and marginally higher for Advantage. As the graph order increases to five, an increase



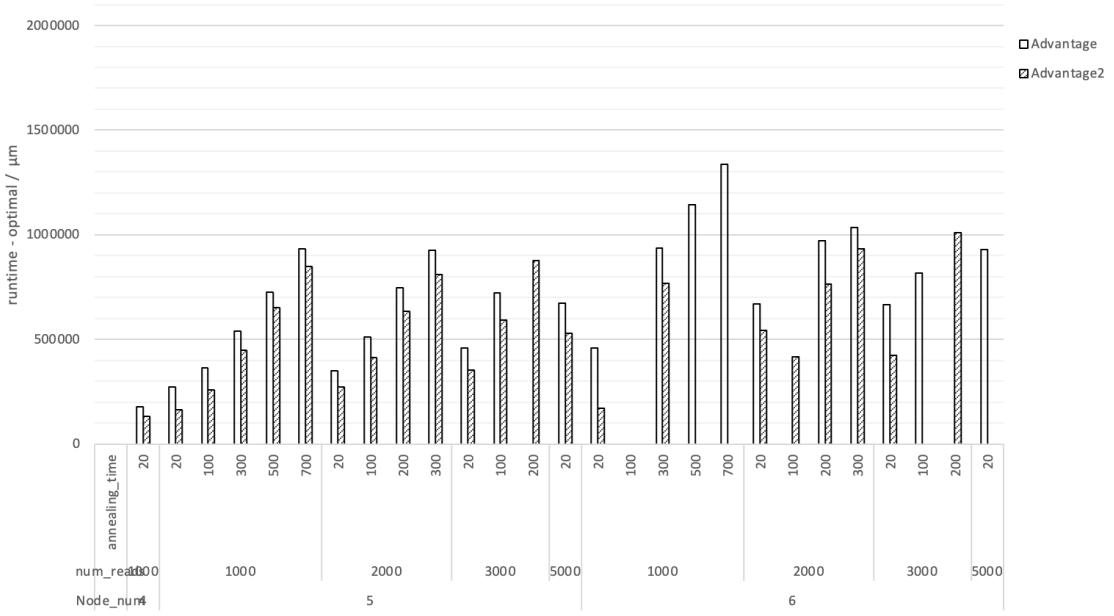
(a) The trend corresponding to an annealing time of $100\mu s$



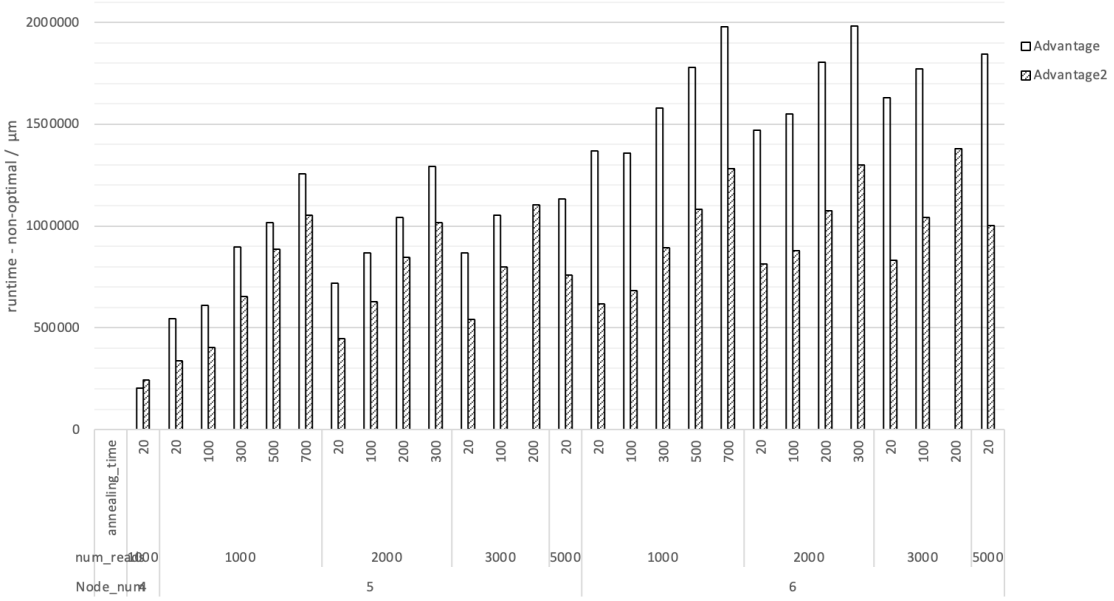
(b) The trend corresponding to an annealing time of $20\mu s$

Figure 5: Comparative performance of the solvers for graphs with five nodes and fixed annealing time.

in embedding time is noticeable, which is more significant for Advantage. A significant



(a) The average runtime of quantum solvers when optimal solutions are achieved.



(b) The average runtime of quantum solvers when optimal solutions are not received.

Figure 6: Runtime comparison for achieving optimal and non-optimal solutions.

increase in embedding time is also noted for graphs with seven and eight nodes.

Figure 8 shows that, generally, regardless of graph order, Advantage tends to have slightly higher average QPU access times than Advantage2 Prototype. However, no significant differences in QPU access times have been observed with increases in either the number of nodes or the number of reads.

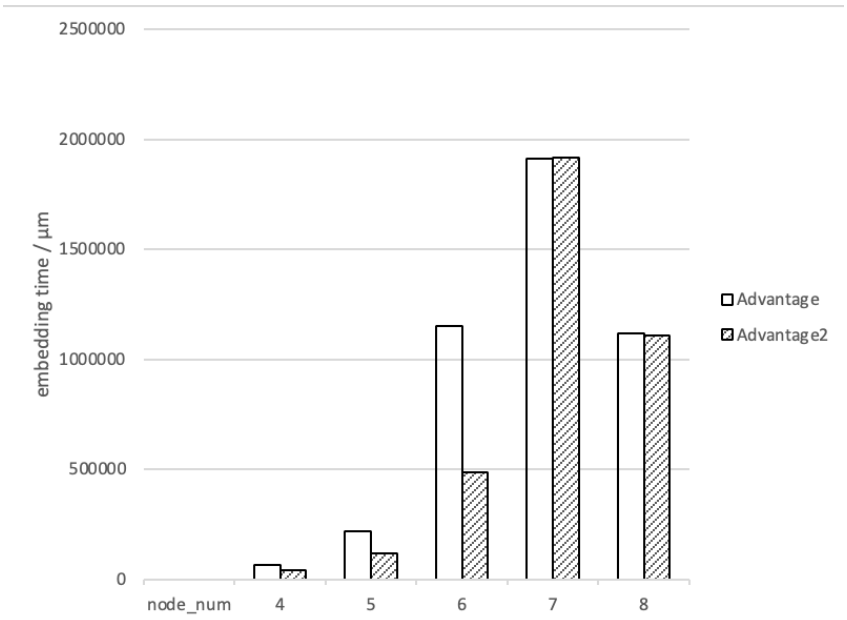


Figure 7: Average times required for the minor embedding of QUBO graphs for Advantage and Advantage2 Prototype.

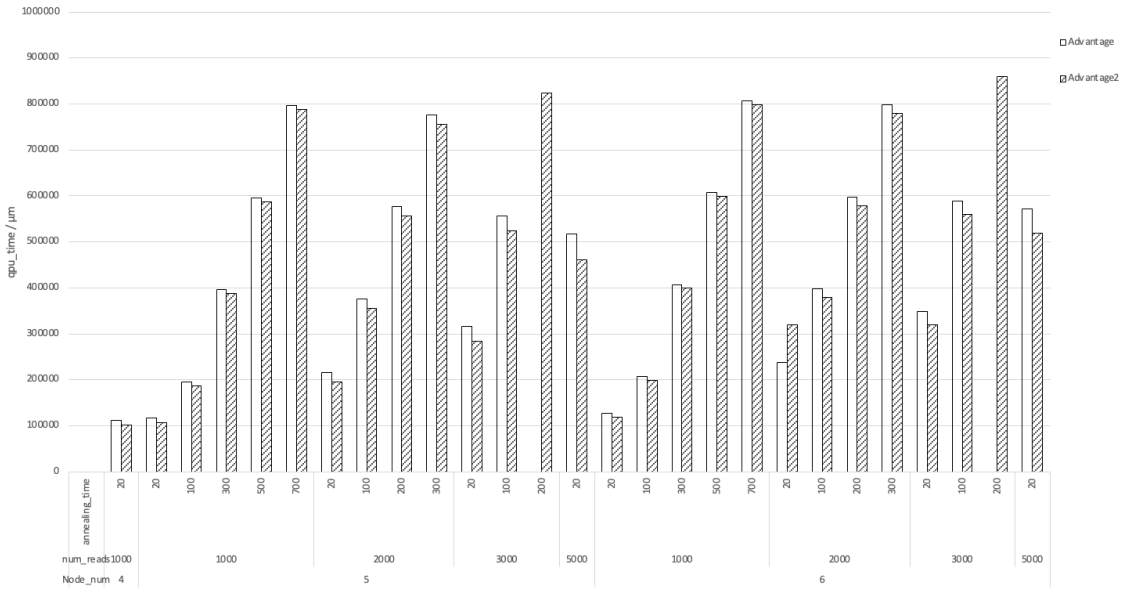


Figure 8: Average QPU access time for quantum solvers with the Advantage and the Advantage2 Prototype.

The average results for the APGLP with fixed vertex labels can be found in Table 5 in Appendix A.

5.2 Unconstrained APGLP

Figure 9 shows the performance of the Advantage and Advantage2 Prototype for the (general version of) APGLP. In all cases, the Advantage2 Prototype surpasses the Advantage, and most configurations' performance gap between the solvers is significant. As the graph order increases to six, the performance of both solvers substantially drops. Advantage fails to find any optimal solutions. Advantage2 Prototype achieves a meager percentage of optimal solutions for 1 000 reads with 700μs annealing time, 2 000 reads with 300 annealing time, and 3 000 reads with 100μs annealing time. Advantage2 Prototype fails to provide optimal solutions for 5 000 reads with 20μs annealing time.

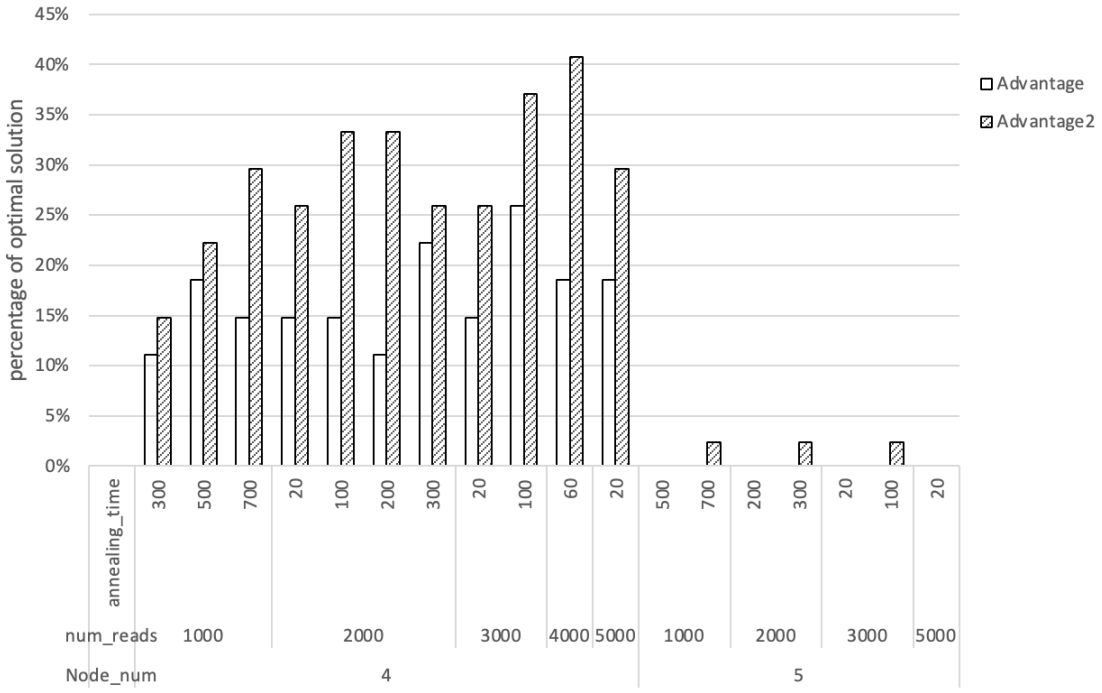
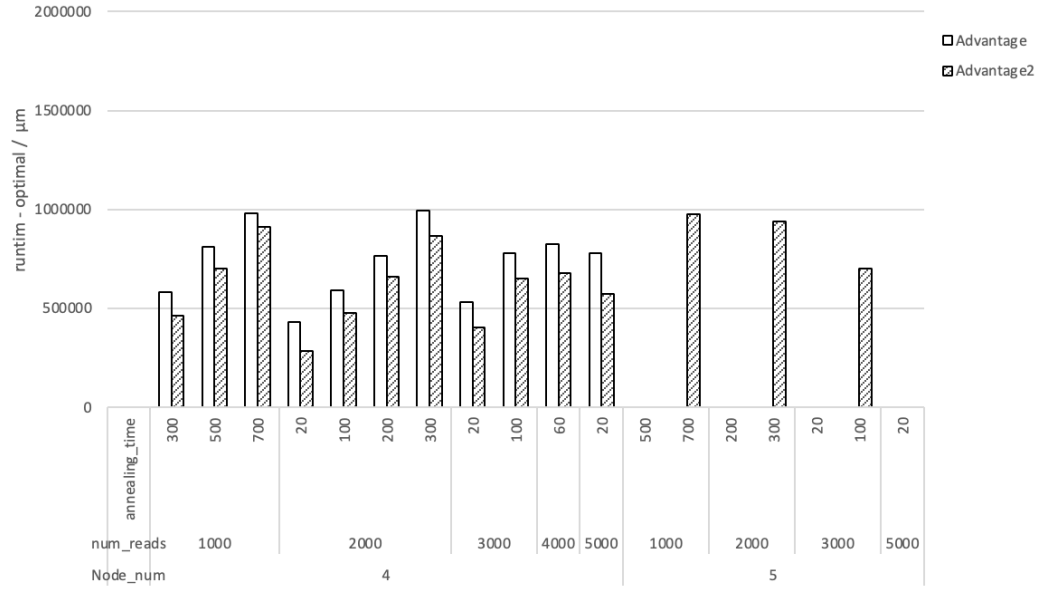


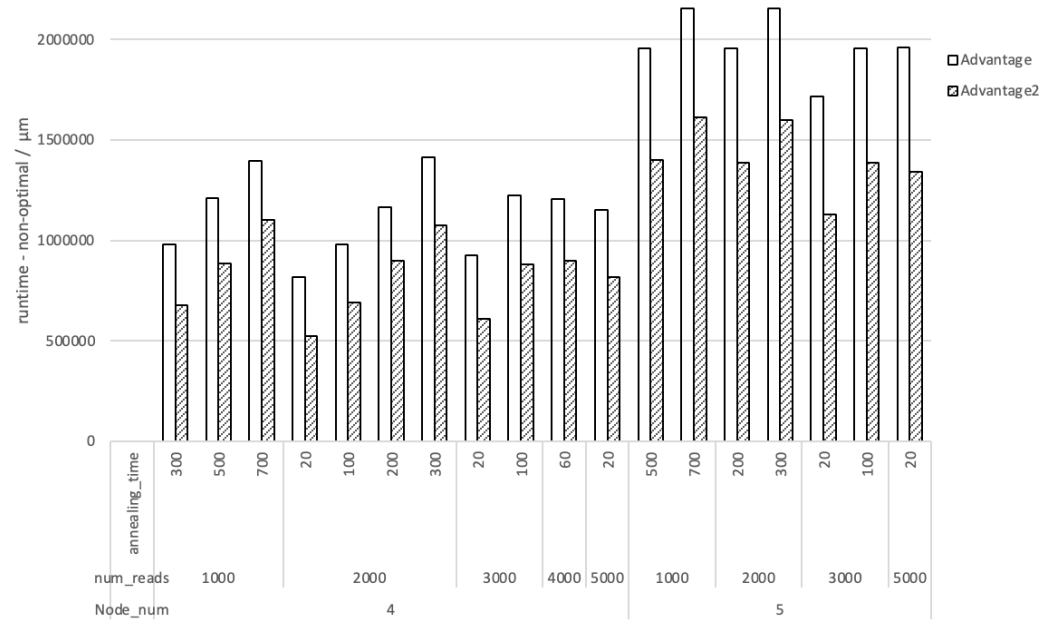
Figure 9: Percentage of optimal solutions achieved by the Advantage and the Advantage2 Prototype for problem graphs with four and five nodes.

The two bar graphs in Figure 10 display the runtimes for graphs of four and five nodes. In both charts, Advantage often requires more time than Advantage2 Prototype. The Advantage system fails to provide optimal solutions for all five-node graphs, whereas the Advantage2 Prototype succeeds in finding optimal results given a longer annealing time.

Figure 11 displays the embedding times of Advantage and Advantage2 Prototype for problem graphs with four to eight nodes. The Advantage2 Prototype recorded shorter embedding times for graphs with nodes less than seven, and both solvers recorded comparable embedding times for graphs with seven and eight nodes. For problem graphs with eight nodes, the Advantage2 Prototype has a longer runtime than the Advantage.



(a) Runtimes for achieving optimal solutions with Advantages and Advantage2 Prototype.



(b) Runtimes for obtaining non-optimal solutions with Advantages and Advantage2 Prototype.

Figure 10: Runtimes for problem graphs of four and five nodes, differentiated by the number of reads and annealing times.

The QPU access times for the Advantage and the Advantage2 Prototype for graphs with four and five nodes are compared in Figure 12. The Advantage2 Prototype used insignificantly shorter QPU access times than the Advantage for both graphs, with four and five nodes for all sampling ranges.

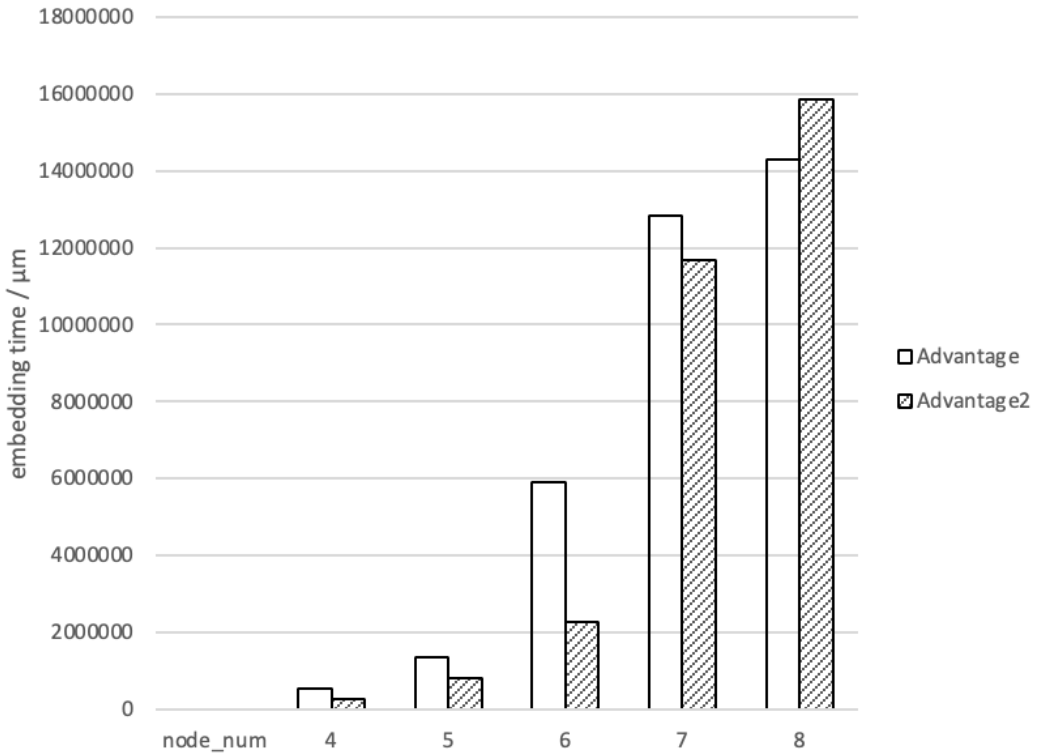


Figure 11: Embedding times recorded by Advantage and Advantage2 Prototype for graphs with four to eight nodes.

Table 3 compares the Hybrid solver and the MIP method. For graphs with four to six nodes, the Hybrid solver found all optimal solutions within a time limit of 300 000 microseconds. For graphs with seven and eight nodes, the performance of the Hybrid solver decreases to a range of 3.33% to 13.33% with time limits ranging from 6 000 000 to 18 000 000. The MIP method consistently achieves optimal solutions 100% of the time for all node sizes. However, the runtime for the MIP method increases rapidly with the increase in node size. Nevertheless, the Hybrid solver maintains a probability of obtaining optimal solutions for some problems within shorter time frames than the time required by the MIP method.

Table 4 displays the mean count of logical qubits for problem graphs with four to six nodes. The calculation of logical qubits for the edge labeling problem involves multiplying the number of binary variables needed to denote the maximum edge label by the total number of graph edges. The average results showing the performance metrics for the APGLP can be found in Table 6 in Appendix A.

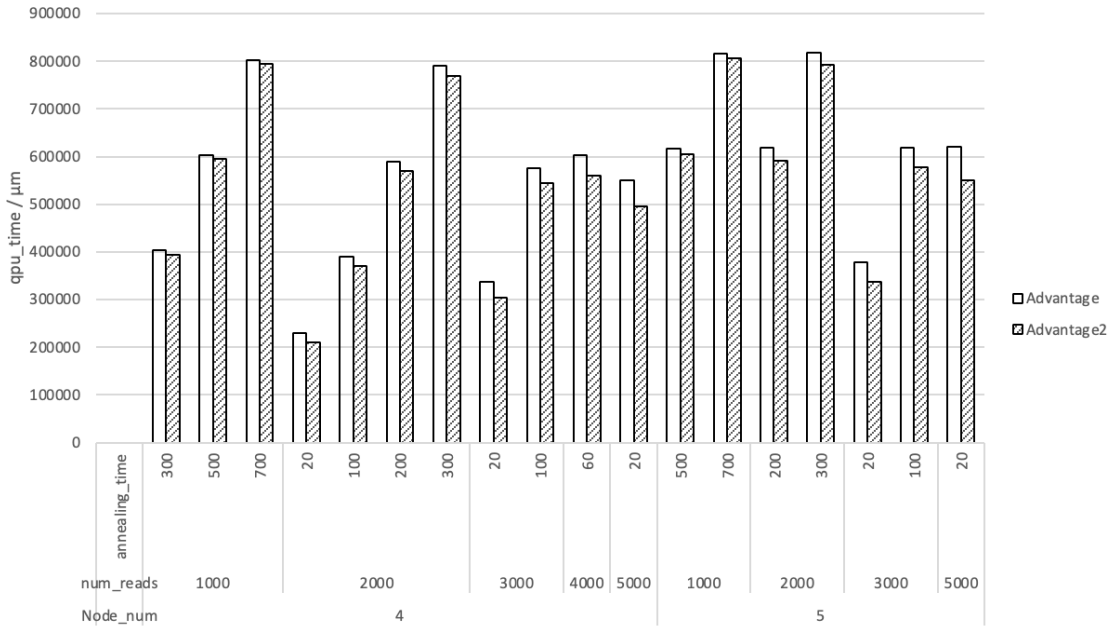


Figure 12: QPU access times for the Advantage and the Advantage2 Prototype quantum solvers for graphs with four and five nodes.

Table 3: A comparison of the Hybrid solver and the MIP method showing the percentage of optimal solutions for graphs with four to eight nodes. This comparison includes the performance of the Hybrid solver within its specified time limits and the average runtimes of the MIP method to obtain optimal solutions.

	Hybrid	Hybrid	MIP	MIP
Number of nodes	Time limit μs	Percentage optimal	MIP runtime μs	Percentage optimal
4	3 000 000	100.00%	4 848.96	100%
5	3 000 000	100.00%	19 087.90	100%
6	3 000 000	100.00%	91 204.77	100%
7	3 000 000	33.33%	864 160.38	100%
7	6 000 000	43.33%		
8	6 000 000	3.33%		
8	9 000 000	10.00%		
8	12 000 000	13.33%		
8	15 000 000	6.67%		
8	18 000 000	6.67%	17 473 136	100%

Table 4: Average number of logical qubits required for formulating the QUBO graphs.

Graph order	Fixed Vertex Labeled APGLP	APGLP
4	15	31
5	22	47
6	35	71

6 Conclusions

The results presented in this paper show that the D-Wave Advantage and Advantage2 Prototype are highly effective in solving the Fixed Vertex Labeled APGLP for small-scale problems. Not unexpectedly, as the graph complexity increased, the performance decreased.

The Advantage performed better than the Advantage2 for the Fixed Vertex Labeled APGLP, while the Advantage2 outperformed the Advantage for the APGLP. The advantages of HSS and MIP methods have been demonstrated for both runtime and quality of solutions.

The performance of the HSS showed the potential of combining classical and quantum computing, which seems to be an efficient and promising use of quantum computers. This approach could be done by developing Hybrid (quassical [1]) solutions for classes of problems.

References

[1] Edward H. Allen and Cristian S. Calude. Quassical computing. *International Journal of Unconventional Computing*, 14:43–57, 2018.

[2] Kelly Boothby, Andrew D. King, and Jack Raymond. Zephyr topology of D-Wave quantum processors. Technical Report 14-1056A-A, D-Wave Systems Inc., Burnaby, BC, Canada, 2021.

[3] Jun Cai, William G Macready, and Aidan Roy. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741*, 2014.

[4] Costantino Carugno, Maurizio Ferrari Dacrema, and Paolo Cremonesi. Evaluating the job shop scheduling problem on a d-wave quantum annealer. *Scientific Reports*, 12(1):6539, 2022.

[5] D-Wave Systems Inc. Whitepaper: Programming the D-Wave QPU: Setting the chain strength. Technical Report 14-1041A-A, Burnaby, BC, Canada, 2020.

[6] D-Wave Systems Inc. Whitepaper: Early progress on lower-noise fabrication development for the future, full-scale advantage2 quantum computer. Technical Report 09-1287A-A, Burnaby, BC, Canada, 2022.

[7] D-Wave Systems Inc. D-Wave announces availability of 1,200+ qubit Advantage2 prototype in the leap quantum cloud service, making its most performant system available to customers today . https://www.dwavesys.com/company/newsroom/press-release/d-wave-announces-availability-of-1-200-qubit-advantage2-prototype/, Feb. 12, 2024.

[8] Prasanna Date, Davis Arthur, and Lauren Pusey-Nazzaro. QUBO formulations for training machine learning models. *Scientific reports*, 11(1):10029, 2021.

[9] Bascom S Deaver Jr and William M Fairbank. Experimental evidence for quantized flux in superconducting cylinders. *Physical Review Letters*, 7(2):43, 1961.

[10] Michael J Dinneen, Nan Rosemary Ke, and Masoud Khosravani. Arithmetic progression graphs. Technical Report CDMTCS-356, Department of Computer Science, The University of Auckland, New Zealand, 2009.

[11] Michael J Dinneen, Nan Rosemary Ke, and Masoud Khosravani. Arithmetic progression graphs. *Universal Journal of Applied Mathematics*, 2(8):290 – 297, 2014.

[12] Robert Doll and Martin Näbauer. Experimental proof of magnetic flux quantization in a superconducting ring. *Physical Review Letters*, 7(2):51, 1961.

[13] Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition, 1979.

[14] Kazuki Ikeda, Yuma Nakamura, and Travis S Humble. Application of quantum annealing to nurse scheduling problem. *Scientific reports*, 9(1):12837, 2019.

[15] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[16] James King, Sheir Yarkoni, Mayssam M Nevisi, Jeremy P Hilton, and Catherine McGeoch. Benchmarking a quantum annealing processor with the time-to-target metric. *arXiv preprint arXiv:1508.05087*, 2015.

[17] Catherine McGeoch and Pau Farré. Advantage processor overview. Technical Report 14-1058A-A, D-Wave Systems Inc., Burnaby, BC, Canada, 2022.

[18] Catherine McGeoch, Pau Farré, and William Bernoudy. D-Wave hybrid solver service + Advantage: Technology update. Technical Report 14-1048A-A, D-Wave Systems Inc., Burnaby, BC, Canada, 2020.

[19] Catherine McGeoch, Pau Farré, and Kelly Boothby. The d-wave advantage2 prototype. Technical Report 14-1063A-A, D-Wave Systems Inc., Burnaby, BC, Canada, 2022.

[20] Brendan D. McKay. Graphs. http://users.cecs.anu.edu.au/~bdm/data/graphs.html.

[21] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[22] Florian Richoux, Jean-François Baffier, and Philippe Codognet. Learning QUBO models for quantum annealing: A constraint-based approach. In *International Conference on Computational Science*, Berlin, Heidelberg, 2023. Springer-Verlag.

[23] Hayato Ushijima-Mwesigwa, Christian FA Negre, and Susan M Mniszewski. Graph partitioning using quantum annealing on the D-wave system. In *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*, pages 22–29. Association for Computing Machinery, 2017.

A Average Results

Table 5: Average results for Fixed Vertex Labeled APGLP – Advantage Systems

Nodes	Number of reads	Annealing time	Advantage optimal %	Advantage2 optimal %
4	1 000	20	96.30%	92.59%
4 Total			96.30%	92.59%
5	1 000	20	76.19%	64.29%
5	1 000	100	78.57%	66.67%
5	1 000	300	78.57%	71.43%
5	1 000	500	69.05%	76.19%
5	1 000	700	73.81%	71.43%
5 SubTotal	1 000		75.24%	70.00%
5	2 000	20	76.19%	76.19%
5	2 000	100	76.19%	71.43%
5	2 000	200	83.33%	80.95%
5	2 000	300	80.95%	69.05%
5 SubTotal	2 000		79.17%	74.40%
5	3 000	20	80.95%	73.81%
5	3 000	100	83.33%	76.19%
5	3 000	200	N/A	71.43%
5 SubTotal	3 000		82.14%	73.81%
5	5 000	20	85.71%	78.57%
5 Total			78.57%	72.89%
6	1 000	20	10.00%	3.33%
6	1 000	100	0.00%	0.00%
6	1 000	300	3.33%	6.67%
6	1 000	500	3.33%	0.00%
6	1 000	700	3.33%	0.00%
6 SubTotal	1 000		4.00%	2.00%
6	2 000	20	10.00%	3.33%
6	2 000	100	0.00%	3.33%
6	2 000	200	6.67%	3.33%
6	2 000	300	3.33%	10.00%
6 SubTotal	2 000		5.00%	5.00%
6	3 000	20	13.33%	6.67%
6	3 000	100	3.33%	0.00%
6	3 000	200	N/A	10.00%
6 SubTotal	3 000		8.33%	5.56%
6	5 000	20	13.33%	0.00%
6 Total			5.83%	3.59%
Grand Total			49.72%	45.38%

Table 6: Average results for APGLP – Advantage Systems

Nodes	Number of reads	Annealing time	Advantage optimal %	Advantage2 optimal %
4	1 000	300	11.11%	14.81%
4	1 000	500	18.52%	22.22%
4	1 000	700	14.81%	29.63%
4 SubTotal	1 000		14.81%	22.22%
4	2 000	20	14.81%	25.93%
4	2 000	100	14.81%	33.33%
4	2 000	200	11.11%	33.33%
4	2 000	300	22.22%	25.93%
4 SubTotal	2 000		15.74%	29.63%
4	3 000	20	14.81%	25.93%
4	3 000	100	25.93%	37.04%
4 SubTotal	3 000		20.37%	31.48%
4	4 000	60	18.52%	40.74%
4	5 000	20	18.52%	29.63%
4 Total			16.84%	28.96%
5	1 000	500	0.00%	0.00%
5	1 000	700	0.00%	2.38%
5 SubTotal	1 000		0.00%	1.19%
5	2 000	200	0.00%	0.00%
5	2 000	300	0.00%	2.38%
5 SubTotal	2 000		0.00%	1.19%
5	3 000	20	0.00%	0.00%
5	3 000	100	0.00%	2.38%
5 SubTotal	3 000		0.00%	1.19%
5	5 000	20	0.00%	0.00%
5 Total			0.00%	1.02%
Grand Total			8.46%	15.06%

B Python codes

B.1 QUBO formulation for Fixed Vertex Labeled APGLP

```
def qubo_formulation_fixed (G, a, d, v_labels):
  start_time = time.process_time()
  n = G.order()
  m = G.size()

  #z2 is the number of binary bits to
  #represent the maximum edge value possible
  z = a + (n-2)*d
  z2 = math.ceil(math.log2(z))

  i_m = nx.incidence_matrix(G).toarray()
  Q = {}
  offset = 0
  #F1^2
  #for each vertex:
  for i in range(n):
    #degree
    delt_v = sum(i_m[i])
    offset += (delt_v - v_labels[i])**2
    #F1^2
    for j in range(m):
      mij = i_m[i][j]
      if mij == 1:
        for k in range(z2):
          #linear part
          if Q.get((f'y{j}{k}',f'y{j}{k}')) == None:
            Q[(f'y{j}{k}',f'y{j}{k}')]= \
            (2**k)**2+2*(delt_v-v_labels[i])*(2**k)
          else:
            Q[(f'y{j}{k}',f'y{j}{k}')]+= \
            (2**k)**2+2*(delt_v-v_labels[i])*(2**k)

          #quadratic part
          for jj in range(j,m):
            mijj=i_m[i,jj]
            if mijj == 1:
              for kk in range(z2):
                if k<kk <= z2 or j<jj<=m:
                  if Q.get((f'y{j}{k}',f'y{jj}{kk}')) == None:
                    Q[(f'y{j}{k}',f'y{jj}{kk}')] = 2*(2**k)*(2**kk)
                  else:
                    Q[(f'y{j}{k}',f'y{jj}{kk}')] += 2*(2**k)*(2**kk)

  #in microseconds
  elapsed_time = (time.process_time() - start_time)*(10**6)
  return Q,  offset, elapsed_time

def write_qubos_fixed(input_file,output_file):
  qubos = []
  formulation_time_list = []
```

```
    offsets = []
    new_df = pd.DataFrame()
    #new_df = pd.read_csv(output_file)
    df = pd.read_csv(input_file)
    n = len(df.index)
    randomList = random.sample(range(0, n),30)
    randomList.sort()
    for i in randomList:
      a = df['a'][i]
      d = df['d'][i]
      #a_list.append(a)
      #d_list.append(d)
      #ess.append(df['graph'][i])
      ad_str = df['adjacency_list'][i]
      ad_list=json.loads(ad_str)
      ad = pd.DataFrame(ad_list)
      G = nx.from_pandas_adjacency(ad)
      vlabels = df['vlabels'][i]
      v_labels = ast.literal_eval(vlabels)

      #qubo formulation 5 times to get the average time it takes
      n = 5
      times = 0
      for i in range(n):
        Q,  offset, formulation_time = \
        qubo_formulation_fixed(G, a, d, v_labels)
        times += formulation_time

      qubos.append(Q)
      formulation_time_list.append(times/n)
      offsets.append(offset)
    new_df['graph_num'] = randomList
    new_df['qubo_fixed'] = qubos
    new_df['offset'] = offsets
    new_df['formulation_time/microseconds'] = formulation_time_list
    new_df.to_csv(output_file)
```

B.2 QUBO formulation for the APGLP

```
def qubo_formulation(G, a, d):
    start_time = time.process_time()
    n = G.order()
    m = G.size()
    #maximum number of binary bits needed to represent any edge
    z = a + (n-2)*d
    z2 = math.ceil(math.log2(z))

    i_m = nx.incidence_matrix(G).toarray()

    Q = {}
    offset = n
    #for each vertex:
```

```
for i in range(n):
  #degree
  degree = sum(i_m[i])
  offset += (degree - a)**2
  #F1^2
  for j in range(m):
    mij = i_m[i][j]
    if mij == 1:
      for k in range(z2):
        #linear part
        if Q.get((f'y{j}{k}',f'y{j}{k}')) == None:
          Q[(f'y{j}{k}',f'y{j}{k}')]= (2**k)**2-\
          2*a*(2**k)+2*degree*(2**k)
        else:
          Q[(f'y{j}{k}',f'y{j}{k}')]+= (2**k)**2-\
          2*a*(2**k)+2*degree*(2**k)

        #quadratic part
        for jj in range(j,m):
          mijj=i_m[i,jj]
          if mijj == 1:
            for kk in range(z2):
              if k<kk <= z2 or j<jj<=m:
                if Q.get((f'y{j}{k}',f'y{jj}{kk}')) == None:
                  Q[(f'y{j}{k}',f'y{jj}{kk}')] = 2*(2**k)*(2**kk)
                else:
                  Q[(f'y{j}{k}',f'y{jj}{kk}')] += 2*(2**k)*(2**kk)

  for h in range(n):
    #F2^2
    if Q.get((f'T{i}{h}',f'T{i}{h}')) == None:
      Q[(f'T{i}{h}',f'T{i}{h}')] = d**2*h**2 + 2*a*d*h - 2*h*d*degree
    for h2 in range(h+1,n):
      Q[(f'T{i}{h}',f'T{i}{h2}')] =  2*d**2*h*h2

    #-2F1F2
    for j in range(m):
      mij = i_m[i][j]
      if mij == 1:
        for k in range(z2):
          Q[(f'y{j}{k}',f'T{i}{h}')] = -2*h*2**k*d

#Constraints P
for i in range(n):
  for h in range(n):
    for h2 in range(h+1,n):
      if Q.get((f'T{i}{h}',f'T{i}{h2}')) == None:
        Q[(f'T{i}{h}',f'T{i}{h2}')] = 1
      else:
        Q[(f'T{i}{h}',f'T{i}{h2}')] += 1
for i in range(n):
  for h in range(n):
    for i2 in range(i+1,n):
      if Q.get((f'T{i}{h}',f'T{i2}{h}')) == None:
```

```
            Q[(f'T{i}{h}',f'T{i2}{h}')] = 1
        else:
            Q[(f'T{i}{h}',f'T{i2}{h}')] += 1
    for i in range(n):
        for h in range(n):
            if Q.get((f'T{i}{h}',f'T{i}{h}')) == None:
                Q[(f'T{i}{h}',f'T{i}{h}')] = -1
            else:
                Q[(f'T{i}{h}',f'T{i}{h}')] += -1
    elapsed_time = (time.process_time() - start_time)*(10**6)
    return Q, offset, elapsed_time

def write_qubos(input_file, output_file):
    qubos = []
    offsets = []
    formulation_time_list = []
    graph = []
    a_list=[]
    d_list =[]
    df = pd.read_csv(input_file)
    new_df = pd.DataFrame()
    n = len(df.index)

    for i in range(n):
        times = 0
        a = df['a'][i]
        d = df['d'][i]
        ad_str = df['adjacency_list'][i]
        ad_list=json.loads(ad_str)
        df_ad = pd.DataFrame(ad_list)
        G = nx.from_pandas_adjacency(df_ad)
        #take the average time among 5 times of formulations
        n = 5
        for i in range(n):
            Q, offset, formulation_time = qubo_formulation(G,a,d)
            times += formulation_time
        g = df['graph'][i]
        qubos.append(Q)
        offsets.append(offset)
        formulation_time_list.append(times/n)
        graph.append(g)
        a_list.append(a)
        d_list.append(d)

    new_df['a'] = a_list
    new_df['d'] = d_list
    new_df['graph'] = graph
    new_df['qubo'] = qubos
    new_df['offset'] = offsets
    new_df['qubo_formulation_time/microseconds'] = \
    formulation_time_list
    new_df.to_csv(output_file)
```

B.3 Embedding QUBOs on D-Wave hardware

```
def find_embedding_pegasus(file_name, col_name):
  solver = DWaveSampler()
  df = pd.read_csv(file_name)
  A_pegasus = solver.edgelist
  pegasus_graph = dnx.pegasus_graph(16,edge_list=A_pegasus)
  embeddings = []
  times = []
  n = len(df.index)
  for i in range(n):
    qubo_str = df[col_name][i]
    qubo = ast.literal_eval(qubo_str)
    start_time = time.process_time()
    embedding_pegasus =  minorminer.find_embedding(qubo,pegasus_graph)
    elapsed_time = (time.process_time() - start_time)*(10**6)
    embeddings.append(embedding_pegasus)
    times.append(elapsed_time)
    #print(i,'/',n)
  df['pegasus_embedding'] = embeddings
  df['pegasus_embedding_time'] = times

  df.to_csv(file_name)

def find_embedding_zephyr(file_name, col_name):
  solver = DWaveSampler()
  df = pd.read_csv(file_name)
  A_zephyr = solver.edgelist
  zephyr_graph = dnx.zephyr_graph(6,edge_list=A_zephyr)
  embeddings = []
  times = []
  n = len(df.index)
  for i in range(n):
    qubo_str = df[col_name][i]
    qubo = ast.literal_eval(qubo_str)
    start_time = time.process_time()
    embedding_zephyr = minorminer.find_embedding(qubo,zephyr_graph)
    elapsed_time = (time.process_time() - start_time)*(10**6)
    embeddings.append(embedding_zephyr)
    times.append(elapsed_time)
    #print(i,'/',n)
  df['zephyr_embedding'] = embeddings
  df['zephyr_embedding_time'] = times
  df.to_csv(file_name)
```

B.4 Python code for running on Advantage QPUs

```
def create_bqm(Q, offset):
  BQM = dimod.BinaryQuadraticModel.empty(dimod.BINARY)
  for (v1,v2), value in Q.items():
    if v1 == v2:
      BQM.add_linear(v1,value)
    else:
```

```
        BQM.add_quadratic(v1,v2,value)
    BQM.offset += offset
    return BQM

def advantage(input_file, output_file, qubo_col, embedding_col,
    num_reads, annealing_time):
    solver = DWaveSampler()
    df = pd.read_csv(input_file)
    n = len(df.index)
    if Path(output_file).is_file():
        new_df = pd.read_csv(output_file)
        n = len(new_df.index)
    else:
        new_df = pd.DataFrame()
        if n==30:
            new_df['graph_num'] = df['graph_num']
    sample = []
    qpu_access_time = []
    optimal = []

    count = 0

    for i in range(n):
        opt = 'N'
        qubo_str = df[qubo_col][i]
        qubo = ast.literal_eval(qubo_str)
        embedding_str = df[embedding_col][i]
        embedding = ast.literal_eval(embedding_str)
        offset = df['offset'][i]

        BQM = create_bqm(qubo, offset)

        sampler = FixedEmbeddingComposite(solver, embedding)
        sampleset = sampler.sample(BQM, num_reads=num_reads, annealing_time=
            annealing_time)
        qpu_time = sampleset.info['timing']['qpu_access_time']
        s = sampleset.first[0]
        qpu_access_time.append(qpu_time)
        sample.append(s)
        energy = sampleset.first[1]
        if energy == 0:
            opt = 'Y'
            count+=1
        optimal.append(opt)

    new_df[f'sample_{num_reads}_{annealing_time}'] = sample
    new_df[f'qpu_access_time_{num_reads}_{annealing_time}'] =
        qpu_access_time
    new_df[f'optimal_{num_reads}_{annealing_time}'] = optimal
    #print(count/n)
    new_df.to_csv(output_file)
```

B.5 Python code to run on HSS

```
def hybrid_run(Q, offset):
  result = 'N'
  BQM = dimod.BinaryQuadraticModel(dimod.BINARY)
  for (v1,v2), value in Q.items():
    if v1 == v2:
      BQM.add_linear(v1,value)
    else:
      BQM.add_quadratic(v1,v2,value)
  BQM.offset += offset
  sampler = LeapHybridSampler()
  sampleset = sampler.sample(BQM)
  run_time = sampleset.info['run_time']
  energy = sampleset.first[1]
  qpu_time = sampleset.info['qpu_access_time']
  sample = sampleset.first[0]
  if energy == 0:
    result = 'Y'
  return result, run_time, qpu_time, sample

def hybrid_run_qubo(Q, offset,time_limit):
  result = 'N'
  sampleset = LeapHybridSampler().sample_qubo(Q,time_limit=time_limit)
  run_time = sampleset.info['run_time']
  energy = sampleset.first[1]
  qpu_time = sampleset.info['qpu_access_time']
  sample = sampleset.first[0]
  if energy == -offset:
    result = 'Y'
  #print(result)
  return result, run_time, qpu_time, sample

def hybrid(input_file, output_file,qubo_col,time_limit):
  df = pd.read_csv(input_file)
  n = len(df.index)
  if Path(output_file).is_file():
    new_df = pd.read_csv(output_file)
    n = len(new_df.index)
  else:
    new_df = pd.DataFrame()
    if n==30:
      new_df['graph_num'] = df['graph_num']
  hybrid_optimal = []
  hybrid_runtime = []
  hybrid_qpu_access_time = []
  samples = []
  for i in range(len(df.index)):
    qubo_str = df[qubo_col][i]
    Q = ast.literal_eval(qubo_str)
    offset = df['offset'][i]
    result, runtime, qpu_time, sample = hybrid_run_qubo(Q, offset,
        time_limit)
    hybrid_optimal.append(result)
```

```
    hybrid_runtime.append(runtime)
    hybrid_qpu_access_time.append(qpu_time)
    samples.append(sample)

new_df[f'optimal_{time_limit}'] = hybrid_optimal
new_df[f'total_runtime_{time_limit}'] = hybrid_runtime
new_df[f'qpu_access_time_{time_limit}'] = hybrid_qpu_access_time
new_df[f'sample_{time_limit}'] = samples
new_df.to_csv(output_file)
```

B.6 Python MIP code for APG graph generalizing from g6 file

```
def read_graphs_from_g6(graph_name):

  G_l = nx.read_graph6(graph_name)
  graphs = []
  intlist = random.sample(range(len(G_l)),10)
  for ii in intlist:
    g = G_l[ii]

    i = True
    #print(ii)
    if ii != 0 and len(graphs) != 0:
      for g2 in graphs:
        if nx.is_isomorphic(g, g2) and len(graphs)!=1:
          i = False
    if i:
      s = g.adjacency()
      listl = []
      for node, ad in s:
        l = []
        for i in range (g.number_of_nodes()):
          if i in list(ad.keys()):
            l.append(1)
          else:
            l.append(0)
        listl.append(l)
      graphs.append(g)

  return graphs

def pythonmip (G, a,d,col_name):

  print('fsdfds')
  n=G.order()
  m=G.size()

  const = a+(n-2)*d
  a_list = []
  list2=[]

  for i in range(n):
    a_list.append(a + i*d)
```

```
    list2.append(0)

#print('running:', a, d)
for p in permutations(range(n)):
  #vertex labels
  B = [a+(p[i])*d for i in range(n)]
  p=Model(sense='MIN', solver_name='CBC')
  b = { (u, v): p.add_var(var_type=INTEGER, name= f"b_{u}_{v}") \
      for (u, v) in G.edges() }

  for i in range(n):
    #sum of all edge of vertex i
    edge_sum = xsum(b[(u, v)] for (u, v) in G.edges()
    if i == int(u) or i == int(v))
    #add constrains to model
    p += edge_sum == B[i]
  p.objective = xsum(b[x] for x in G.edges())
  for e in G.edges():
    p.add_constr(b[e]>=1)

  # Solve the model
  p.verbose = 0
  p.optimize()

  # Check if a solution was found'  "
  if p.status == OptimizationStatus.OPTIMAL:
    solution = {var.name: var.x for var in b.values()}
    for (edge, value) in solution.items():
      # vertex 1
      e1 = int(edge[2])
      # vertex 2
      e2 = int(edge[4])
      # set edge weight in G
      G[e1][e2]['weight'] = value
    ad_list = nx.adjacency_matrix(G, weight=None).toarray().tolist()
    weight = nx.get_edge_attributes(G, 'weight')
    return { col_name[0]: ad_list, col_name[1]: a, col_name[2]: d, \
             col_name[3]: 'Y', col_name[4]: G, col_name[5]: weight }
  elif p.status == OptimizationStatus.FEASIBLE:
    # if a solution exists
    solution = {var.name: var.x for var in b.values()}
    for (edge, value) in solution.items():
      #vertex 1
      e1 = int(edge[2])
      #vertex 2
      e2 = int(edge[4])
      #set edge weight in G
      G[e1][e2]['weight'] = value
      #all vertex values in list2 by adding all its
      #edges values
      list2[e1] += value
      list2[e2] += value
    #check if the vertex values illegal
    if sorted(a_list) == sorted(list2):
```

```
        #print('uuu')
        ad_list=nx.adjacency_matrix(G, weight=None).toarray().tolist()
        weight = nx.get_edge_attributes(G, 'weight')
        return {col_name[0]: ad_list, col_name[1]: a, col_name[2]: d,\
                col_name[3]:'Y', col_name[4]:G, col_name[5]:weight}
    return 0

def find_apgs (a_min, a_max, d_min, d_max, graphs, df, col_name):

  all = len(graphs)
  count = 0
  for G in graphs:
    count += 1
    #print(count,'/',all)
    #print('graph')
    for aa in range(a_min, a_max):
      for dd in range(d_min, d_max):
        df2 = pythonmip(G, aa, dd, col_name)
        if df2 != 0 and df2 != None:
          df = pd.concat([df, pd.DataFrame([df2])],ignore_index=True)
  return df

def mip_find_apgs(graph_file_name, df_file_name):
  col_name = ['adjacency_list', 'a', 'd', 'APG_label_availability', \
              'graph', 'weight']
  dict = {col_name[0]:[],
      col_name[1]:[],
      col_name[2]:[],
      col_name[3]:[],
      col_name[4]:[],
      col_name[5]:[]}
  df = pd.DataFrame(dict)

  # set the range of value a and d here
  a_min = 2
  a_max = 4
  d_min = 1
  d_max = 4

  graphs = read_graphs_from_g6(graph_file_name)
  df = find_apgs(a_min, a_max, d_min, d_max, graphs, df, col_name)
  df.to_csv(df_file_name)
```

B.7 Python MIP code for running APGLP

```
def mip_a_d (input_file, output_file):

  df = pd.read_csv(input_file)
  graphs = []
  optimal = []
  a_value = []
  d_value = []
  weights = []
```

```
runtime = []
n=len(df.index)
for i in range(n):
  a = df['a'][i]
  d = df['d'][i]
  ad_str = df['adjacency_list'][i]
  ad_list=json.loads(ad_str)
  ad = pd.DataFrame(ad_list)
  graph = nx.from_pandas_adjacency(ad)
  result = []
  result = mip(graph, a, d)
  if result != 0:
    graph = result[0]
    a = result[1]
    d = result[2]
    weight = result[3]
    time = result[4]
    graphs.append(ad_list)
    a_value.append(a)
    d_value.append(d)
    weights.append(weight)
    runtime.append(time)
    optimal.append('Y')
  else:
    graphs.append(0)
    a_value.append(0)
    d_value.append(0)
    weights.append(0)
    runtime.append(0)
    optimal.append('N')
new_df = pd.DataFrame()
new_df['graph_num'] = df['graph_num']
new_df['graph'] = graphs
new_df['a'] = a_value
new_df['d'] = d_value
new_df['weights'] = weights
new_df['runtime'] = runtime
new_df['optimal'] = optimal
new_df.to_csv(output_file)
```