mathematics

IMPACT FACTOR 2.592

CITESCORE 3.5

Article

Finding Debt Cycles: QUBO Formulations for the Maximum Weighted Cycle Problem Solved Using Quantum Annealing

Hendrik Künnemann and Frank Phillipson

Special Issue
Advances in Quantum Computing and Applications

Edited by
Prof. Dr. Frank Phillipson, Dr. Sebastian Feld, Dr. Matthias Möller, Ward van der Schoot and Niels Neumann

MDPI

https://doi.org/10.3390/math11122741

mathematics

MDPI

Article

Finding Debt Cycles: QUBO Formulations for the Maximum Weighted Cycle Problem Solved Using Quantum Annealing

Hendrik Künnemann[1] and Frank Phillipson [1,2,*]

[1] School of Business and Economic, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands
[2] Department of Applied Cryptography & Quantum Algorithms, TNO, P.O. Box 96800, 2509 JE Den Haag, The Netherlands
* Correspondence: f.phillipson@maastrichtuniversity.nl

Abstract: The problem of finding the maximum weighted cycle in a directed graph map to solve optimization problems is $\mathcal{NP}$-hard, implying that approaches in classical computing are inefficient. Here, Quantum computing might be a promising alternative. Many current approaches to the quantum computer are based on a Quadratic Unconstrained Binary Optimization (QUBO) problem formulation. This paper develops four different QUBO approaches to this problem. The first two take the starting vertex and the number of vertices used in the cycle as given, while the latter two loosen the second assumption of knowing the size of the cycle. A QUBO formulation is derived for each approach. Further, the number of binary variables required to encode the maximum weighted cycle problem with one or both assumptions for the respective approach is made explicit. The problem is motivated by finding the maximum weighted debt cycle in a debt graph. This paper compares classical computing versus currently available (hybrid) quantum computing approaches for various debt graphs. For the classical part, it investigated the Depth-First-Search (DFS) method and Simulated Annealing. For the (hybrid) quantum approaches, a direct embedding on the quantum annealer and two types of quantum hybrid solvers were utilized. Simulated Annealing and the usage of the hybrid CQM (Constrained Quadratic Model) had promising functionality. The DFS method, direct QPU, and hybrid BQM (Binary Quadratic Model), on the other hand, performed less due to memory issues, surpassing the limit of decision variables and finding the right penalty values, respectively.

Keywords: QUBO; graph theory; maximum weighted cycle; debt graphs

MSC: 05C90; 05C45; 05-04; 91B80

Citation: Künnemann, H.; Phillipson, F. Finding Debt Cycles: QUBO Formulations for the Maximum Weighted Cycle Problem Solved Using Quantum Annealing. *Mathematics* **2023**, *11*, 2741. https://doi.org/10.3390/math11122741

Academic Editor: João Nuno Prata

Received: 18 May 2023
Revised: 12 June 2023
Accepted: 14 June 2023
Published: 16 June 2023

1. Introduction

The word 'debt' often has a negative connotation. Debt, however, is an essential tool for many individuals, companies, and even societies. Without debt, the only force that drives the economy is productivity growth. With debt, on the other hand, individuals, companies, etc., can fuel growth, as they can consume more than they produce. Using debt as a tool can therefore increase standards of living. A farmer could, for example, use debt to buy a tractor. This tractor helps the farmer to harvest fields faster and more efficiently, enabling the farmer to generate more income than before. With this additional income, the debtor can pay back its debt and enjoy a better quality of life [1]. This positive effect does not even have to be that immediate. Student loans are also considered 'good' debt presuming that in a later stage of life, the student can pay back the debt and again enjoy a better quality of life. The negative associations, however, also have their reasons. As aforementioned, consumers using debt can consume more than they can produce. This can lead to over-consumption and to situations where the debtor is not able to pay back the entire amount. Such over-consumption can lead to debt crises on an individual, organizational, and even governmental level. A well-known example in which the debtor was not able to pay back

its debt is the sovereign debt crisis Greece faced in the aftermath of the financial crisis in the years 2007/2008. Additionally, debt continues to play a central role present-day. During the COVID-19 pandemic, a new all-time high of global debt was reached [2,3]. It is clear that it is of high importance to use debt as a tool to boost the economy, but it should be the goal to keep the amount as low as possible in order to avoid falling into a debt crisis.

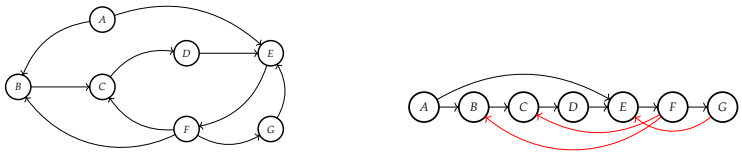
To model the complex debt relationships among entities, such as banks, governments, individuals, etc., it is convenient to use a directed graph in which all the debtors and creditors are represented by vertices. The weight of the arc, with the debtor being the source and the creditor the sink, stands for the amount of debt the debtor owes the respective creditor. Such a graph, DebtG, was introduced by Huanquing Cui [4], and this paper will make use of it. With this definition, it is possible that one debtor has multiple debts to the same creditor, but each debt has different properties, such as a different due date, a different interest rate, etc. This paper, however, uses a simplification by only looking at the amount of debt and disregarding all other properties. This leads to a graph in which there exists at most one arc between two entities, starting at the debtor and pointing towards the creditors' node. The amount of debt is then represented by the weight of the respective arc. We assume only positive weights here. If we model debt relations in such a way, a chain of debt relations that forms a cycle can be searched for. Once a cycle in the debt system is found, the settlement amount, i.e., the lowest amount of debt between a creditor and a debtor in the cycle, can be removed from the system without influencing anything else. Therefore, this approach can be used to remove unnecessary debt in the system, which is our goal. In terms of graph theory, this removal of unnecessary debts means that the particular settlement amount can be subtracted from each edge in the cycle. This modification could make the addressed graph more sparse and even disjointed. Finding the cycle with the highest amount is then equal to finding the Maximum Weighted Cycle (MWC) in a graph. Finding the MWC in a directed graph is $\mathcal{NP}$-hard [5], as a simple reduction from the well-known $\mathcal{NP}$-complete Hamiltonian Cycle Problem (HCP) can be used. A Hamiltonian Cycle is a cycle in a graph, directed or undirected, such that each vertex is visited exactly once. If the longest cycle of the graph is as large as the number of vertices of the graph, the graph has a Hamiltonian Cycle. Thus, the HCP is a special case of the longest cycle. As the longest cycle is the longest in terms of vertices used, this reduction is not yet complete. However, by setting the weights of the edges to one, we see that the MWC problem is a special case of the unweighted longest cycle problem. This, therefore, proves that the MWC problem is $\mathcal{NP}$-complete [6]. Because of the complexity of this problem, this paper suggests utilizing a combination of both classical and quantum computing, aiming for a decrease in computation time, as inspired by, e.g., [7–9]. Additionally, in the financial sector, first attempts are known [10,11]. For many current (hybrid) approaches in quantum computing, a Quadratic Unconstrained Binary Optimisation (QUBO) formulation of the problem is used. The aim of this paper is to provide multiple QUBO formulations for this problem and give an indication of the potential benefit that (hybrid) quantum computing can bring using the currently available hardware. McCollum and Krauss [12] have defined QUBO formulations for the longest path problem. However, we are, to the best of our knowledge, the first to propose this for the MWC problem.

This paper is organized as follows. First, Section 2 gives an overview of cycle-finding algorithms. Section 3 then introduces four QUBO formulations for the problem at hand. The first approach is about finding the MWC with a given starting vertex and using k vertices in total, while the second approach specifies $k = |V|$, where V is the set of vertices in the graph. For the other two approaches, the assumption of knowing the number of vertices used in the cycle will be loosened. In Section 4, we leave the theoretical part behind and focus on the implementation of one of these four approaches on the current (hybrid) quantum annealer. First, the focus is on the number of decision variables needed to encode the problem and on the penalty values that maximize the probability of finding good solutions. The following section, Section 5, compares the most efficient quantum approach with classical computing approaches. Section 6 concludes this paper and outlines potential directions for further research.

2. Literature Review

The simplest of all cycles in a directed graph, without self-loops and at most one arc between two vertices, consists of three vertices. Algorithms for finding those three-vertex cycles are called Triangle Algorithms. Classical triangle algorithms are mostly based on adjacency matrices or adjacency lists in combination with exhaustive search or simple matrix manipulations. Szegedy [13] came up with a respective quantum algorithm, but it is also possible to apply Grover's algorithm [14] in order to find triangles in a graph. Finding Quadrilaterals, cycles consisting of four vertices, is the following step. Again, there already exists both classical and quantum algorithms [15,16]. Simple modifications of the triangle algorithms are already sufficient. Generalizations of these algorithms followed by finding even/odd-k-cycles, where k is the number of vertices used [17–19].

As shown before, finding the MWC in general in a graph is $\mathcal{NP}$-complete. No known algorithm is asymptotically faster than trying out all (exponentially many) combinations, which is a viable approach for small instances, but quickly becomes computationally too expensive. For the comparison of quantum versus classical, one of the approaches used in Section 5 is a Depth-First-Search (DFS) algorithm. There, you are given a directed graph (e.g., Figure 1a), traverse it, and build a so-called DFS-tree (Figure 1b). The red arcs in Figure 1b are back arcs, and it is possible to find a cycle with each back arc. It is, however, known that this approach can use too much time [20].



(**a**) (**b**)

Figure 1. Depth-First-Search. (**a**) A random directed graph. (**b**) Respective DFS tree.

Another idea for finding all cycles is using topological ordering [21]. Bringing a graph into a topological order means that for every arc $a = (u, v)$, the vertex u is a parent vertex of its child vertex v. When a parent vertex is a child vertex at the same time, this already implies that there exists a cycle in the directed graph as this particular vertex 'breaks' the topological order. To reach the topological order, it is important that the given graph is directed. Two approaches for this are threading and ranking [22]. The Breadth-First Search (BFS) is one approach to finding cycles in a topologically ordered, directed graph. This traverses the graph and uses Boolean variables to mark nodes as visited or not-visited. The BFS algorithm, however, is often too memory-consuming in contrast to the DFS.

For large instances, approximation algorithms are used such that classical computers are still able to obtain reasonably good results. Bläser and Manthey [23] came up with a $\frac{2}{3}-$approximation algorithm.

Looking at algorithms for the MWC itself, Alon et al. [24] and Gabow and Nie [25] found a classical polynomial-time algorithm for directed cycles. The former finds a cycle with length of exactly $c \log(|V|)$ for any constant c, while the latter is of length $\frac{\log(|V|)}{\log(\log(|V|))}$, provided those exist. Björklund et al. [26], however, showed that there does not exist a polynomial time approximation within an approximation ratio of $|V|^{1-\epsilon}$ for any ϵ unless $\mathcal{P} = \mathcal{NP}$. Cui [4] aimed at finding all cycles in a directed graph with the use of classical computing but failed.

If we look at QUBO-related research, there is the research conducted by McCollumn et al. [12] that deals with finding the max-weighted path in a graph using k vertices. For this, they utilized two different approaches, the order-based and the degree-based formulation. For both approaches, they derived QUBOs. The same authors also derived QUBO formulations for the shortest path problem and the maximum flow problem [27,28]. Mahasinghe et al., and Nüßlein et al. [29,30] give a QUBO formulation for the Hamiltonian cycle problem and solve it using quantum annealing. As indicated, the Hamiltonian

cycle problem is a specific case of the MWC problem. Finally, also Rosenberg [31] says to investigate finding the negative cycle with the most negative weight of arbitrary length, which is NP-hard, as they say. However, in their implementation, they allow multiple cycles, solving a different problem, and finding cycle sets, which can be reduced to a minimum cost flow problem and is much easier to solve.

To conclude, it becomes clear that there does not yet exist a complete approach free of drawbacks, such as being time- or space-inefficient or only an approximation. This underlines the importance of this paper.

3. QUBO Formulations

In this section, we provide four QUBO formulations for the MWC problem. The problem of finding the MWC in the graph will be approached step by step. First, it will be assumed that the starting point is known, and the MWC using k vertices will be found. After that, we choose k equal to the total number of vertices, implying that all vertices in the graph have to be part of the cycle. Lastly, the assumption of knowing the number of vertices contained by the cycle is relaxed, while the assumption of knowing the starting point is maintained. Before writing the QUBO form, a basic integer linear program (ILP) is presented, which defines this problem. Afterward, the ILP is reformulated into a QUBO. First, we introduce the QUBO formulation itself.

3.1. Quadratic Unconstrained Binary Optimization

A Quadratic Unconstrained Binary Optimization (QUBO) problem is a problem in the field of Combinatorial Optimization. It is expressed as follows:

$$\min_{x \in \{0,1\}^n} \quad x'Qx, \tag{1}$$

where x is a n-dimensional vector of binary decision variables, so $x_i \in \{0,1\}\ \forall i$, and has to be set such that, given Q, the expression is minimized. Note that QUBO problems belong to the set of $\mathcal{NP}$-hard problems, and for many problems, the QUBO formulation, or the related Ising formulation, is known [32,33]. The matrix Q is the weight matrix containing the coefficients to encode the problem instance. Q is a real-valued square matrix. Without loss of generality, we can assume Q to be in the upper triangular form [33]. With this formulation, we either have off-diagonal terms ($q_{ij}x_ix_j, i \neq j$) or diagonal terms ($q_{ii}x_ix_i = q_{ii}x_i^2$). Solving this single line of Equation (1) solves the entire problem—there are no additional constraints as in, e.g., a Linear Programming Problem. Constraints from the original problem can be incorporated into the objective function using penalty coefficients that enforce the constraints to be fulfilled. Theoretically, lower bounds of the penalty values can be derived, guaranteeing that the optimal solution of the QUBO meets all the original constraints [32]. However, for the implementation of the quantum annealer, these values do not necessarily work, as we will show later.

Quantum annealing started with the work of Kadowaki and Nishimori [34] and is explained further in [35]. In quantum annealing, an equal superposition over all possible states is created. Then, in order to create the solution to the optimization problem, a problem-specific magnetic field is turned on. This causes the qubits to interact with each other and to move to the lowest energy state of the system. D-Wave Systems built one of the earliest and most advanced versions of the quantum annealer [36]. The most recent machine that is offered by D-Wave, via their LEAP environment is the D-Wave Advantage. This machine has more than 5000 qubits in the Pegasus architecture, having connectivity of at most 15. To solve a QUBO problem, it has to be embedded in the hardware. Each variable is translated to one more (a chain of) qubits. The reason that sometimes more qubits are needed for one variable is connectivity. However, the number of qubits and their connectivity has grown considerably; they are still of limited size. This means that often the problem can not be embedded in the chip and has to be decomposed. The most recent decomposition approaches D-Wave offers are D-Wave Hybrid and the Hybrid Solver

Service, offering more flexibility. Both the direct implementation of the QPU and the hybrid approaches are used in this work.

A classical method to solve QUBO problems that are also used in this study is Simulated Annealing. Simulated annealing is a powerful optimization technique inspired by the physical process of annealing, in which a material is slowly cooled to reduce defects and reach a more stable state. Introduced in [37], this algorithm has gained significant popularity in various fields of science and engineering. Simulated annealing utilizes a probabilistic search strategy that allows it to explore a large solution space while effectively escaping local optima. By gradually accepting worse solutions during the initial stages, simulated annealing mimics the annealing process and eventually converges toward the global optimum. This versatile technique has found applications in diverse domains, including computer science, operations research, and artificial intelligence, making it a cornerstone in the field of optimization.

3.2. Finding the Maximum Weighted Cycle with Length K and Known Starting Point

We start by defining the Integer Linear Programming (ILP) formulation for the first problem. All linear programs consist of an objective function, which has to be either minimized or maximized, and constraints, which assure the feasibility of the solution. The solution to the problem of finding an MWC in the weighted, directed graph $G = (V, A, W)$ containing vertices $v \in V$ and arcs $a = (v_i, v_j) \in A$, each with weight $w_{i,j} \in W$, should be a cycle $S = (V', A', W') \subseteq G$. To formulate this problem as an ILP, some decision variables are introduced:

$$y_i = \begin{cases} 1 & \text{if } v_i \in V' \\ 0 & \text{otherwise} \end{cases} \qquad \forall\, i : v_i \in V,$$

$$x_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in A' \\ 0 & \text{otherwise} \end{cases} \qquad \forall\, i, j : (v_i, v_j) \in A,$$

$f_{i,j,\pm}$ the flow from i to j, where +/- indicates the direction.

The complete ILP for finding a cycle of length k with a known starting point, vertex 1, can now be written as:

$$\text{minimise } T_0 = \sum_{i=1}^{|V|} \sum_{j:(v_i,v_j)\in A} -w_{i,j}x_{i,j}, \tag{2}$$

subject to (for $i \in I = 1, ..., |V|$):

$$\sum_{j:(i,j)\in A} x_{i,j} = y_i, \ \forall i, \tag{3}$$

$$\sum_{j:(j,i)\in A} x_{j,i} = y_i, \ \forall i, \tag{4}$$

$$\sum_{j:(i,j)\in A} f_{i,j,+} + \sum_{j:(j,i)\in A} f_{j,i,-} = (k-2)y_i, \ \forall i > 1, \tag{5}$$

$$f_{i,j,+} + f_{i,j,-} = (k-1)x_{i,j}, \ \forall (v_i, v_j) \in A, i, j \neq 1, \tag{6}$$

$$\sum_{i=1}^{|V|} y_i = k \tag{7}$$

$$y_1 = 1, \tag{8}$$

$$y_i \in \{0, 1\}, \forall v_i \in V_{\setminus\{v_1\}}, \tag{9}$$

$$x_{i,j} \in \{0, 1\}, \forall (v_i, v_j) \in A, \tag{10}$$

$$f_{i,j} \in \mathbb{N}, \forall (v_i, v_j) \in V_{\setminus\{v_1\}}, (v_i, v_j) \in A \tag{11}$$

Equations (3) and (4) assure that the solution returns only cycles. Here the inbound degree and the outbound degree of vertex v both have to be equal to 1 $\forall\, v \in V'$. If vertex v is in the cycle, there is exactly one arc entering and exactly one arc leaving v_i. Now, the solution cannot be a path anymore. Nevertheless, those constraints still allow the solution to have multiple (disjoint) cycles. There must be a constraint that assures that the solution is only one single cycle. This can be accomplished with a constraint ensuring that any solution without a vertex $v \in V'$ does not contain a cycle. According to Equation (8), $v_1 \in V'$, so let v_1 be this specific vertex. Thus, if removing the starting vertex and all connected arcs from the solution leads to an acyclic path, this implies that the solution is only one single cycle instead of a set of disjoint cycles. Checking whether a graph is acyclic can be done with the maximum average degree (mad):

$$mad(G) = \max_{H \subseteq G} \left\{ \frac{2|E(H)|}{|V(H)|} \right\}.$$

If there exists a subgraph $H \subseteq G$ that contains a cycle, then the average degree of H $ad(H) \geq 2$. The average degree of a path with $k-1$ vertices, so excluding vertex v_1, however, is $\frac{2(k-2)}{k-1} = 2 - \frac{2}{k-1} < 2$ (for $k > 1$) [12].

To ensure that the maximum average degree is less than 2 in the solution without v_1, flow variables are used. Here, it does not matter that the given graph is directed, i.e., there is some flow in both directions over arc a if a is part of the solution. To make the solution acyclic, the flow on each arc $a \in A'$ has to be equal to $k-1$, while the incoming flow for vertex $v \in V'$ has to be equal to $k-2$ [12]. This leads to Equations (5) and (6). Here, variable $f_{i,j,\pm}$ represents the flow sent from vertex i to vertex j, where the $+$ or the $-$ are indicators of the direction. In order to reduce the search space, a constraint that assures that the number of nodes used in the cycle is precisely equal to k is added by Equation (7).

To reach the classical QUBO form, the first step is to reformulate the ILP to a single objective function formulation, such as:

$$\text{minimise: obj.function} + \lambda' \cdot \text{constraints.} \tag{12}$$

Note that Equation (12) is not a QUBO yet, but rather a combination of Hamiltonian matrices. As this formulation is, however, sufficient for the implementation, we will not derive the finalized QUBO formulation. From here on, we, therefore, refer to the combination of the Hamiltonian matrices as the QUBO formulation.

For that, λ is a vector of penalty weights. The higher the λ-value of a constraint, the more weight is put on this particular constraint, and the objective is down-weighted correspondingly. This topic will be discussed in more detail in Section 4.

To formulate this problem as a QUBO, the constraints have to be reformulated as penalty functions, which are minimized, equal to zero, when the condition is met. In order to reach this, the difference between the LHS and RHS is squared. Thus, Equation (3) becomes

$$\left(y_i - \sum_{j:(i,j)\in A} x_{i,j} \right)^2.$$

As this holds $\forall\ i = 1, ..., |V|$, the following needs to be included in the QUBO formulation:

$$\sum_{i=1}^{|V|} \left(y_i - \sum_{j:(i,j)\in A} x_{i,j} \right)^2.$$

It is also possible to formulate multiple constraints as one single QUBO formulation, such as for Equations (3) and (4):

$$T_1 = \sum_{i=1}^{|V|} \left(\left(y_i - \sum_{j:(i,j)\in A} x_{i,j} \right)^2 + \left(y_i - \sum_{j:(j,i)\in A} x_{j,i} \right)^2 \right).$$

The same holds for Equations (5) and (6):

$$T_2 = \sum_{i=2}^{|V|} \left(\sum_{j\neq 1:(i,j)\in A} \left((k-1)x_{i,j} - f_{i,j,+} - f_{i,j,-} \right)^2 + \left((k-2)y_i - \sum_{j\neq 1:(i,j)\in A} f_{i,j,+} - \sum_{j\neq 1:(j,i)\in A} f_{j,i,-} \right)^2 \right).$$

Note that combining constraints decreases the number of hyper-parameters; however, it comes at the cost of fine-tuning the penalty values. Last, but not least, Equations (7) and (8) also need to be transformed to penalty functions:

$$y_1 = 1 \Rightarrow (1 - y_1)^2,$$
$$\sum_{i=1}^{|V|} y_i = k \Rightarrow \left(k - \sum_{i=1}^{|V|} y_i \right)^2 = T_3.$$

The complete QUBO input is then:

$$T_0 + \lambda_1 (1 - y_1)^2 + \lambda_2 T_1 + \lambda_3 T_2 + \lambda_4 T_3. \tag{13}$$

Note that in the implementation, y_1 can be fixed, removing the penalty λ_1.

3.3. Finding the Maximum Weighted Cycle of Length $|V|$

One special case of finding the MWC is finding the MWC, which includes all vertices of the graph. As every vertex is part of the cycle, it is possible to choose an arbitrary vertex as the starting vertex. However, v_1 will still be referred to as the starting vertex. The approach to finding such a cycle is then to look for a max-weighted path instead of an MWC. In order to still get the MWC as a result, node v_1 is copied with all of its connecting arcs, now called $v_{|V|+1}$. The goal is then to find the max-weighted v_1–$v_{|V|+1}$ path.

To find this path ($\mathcal{P}$), new decision variables need to be introduced [38] ($\forall (v_i, v_j) \in A$):

$$x_{i,j,0} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{P} \\ 0 & \text{otherwise,} \end{cases}$$
$$x_{i,j,1} = \begin{cases} 1 & \text{if } (v_i, v_j) \notin \mathcal{P} \text{ and } p(i) < p(j) \\ 0 & \text{otherwise,} \end{cases}$$
$$x_{i,j,2} = \begin{cases} 1 & \text{if } (v_i, v_j) \notin \mathcal{P} \text{ and } p(i) > p(j) \\ 0 & \text{otherwise.} \end{cases}$$

Here, $p(i)$ indicates the position of node i in path $\mathcal{P}$. If the decision variable $x_{i,j,0} = 1$, it indicates that node i is reached earlier than node j in path $\mathcal{P}$ and the direct connection between those two nodes is part of path $\mathcal{P}$. Next, $x_{i,j,1} = 1$ if the direct connection between nodes i and j is not in path $\mathcal{P}$, however node i is reached in path $\mathcal{P}$ before node j. Lastly, $x_{i,j,2} = 1$ if the direct connection between nodes i and j is not in path $\mathcal{P}$, however node j is reached in path $\mathcal{P}$ before node i.

A simple 'cycle' is presented in Figure 2. For that cycle, $x_{1,2,0} = 1$, as the arc $a = (1, 2)$ is part of the cycle, and node 1 appears before node 2. The decision variable $x_{2,4,1} = 1$ because there is no connection between nodes 2 and 4, but node 2 is reached before node 4. Further, $x_{3,2,2} = 1$, because $a = (3, 2)$ is not in the path and node 2 is reached before node 3.

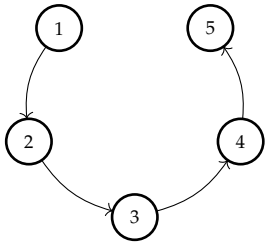


Figure 2. Simple cycle using 4 vertices, with Node 5 being a duplicate of Node 1.

The objective function is

$$\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} -w_{i,j}x_{i,j,0}. \tag{14}$$

Here, only the $(r = 0)$–cases are considered because only those arcs are part of the cycle. As $a = (v_i, v_j)$ is either in the path or not, and i is either before or after j,

$$\sum_{r=0}^{2} x_{i,j,r} = 1 \; \forall \; i, j = 1, ..., |V| + 1, i \neq j. \tag{15}$$

Further, $x_{i,j,r}$ is always the complement of $x_{j,i,r}$ $\forall$ r. Thus,

$$x_{i,j,2} = 1 - x_{j,i,2} \; \forall \; i, j = 1, ..., |V| + 1, i \neq j. \tag{16}$$

Equation (16) should, in principle, hold for all $r = \{0, 1, 2\}$, but Equation (15) makes the other two cases redundant. Next to these constraints, it must be ensured, just as before, that each node is entered (Equation (17)) and left (Equation (18)) exactly once.

$$\sum_{i=1}^{|V|} x_{i,j,0} = 1 \; \forall \; j : a = (v_i, v_j) \in A, j \neq 1, \tag{17}$$

$$\sum_{j=1}^{|V|+1} x_{i,j,0} = 1 \; \forall \; i : a = (v_i, v_j) \in A, i \neq |V| + 1. \tag{18}$$

However, the graph in question is not necessarily a complete one. Therefore, $x_{i,j,0}$ has to be zero if $a = (v_i, v_j) \notin A$:

$$x_{i,j,0} = 0 \; \forall \; a = (v_i, v_j) \notin A. \tag{19}$$

Lastly, cycles need to be prevented. To avoid these, the logic that if node i is reached before node j and node j is reached before node k, then node i has to be reached before node k as well, can be applied. Let $P(a_{i,j}, a_{j,k}, a_{i,k})$ be the penalty function for the QUBO. The only two contradicting combinations would be $P(1, 1, 0)$ and $P(0, 0, 1)$. If those two combinations were not contradicting, this would imply that sub-cycles are possible. Thus, the penalty function has to have the lowest value for all other combinations while it takes the highest value for those two particular combinations. Here, zero and one are picked as the lowest and highest values, respectively. To reach exactly this, the penalty function has the following form:

$$\sum_{i=2}^{|V|}\sum_{j=2}^{|V|}\sum_{k=2}^{|V|}(x_{j,i,2}x_{k,j,2} - x_{j,i,2}x_{k,i,2} - x_{k,j,2}x_{k,i,2} + x_{k,i,2}). \tag{20}$$

The complete QUBO formulation is shown in Appendix A as Equation (A1). Additionally, here, in the implementation, λ_3 can be removed by fixing those variables to zero.

3.4. Finding the Maximum Weighted Cycle with a Known Starting Point (I)

Small changes in the approach of Section 3.3 make it possible to find the MWC with a given starting vertex while the number of vertices used is unknown. In order to reach this, some additional decision variables need to be introduced. We now again assume the cycle $S = (V', A', W') \subseteq G$ and introduce ($\forall\, i, j : a = (v_i, v_j) \in A$)

$$x_{i,j,3} = \begin{cases} 1 & \text{if node } v_i \notin V' \text{ or } v_j \notin V' \\ 0 & \text{otherwise,} \end{cases}$$

and $\forall\, i : v_i \in V$

$$y_i = \begin{cases} 1 & \text{if node } v_i \in V' \\ 0 & \text{otherwise.} \end{cases}$$

Firstly, as r goes in this ILP from zero to three now, Equation (15) becomes

$$\sum_{r=0}^{3} x_{i,j,r} = 1 \;\forall\, i, j = 1, ..., |V| + 1, i \neq j. \tag{21}$$

The decision variable $x_{i,j,3}$ has to be one if y_i, y_j, or both, y_i and y_j, are equal to zero. If both, y_i and y_j, are, however, equal to one, $x_{i,j,3}$ must be zero. Equations (22)–(24) assure that.

$$1 - x_{i,j,3} \leq y_i \;\forall\, i, j = 1, ..., |V| + 1, \tag{22}$$

$$1 - x_{i,j,3} \leq y_j \;\forall\, i, j = 1, ..., |V| + 1, \tag{23}$$

$$x_{i,j,3} \leq 2 - y_i - y_j \;\forall\, i, j = 1, ..., |V| + 1. \tag{24}$$

Moreover, the complementary constraint (Equation (16)) is only the same as in the previous linear program if both nodes, v_i and v_j, are used. Otherwise, both $x_{i,j,2}$ and $x_{j,i,2}$ have to be equal to zero for all $i, j = 1, ..., n + 1$. To ensure that, the following penalty function is introduced:

$$x_{i,j,2} + x_{j,i,2} \geq y_j + y_i - 1 \;\forall\, i, j = 1, ..., |V| + 1. \tag{25}$$

Equation (25) assures that $x_{i,j,2}$ is the complement of $x_{j,i,2}$ if both vertices, v_i and v_j, are part of the cycle. In combination with Equation (21), all conditions are fulfilled if those constraints are met.

It might be the case that in the optimal solution, not all nodes are used. Nevertheless, the starting vertex must always be part of the cycle. As this approach, however, does not directly look at cycles but rather at a path from v_1 to a copy of it, $v_{|V|+1}$, this implies that $v_{|V|+1}$ must be part of the cycle as well. Equation (26) assures this:

$$y_1 = y_{|V|+1} = 1. \tag{26}$$

Further, every node but v_1 has to be entered, and every node but $v_{|V|+1}$ has to be left exactly once, given that they are part of the cycle. Thus, Equations (17) and (18) turn into:

$$\sum_{i=1}^{|V|} x_{i,j,0} = y_i \;\forall\, j : a = (v_i, v_j) \in A, j \neq 1, \tag{27}$$

$$\sum_{j=1}^{|V|+1} x_{i,j,0} = y_i \;\forall\, i : a = (v_i, v_j) \in A, i \neq |V| + 1. \tag{28}$$

Lastly, Equation (20) must be redundant if node v_j is not part of the cycle. For that $-x_{k,i,2}(1 - y_j)$ is added to this penalty function. If node v_j is used, this penalty function is

exactly the same as Equation (20), but if it is not, this penalty function is always zero, thus becoming redundant. Equations (14) and (19) remain the same as above.

In order to obtain the complete QUBO formulation, the constraints have to be transformed into penalty functions, which are minimized when the constraint is met. Equations (22)–(25), however, contain an inequality sign. This implies that the approach used before cannot be applied here anymore. Nevertheless, by introducing slack variables, these inequalities can be converted into equalities ($\forall\, i, j = 1, ..., |V| + 1, i \neq j$):

$$\begin{aligned} 1 - x_{i,j,3} \leq y_i &\rightarrow 1 - x_{i,j,3} + s_{i,j,3,1} = y_i, \\ 1 - x_{i,j,3} \leq y_j &\rightarrow 1 - x_{i,j,3} + s_{i,j,3,2} = y_j, \\ x_{i,j,3} \leq 2 - y_i - y_j &\rightarrow x_{i,j,3} + s_{i,j,3,3} = 2 - y_i - y_j, \\ x_{i,j,2} + x_{j,i,2} &\geq y_j + y_i - 1 \\ \rightarrow x_{i,j,2} + x_{j,i,2} &= y_j + y_i - 1 + 2^0 s_{i,j,2,1} + 2^1 s_{i,j,2,2}. \end{aligned}$$

To use as few additional decision variables (additional slack variables) as possible, binary formulation (called log integer encoding) for rewriting Equation (25) is used. It does not make a difference for the other three equations. The complete QUBO formulation is shown in Appendix A as Equation (A2).

3.5. Finding the Maximum Weighted Cycle with a Known Starting Point (II)

Another approach for finding the MWC with a known starting point is by using the degree-based formulation as in Section 3.2. The flow constraints have to change, as they both depend on k. To fix this problem, a subtour-elimination constraint proposed by Miller, Tucker, Zemlin (MTZ) [39] is used:

$$t_i + 1 + |V|(x_{i,j} - 1) \leq t_j,\ \forall (v_i, v_j) \in A, i, j > 1. \tag{29}$$

This equation also represents flow (t_i), but the interpretation is different from Equations (5) and (6). If the arc $a = (i, j)$ is in the cycle (i.e., $x_{i,j} = 1$), this implies that $t_j \geq t_i + 1$. The complete ILP formulation for this problem is:

$$\text{minimise} \sum_{i=1}^{|V|} \sum_{j:(v_i,v_j)} \in A - w_{i,j} x_{i,j},$$

subject to:

$$\begin{aligned} \sum_{j \neq 1:(i,j) \in A} x_{i,j} &= y_i,\ \forall\, i = 1, ..., |V|, \\ \sum_{j \neq 1:(j,i) \in A} x_{j,i} &= y_i,\ \forall\, i = 1, ..., |V|, \\ t_i + 1 + |V|(x_{i,j} - 1) &\leq t_j, \forall (v_i, v_j) \in A, i, j > 1, \\ y_1 &= 1, \\ y_i &\in \{0, 1\},\ \forall\, v_i \in V \setminus v_1, \\ x_{i,j} &\in \{0, 1\},\ \forall\, a = (v_i, v_j) \in A, \\ t_i &\in \mathbb{Z}^+, \quad \forall\, i = 2, ..., |V|. \end{aligned}$$

The only new element in the QUBO formulation is the subtour-elimination constraint. This additional constraint, however, is an inequality constraint, such that the usual way of reformulation does not work anymore. To make it work, slack variables are used once more. In order to use as few slack variables as possible, they are again represented with a binary expansion. Additionally, the non-binary t_i variables have to be translated to

binary variables in the same way, introducing the binary auxiliary variables $z_{i,k}$. The QUBO formulation of the MTZ-subtour-elimination constraint is, therefore:

$$T_4 = \sum_{i=2}^{|V|} \sum_{j=2:(i,j)\in A}^{|V|} \left(\sum_{k=0}^{K_1-1} 2^k z_{j,k} - \left(\sum_{k=0}^{K_1-1} 2^k z_{i,k} + 1 + |V|(x_{i,j}-1) + \sum_{k=0}^{K_2-1} 2^k s_{j,k} \right) \right)^2,$$

where K_1 is the number of binary variables required to encode the flow variables, and K_2 is the number of slack variables needed. This number must be large enough but parsimonious at the same time to avoid wasting resources:

$$\begin{aligned} & t_i + 1 + |V|(x_{i,j}-1) + \text{slack} = t_j \\ \Leftrightarrow\ & \text{slack} \leq \max\left(t_j - (t_i + 1 + |V|(x_{i,j}-1))\right) \\ & \quad = \max\left(t_j\right) - \min\left(t_i + 1 + |V|\left(x_{i,j}-1\right)\right) \\ & \quad = \max\left(t_j\right) - \min\left(t_i\right) - 1 - \min\left(|V|\left(x_{i,j}-1\right)\right) \\ & \quad = (|V|-1) - 0 - 1 - (-|V|) \\ & \quad = 2|V| - 2. \end{aligned}$$

As a number x needs $\lfloor \log_2(x) \rfloor + 1$ digits, this implies that $K_2 = \lfloor \log_2(2|V|-2) \rfloor + 1$. For K_1 it holds that $K_1 = \lfloor \log_2(|V|) \rfloor + 1$.

Finally, the complete QUBO formulation for this approach is:

$$\begin{aligned} & \sum_{i=1}^{|V|} \sum_{j:(v_i,v_j)} \in A - w_{i,j}x_{i,j} + \lambda_1 \left((1-y_1)^2\right) \\ & + \lambda_2 \sum_{i=1}^{|V|} \left(\left(y_i - \sum_{j:(i,j)\in A} x_{i,j} \right)^2 + \left(y_i - \sum_{j:(i,j)\in A} x_{i,j} \right)^2 \right) \\ & + \lambda_3 T_4. \end{aligned}$$

Again λ_1 can be removed by fixing $y_1 = 1$ in the implementation.

4. Implementation

Now we have theoretical QUBO formulations for the MWC problem, we move to the implementation, using conventional and (hybrid) quantum solvers. For the implementation, it is crucial to keep track of the number of decision variables used for the problem. The more decision variables needed to represent the QUBO formulation of a particular problem (and the less these decision variables are cross-related), the more qubits are needed. However, the number of qubits available is limited. Thus, the fewer decision variables needed to encode the problem, the more likely it is that the approach on a (hybrid) quantum solver is more efficient. Note that it can also depend on the cross-relation between the decision variables.

The number of decision variables that the QUBO for finding the MWC of length k with a given starting vertex equals:

$$|V| + |A| + 2(\lfloor \log_2(k-2) \rfloor + 1)|A'|$$

The first $|V|$ decision variables are indicators for the usage of each vertex, while the following $|A|$ decision variables do that for all arcs. The last $2(\lfloor \log_2(k-2) \rfloor + 1)|A'|$ variables are for the flow constraint. Here, A' is the set of all arcs in the graph, excluding those which have v_1 as a start or endpoint. Each arc $a \in A'$ needs two flow values (back and forth), and the maximum value of a flow is $k-2$. Using the binary numbering, this results in $2(\lfloor \log_2(k-2) \rfloor + 1)|A'|$ flow-decision variables. The approach for finding the MWC of length $|V|$ using the $x_{i,j,r}$ decision variables requires in total $|V'| \times |V| \times 3$ decision variables. $|V'|$ is the number of vertices in the graph with the copied start-

ing vertex v_1 ($|V'| = |V| + 1$). Those $|V'| \times |V| \times 3$ variables represent $x_{i,j,r}$ for all $i, j = 1, ..., |V| + 1, i \neq j, \mathrm{r} = 0, 1, 2$. The approach [I] for finding the MWC with a given starting vertex which builds up on the idea of the previously described approach, needs in total $|V'| + |V'| \times |V| \times (4 + 6)$ decision variables. Here, the first $|V'|$ variables indicate whether a node $v \in V'$ is part of the cycle (y-variables). The following $|V'| \times |V| \times 4$ are, just as before, the $x_{i,j,r}$ variables. It is $\times 4$ instead of $\times 3$ because, for this approach, the introduction of an additional $x_{i,j,3}$ variable was necessary. Because there are six different types of slack variables, in total, $|V'| \times |V| \times 6$ decision variables are required to represent them all. For the second approach to find the cycle with a given start vertex, the number of decision variables is equal to

$$|A| + |V| + (\lfloor \log_2(|V| - 1) \rfloor + 1)(|V| - 1) + \\ + (\lfloor \log_2(2|V| - 2) \rfloor + 1)|A'|, \tag{30}$$

where $A' = A \setminus \{a = (v_1, v_i) \cup a = (v_i, v_1) \; \forall \, i : a \in A\}$, i.e., number of arcs after removing v_1 and all connecting arcs from the graph. The first $|A|$ decision variables are all $x_{i,j}$ variables. They are one if arc $a = (v_i, v_j)$ is in the cycle and zero otherwise. The next $|V|$ variables represent the vertex decision variables y_i. If y_i is one, this implies that vertex v_i is used. Further, there are $(\lfloor \log_2(|V| - 1) \rfloor + 1)(|V| - 1)$ decision variables for the MTZ cycle-canceling constraints and additional $(\lfloor \log_2(2|V| - 2) \rfloor + 1)|A'|$ for the respective slack variables (see Section 3.5). Each vertex but v_1 gets one t-value (MTZ-constraint), and each of the $|A'|$ MTZ-constraint has its own slack variables. The principle of binary numbers was used for both constraints. $|V| - 1$ is an upper-bound for $\max_j\{t_j\}$. Thus, there are $(\lfloor \log_2(|V| - 1) \rfloor + 1)(|V| - 1)$ decision variables which represent the flow in the MTZ cycle-canceling constraints. Based on the number of decision variables, the second approach seems to be superior.

Next to the number of decision variables, the choice of the penalty values is also crucial. Purely mathematically, sufficient high penalty values guarantee that the optimal solution for the QUBO-formulation is feasible and optimal for the original problem. Ideas for finding the lower bounds for those penalties value are given by Lucas [32]. For the implementation here, due to the relatively low precision of the quantum annealer, the larger the penalty value of a constraint, the more weight is put on it, and the less likely it is that this constraint will be violated. If the chosen penalty values are too large, all constraints will be met, but the weight of the objective function is relatively small. D-Wave's autoscale function rescales the QUBO coefficients into a fixed range, making the objective essentially zero after rounding from some point onwards. Therefore, the solution set will only be in the feasible part, but the optimal solution might not be part of it. If, however, the chosen penalty values are too small, it might be better to violate certain constraints in order to make the objective smaller. Nevertheless, infeasible solutions are clearly not desired either. The goal is to set the λ-values such that the solution to the penalized QUBO formulation returned by the solver is feasible and near the optimal value. In practice, this optimal value is unknown, so we are looking for good, feasible solutions. Where there are ideas on how to do this [40,41], a 'one-fits-all' penalty value was not (yet) found. Therefore, a penalty grid search is conducted to meet the goal described before. The penalty grid search was performed by looping over exponential growing values (1, 10, 100, etc.) for all penalty variables and a linear search in the range of the best found.

Once the number of decision variables and the fitting penalty values are determined, the implementation of the D-Wave Advantage System with over 5000 qubits is then only defining all Hamiltonian matrices and the respective penalty functions. The environment that is used for the implementation is PyQUBO. This package transforms the Hamiltonian-matrix formulation into a proper QUBO formulation. In order to use the tools D-Wave Systems provides to find a solution, the D-Wave Ocean SDK software is utilized.

D-Wave offers multiple types of solvers [42]. This paper, however, narrows its focus to three of them:

D-Wave QPU: The quantum processing unit (QPU) from D-Wave is a lattice of interconnected qubits. This is D-Wave's real quantum computer and would therefore be the best option to use. Nevertheless, the number of qubits is limited, which limits the number of decision variables needed to encode the problem. Thus, using the QPU directly is not an option for most instances.

D-Wave Hybrid BQM Solver: For larger instances, the hybrid BQM solver from D-Wave is the better option. Here, it is allowed to use 10,000–1,000,000 decision variables, depending on the interaction between the variables. This option is, however, as the name already suggests, a hybrid rather than a full quantum approach. In order to find a solution to the input, the hybrid solver dispatches one or more hybrid solvers and therefore implements multiple individual heuristics. BQM is an acronym for Binary Quadratic Models and specifies the form of the input. The specific methods used are not specified.

D-Wave Hybrid CQM solver: This hybrid solver can, just like the BQM hybrid solver, solve larger instances. CQM is an acronym for Constrained Quadratic Model. The input for this solver is therefore not the QUBO, as it is for the other two solvers, but rather the linear program with the objective function and the respective constraints, where it creates the QUBO itself.

A great advantage of using the Hybrid CQM solver is that there is no need to define the penalty values anymore. Here also, the specific methods used are not specified.

5. Classic Versus Quantum

For the final analysis, we use 7 graphs of approximately three sizes. Some characteristics of these graphs are shown in Table 1: number of vertices, number of arcs, and size of the problem in a number of variables, using Equation (30). The first five are randomly created. For the last and biggest instance, we use the debt data (Retrieved from https://figshare.com/articles/dataset/data_txt/14547432/1. (accessed on 10 June 2023)) collected in Huangdao Zone, Shandong Province, China [4]. This graph, as depicted in Figure 3, was decreased in size by deleting all vertices, with an inbound or outbound degree equal to zero and all the corresponding in-/out- arcs. To investigate the maximum capabilities of the D-Wave QPU solver, we used a specific graph structure for the last instance. This graph structure is such that there are n nodes in a full circuit, then adding one extra arc to connect node 2 and n, having weight $n-1$, ensuring that this small circuit has a higher weight $(n+1)$ than the larger circuit (n). The largest graph of this form that can be solved on the current QPU has $n = 58$. For all graphs, a QUBO is generated, which includes finding the penalty value, as described earlier in Section 4. In Table 2, an example, for instance, 3, is given of how the solution and the number of feasible solutions found depends on the penalty value. We here only show one dimension as an illustration of the idea, varying λ_2 only. Here λ_1 is removed by fixing the constraint and $\lambda_3 = 10$. The real grid search is executed in both dimensions.

Table 1. Overview of instances used.

Instance	Vertices	Edges	Variables
Instance 1	4	5	29
Instance 2	4	6	38
Instance 3	21	89	878
Instance 4	26	104	1043
Instance 5	31	116	1184
Instance 6	202	375	5943
Instance 7	58	59	857

The focus for the comparison of classical computing versus quantum computing is on the problem formulation of finding the MWC, given a starting vertex, as this is the problem with the least number of assumptions. As described in Section 4, the approach of Section 3.5 uses fewer decision variables than that of Section 3.4. Further, the results obtained from the

penalty-value grid search were also better. Therefore, the approach from Section 3.5 is the one of choice for the comparison. For the comparison, we used the classical *DFS*-approach described in Section 2, Simulated Annealing (SA) based on the defined QUBO and the three (hybrid) quantum approaches as presented in the previous section. Simulated Annealing is a probabilistic technique that approximates the optimal solution. Thus, it is a heuristic in the universe of classical computing. For all methods, we used the default settings, meaning, e.g., default chain strength and no post-processing in the pure QPU approach. Only the maximum calculation time for the Hybrid BQM method was set to 3 and 60 s, respectively, for the small and medium-sized problems.

Table 2. Effect penalty value on best solution found and on the percentage feasible solutions in 250 reads.

Penalty Value	Best Solution	% Feasible
1000	-	-
2000	47	4.4%
3000	46	3.2%
3500	42	6.4%
4000	76	4.0%
4500	51	2.8%
5000	52	2.0%
6000	31	1.6%
7000	32	2.0%
8000	31	0.8%
9000	44	2.0%
10,000	45	1.2%
20,000	10	0.8%
30,000	-	-

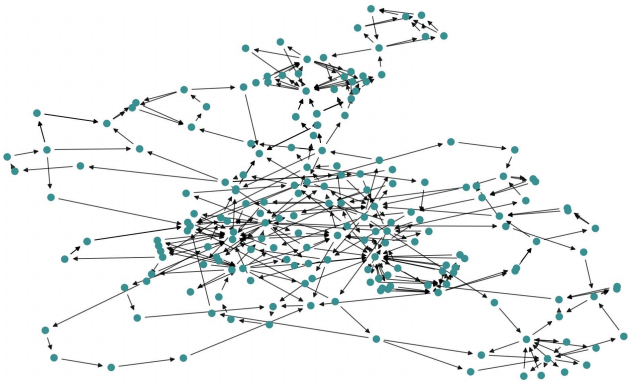


Figure 3. Huangdao Zone graph without vertices of in-/out-degree 0.

Looking at both classical approaches, it is clear that with an increasing number of arcs (and nodes), the computation time also increases.

For small instances, all approaches, quantum and classical, are very fast and accurate, as shown in Tables 3 and 4. With an increasing number of arcs, however, the computation time for the DFS method grows exponentially. The calculation, for instance, 6 has even been aborted due to a memory problem, using a computer cluster (8 GB RAM/8 Intel Core Processors (Broadwell, no TSX, IBRS) CPU, 2100 MHz). However, by the time of receiving the error message, only a very small part of the problem was solved. This implies that, even if the memory issue was fixed, the entire computation would exceed a month. This underlines the computational expensiveness of the problem in classical computing. The computation time for simulated annealing also grows exponentially with an increasing number of arcs but by far not as fast as for the DFS method. Nevertheless, there is a trade-off: the solutions found by the simulated annealing for larger instances are not the optimal solutions found by the DFS method. However, also for bigger instances, SA will not be suitable.

Table 3. Comparing calculation times solvers and methods. The QPU used is the D-Wave Advantage system. The DFS approach is run on a computer cluster (8GB RAM/8 Intel Core Processors (Broadwell, no TSX, IBRS) CPU, 2100 MHz). The SA approach was run on a laptop using a single Intel i5 core.

Instances	Classical Approaches	Quantum Approaches						
	DFS	SA	QPU (100 Reads)		Hybrid BQM		Hybrid CQM	
		(750 Reads)	Access Time	E2E	pure QPU	E2E	pure QPU	E2E
Instance 1	0.201 ms	0.303 s	30.390 ms	5.501 s	0.130 s	2.994 s	0.015 s	5.000 s
Instance 2	0.099 ms	0.474 s	27.918 ms	5.081 s	0.125 s	2.995 s	0.015 s	4.944 s
Instance 3	8.270 s	15.842 s	-	-	1.254 s	59.997 s	0.015 s	5.000 s
Instance 4	104.520 s	18.471 s	-	-	1.387 s	59.998 s	0.015 s	5.061 s
Instance 5	1791.870 s	20.309 s	-	-	1.263 s	59.998 s	0.015 s	4.980 s
Instance 6	-	150.635 s	-	-	-	-	0.015 s	5.256 s
Instance 7	0.600 ms	36.213 s	42.038 ms	459.220 s	1.387 s	59.997 s	0.015 s	5.079 s

Table 4. Comparing solution solvers and methods.

Instances	DFS	SA	QPU	BQM	CQM
Instance 1	5	5	5	5	5
Instance 2	5	5	5	5	5
Instance 3	136	79	-	58	131
Instance 4	167	71	-	52	148
Instance 5	194	55	-	45	135
Instance 6	-	182	-	-	645
Instance 7	59	59	59	59	59

Looking at times for the direct QPU-solver, one recognizes that more than half of the data are missing. The number of decision variables needed to encode some of the problems simply exceeds the capacity of the QPU. Nevertheless, it is still good to observe that the approach is at least feasible for the smaller instances. The largest problem that was solved here needs much time to find the embedding for the problem. For the hybrid BQM solver, the problem here is not the number of decision variables but rather the solution itself. The BQM solver returns only one solution, whereas the direct QPU solver returns all the reads it performs. To get a feasible best (in terms of energy only) solution, the penalty values are set such that the solution delivered is relatively poor. Lastly, let us have a look at the data for the hybrid CQM solver. The pure QPU time is less than 1 s—for all instances. For this solver, it does not make a difference whether the number of arcs is 4 or 31. Nevertheless, just as for the simulated annealing, there is a trade-off in the quality of the result. Considering the motivation of eliminating debt from the system, this solution includes 11 entities and the same number of arcs. The settled amount is 6, implying that in total, $11 \times 6 = 66$ units of debt can be eliminated. Because of the memory issue for the DFS method, the quality of this result can, however, not be checked. Nevertheless, compared to the result found by simulated annealing, this solution is a major improvement.

6. Conclusions and Discussion

In this paper, four different QUBO formulations for finding the MWC in a directed graph were presented. It is expected that this formulation will be of importance when quantum computers are at a size that allows them to bring added value. It was first assumed that the starting point and the number of vertices used in the cycle were both given. Later on, the latter assumption of knowing the number of vertices used was loosened. However, the assumption of knowing the starting vertex is important for all four approaches listed in this paper. The most general problem of finding the MWC in a directed graph would therefore be left as an open problem.

These formulations made it possible to implement these approaches on the D-Wave Ocean SDK software in order to use D-Wave's quantum annealer. For the implementation,

the QUBO formulation itself was not sufficient, however, because the choice of the penalty values is essential. There is still no method that assures optimal results for every instance and problem. The problem of finding more fitting and general methods for setting those penalty coefficients is, therefore, still a problem to be addressed in the future. Due to the fact that there exists limited literature in this field, a penalty grid search was executed. During execution, it became clear that the number of feasible solutions found decreased as the number of nodes and arcs increased.

The final comparison of (hybrid) quantum versus classical computing for this particular problem and the chosen hardware configurations revealed both the up- and downsides of this form of quantum computing. The goal of this paper was to use (hybrid) quantum annealing in order to speed up the computation time needed to solve a problem that is $\mathcal{NP}$-hard, such as the MWC problem. Using the CQM hybrid solver, the goal of reducing the computation time was reached. The computation time needed for large instances is much lower using quantum computing than it is for classical computing. The results derived by the (hybrid) quantum annealer, however, underline that this approach is a heuristic approach that does not guarantee optimal solutions. Thus, there is a trade-off in time to quality. For the other two solvers, the direct QPU embedding and the BQM hybrid solver, no (good) solutions for larger instances were obtained. The source of the problem, however, differs among these two solvers. In order to solely use the direct QPU, it has to be possible to encode the problem with only a few decision variables. The worst case here is around 175 in a fully connected graph; in our case, we could go up to around 850 variables, as shown before. In the near future, however, a new machine from D-Wave, the Advantage 2, will be available, allowing more decision variables and giving a higher connectivity. For the BQM hybrid solver, on the other hand, to find a solution, the penalty values are set such that worse solutions are found. This highlights once again the importance of penalty values.

For the debt data collected in Huangdao Zone, Shandong Province, China, classical computing for finding the optimal solution (DFS method) did not function at all due to a memory problem. On the other hand, simulated annealing, which is another classical approach, did derive a result for this problem. Nevertheless, simulated annealing is also only a heuristic. The result generated by using the simulated annealing was much lower than the result found by the CQM hybrid solver from D-Wave. Further, the CQM solver also took only about 0.01% of the time. When looking at the amount of debt that can be eliminated with these solutions, the difference, however, is not as substantial.

Applying the power of quantum computing to related problems might be something to consider for future research. Related problems could entail finding the cycle with the maximal settlement amount or finding the set of disjoint cycles that maximizes the overall weight. Solving these would lead to solutions that allow the elimination of more debt. Further, one could also use different definitions for a cycle, e.g., allow the re-usage of vertices.

Author Contributions: Conceptualization, F.P. and H.K.; methodology, F.P. and H.K.; software, H.K.; validation, F.P. and H.K.; investigation, H.K.; writing—original draft preparation, H.K.; writing—review and editing, F.P.; supervision, F.P.; All authors have read and agreed to the published version of the manuscript.

Funding: The work of F.P. was funded by the Dutch National Growth Fund (NGF), as part of the Quantum Delta NL programme.

Data Availability Statement: Part of the data is retrieved from public sources. The other data instances are available upon request.

Acknowledgments: This work was supported by the Dutch National Growth Fund (NGF), as part of the Quantum Delta NL programme. The work is based on the bachelor thesis of the first author.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A

$$\begin{aligned}
&\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}\left(-w_{i,j}x_{i,j,0}\right)+\lambda_1\left(\sum_{j=2}^{|V|+1}\left(1-\sum_{i=1,(i,j)\in A}^{|V|}x_{i,j,0}\right)^2+\sum_{i=1}^{|V|}\left(1-\sum_{j=1,(i,j)\in A}^{|V|+1}x_{i,j,0}\right)^2\right)\\
&+\lambda_2\left(\sum_{i=1}^{|V|+1}\sum_{j=1,j\neq i}^{|V|+1}\left(\left(1-\sum_{r=0}^{2}x_{i,j,r}\right)^2+\left(1-x_{j,i,2}-x_{i,j,2}\right)^2\right)\right)+\lambda_3\sum_{a=(v_i,v_j)\notin A}\left(x_{i,j,0}\right)^2\\
&+\lambda_4\left(\sum_{i=2}^{|V|}\sum_{j=2}^{|V|}\sum_{k=2}^{|V|}\left(x_{j,i,2}x_{k,j,2}-x_{j,i,2}x_{k,i,2}-x_{k,j,2}x_{k,i,2}+x_{k,i,2}\right)\right).
\end{aligned}\tag{A1}$$

$$\begin{aligned}
&\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}\left(-w_{i,j}x_{i,j,0}\right)+\lambda_1\left(\sum_{j=2}^{|V|+1}\left(y_i-\sum_{i=1(i,j)\in A}^{|V|}x_{i,j,0}\right)^2+\sum_{j=2}^{|V|}\left(y_j-\sum_{j=1,(i,j)\in A}^{|V|+1}x_{i,j,1}\right)^2\right)\\
&+\lambda_2\sum_{i=1}^{|V|+1}\sum_{j=1,j\neq i}^{|V|+1}\left(1-\sum_{r=0}^{3}x_{i,j,r}\right)^2+\lambda_3\left(\left(1-y_1\right)^2+\left(1-y_{|V|+1}\right)^2\right)+\lambda_4\sum_{a\notin A}\left(x_{i,j,0}\right)^2\\
&+\lambda_5\sum_{i=1}^{|V|+1}\sum_{j=1,j\neq i}^{|V|+1}\left(\left(y_i-\left(1-x_{i,j,3}+s_{i,j,3,1}\right)\right)^2\left(y_j-\left(1-x_{i,j,3}+s_{i,j,3,2}\right)\right)^2+\left(2-y_i-y_j-\left(x_{i,j,3}+s_{i,j,3,3}\right)\right)^2\right)\\
&+\lambda_6\left(y_j+y_i-1+2^0s_{i,j,2,1}+2^1s_{i,j,2,2}-\left(x_{i,j,2}+x_{j,i,2}\right)\right)^2\\
&+\lambda_7\left(\sum_{i=2}^{|V|}\sum_{j=2}^{|V|}\sum_{k=2}^{|V|}\left(x_{j,i,2}x_{k,j,2}-x_{j,i,2}x_{k,i,2}-x_{k,j,2}x_{k,i,2}+x_{k,i,2}-x_{k,i,2}(1-y_j)\right)\right).
\end{aligned}\tag{A2}$$

References

1. Dalio, R. How the economic machine works. *Econ. Princ.* **2012**. Available online: https://www.economicprinciples.org/downloads/ray_dalio_how_the_economic_machine_works_leveragings_and_deleveragings.pdf (accessed on 10 June 2023).
2. Maki, S. *World's $ 281 Trillion Debt Pile Is Set to Rise Again in 2021*; Bloomberg: New York, NY, USA, 2021.
3. Ozili, P.K. The COVID-19 global debt crisis: How to avoid it. In *Smart Analytics, Artificial Intelligence and Sustainable Performance Management in a Global Digitalised Economy*; Emerald Publishing Limited: Bingley, UK, 2021.
4. Cui, H. DebtG: A Graph Model for Debt Relationship. *Information* **2021**, *12*, 347. [CrossRef]
5. Kosaraju, S.R.; Park, J.K.; Stein, C. Long tours and short superstrings. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 166–177.
6. Krentel, M.W. The complexity of optimization problems. *J. Comput. Syst. Sci.* **1988**, *36*, 490–509. [CrossRef]
7. Neukart, F.; Compostella, G.; Seidel, C.; Von Dollen, D.; Yarkoni, S.; Parney, B. Traffic flow optimization using a quantum annealer. *Front. ICT* **2017**, *4*, 29. [CrossRef]
8. Phillipson, F.; Bhatia, H.S. Portfolio optimisation using the D-Wave quantum annealer. In Proceedings of the International Conference on Computational Science, Krakow, Poland, 16–18 June 2021; pp. 45–59.
9. Phillipson, F.; Chiscop, I. Multimodal container planning: A QUBO formulation and implementation on a quantum annealer. In Proceedings of the International Conference on Computational Science, Krakow, Poland, 16–18 June 2021; pp. 30–44.
10. Orús, R.; Mugel, S.; Lizaso, E. Quantum computing for finance: Overview and prospects. *Rev. Phys.* **2019**, *4*, 100028. [CrossRef]
11. Yarkoni, S.; Raponi, E.; Bäck, T.; Schmitt, S. Quantum annealing for industry applications: Introduction and review. *Rep. Prog. Phys.* **2022**, *85*, 104001. [CrossRef] [PubMed]
12. McCollum, J.; Krauss, T. QUBO formulations of the longest path problem. *Theor. Comput. Sci.* **2021**, *863*, 86–101. [CrossRef]
13. Szegedy, M. On the quantum query complexity of detecting triangles in graphs. *arXiv* **2003**, arXiv:quant-ph/0310107.
14. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
15. Richards, D.; Liestman, A.L. Finding cycles of a given length. In *North-Holland Mathematics Studies*; Elsevier: Amsterdam, The Netherlands, 1985; Volume 115, pp. 249–255.
16. Cirasella, J. Classical and Quantum Algorithms for Finding Cycles. Master's Thesis, University of Amsterdam, Amsterdam, The Netherlands, 2006.
17. Alon, N.; Yuster, R.; Zwick, U. Finding and counting given length cycles. *Algorithmica* **1997**, *17*, 209–223. [CrossRef]
18. Yuster, R.; Zwick, U. Finding even cycles even faster. *SIAM J. Discret. Math.* **1997**, *10*, 209–222. [CrossRef]
19. Wang, C.; Zhou, R.G. A quantum search algorithm of two-dimensional convex hull. *Commun. Theor. Phys.* **2021**, *73*, 115102. [CrossRef]

20. Korf, R.E. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.* **1985**, *27*, 97–109. [CrossRef]
21. Haeupler, B.; Kavitha, T.; Mathew, R.; Sen, S.; Tarjan, R.E. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Trans. Algorithms (TALG)* **2012**, *8*, 1–33. [CrossRef]
22. Kahn, A.B. Topological sorting of large networks. *Commun. ACM* **1962**, *5*, 558–562. [CrossRef]
23. Bläser, M.; Manthey, B. Approximating maximum weight cycle covers in directed graphs with weights zero and one. *Algorithmica* **2005**, *42*, 121–139. [CrossRef]
24. Alon, N.; Yuster, R.; Zwick, U. Color-coding. *J. ACM (JACM)* **1995**, *42*, 844–856. [CrossRef]
25. Gabow, H.N.; Nie, S. Finding a long directed cycle. *ACM Trans. Algorithms (TALG)* **2008**, *4*, 1–21. [CrossRef]
26. Björklund, A.; Husfeldt, T.; Khanna, S. Approximating longest directed paths and cycles. In Proceedings of the International Colloquium on Automata, Languages, and Programming, Turku, Finland, 12–16 July 2004; pp. 222–233.
27. Krauss, T.; McCollum, J. Solving the network shortest path problem on a quantum annealer. *IEEE Trans. Quantum Eng.* **2020**, *1*, 1–12. [CrossRef]
28. Krauss, T.; McCollum, J.; Pendery, C.; Litwin, S.; Michaels, A.J. Solving the max-flow problem on a quantum annealing computer. *IEEE Trans. Quantum Eng.* **2020**, *1*, 1–10. [CrossRef]
29. Mahasinghe, A.; Hua, R.; Dinneen, M.J.; Goyal, R. Solving the Hamiltonian cycle problem using a quantum computer. In Proceedings of the Australasian Computer Science Week Multiconference, Sydney, Australia, 29–31 January 2019; pp. 1–9.
30. Nüßlein, J.; Gabor, T.; Linnhoff-Popien, C.; Feld, S. Algorithmic QUBO Formulations for k-SAT and Hamiltonian Cycles. *arXiv* **2022**, arXiv:2204.13539.
31. Rosenberg, G. Finding optimal arbitrage opportunities using a quantum annealer. In *1QB Information Technologies White Paper*; 1QBit: Vancouver, BC, Canada, 2016; pp. 1–7.
32. Lucas, A. Ising formulations of many NP problems. *Front. Phys.* **2014**, *2*, 5. [CrossRef]
33. Glover, F.; Kochenberger, G.; Du, Y. A tutorial on formulating and using QUBO models. *arXiv* **2018**, arXiv:1811.11538.
34. Kadowaki, T.; Nishimori, H. Quantum annealing in the transverse Ising model. *Phys. Rev. E* **1998**, *58*, 5355. [CrossRef]
35. Das, A.; Chakrabarti, B.K. Quantum annealing and analog quantum computation. *Rev. Mod. Phys.* **2008**, *80*, 1061–1081. [CrossRef]
36. McGeoch, C.C. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synth. Lect. Quantum Comput.* **2014**, *5*, 1–93.
37. Kirkpatrick, S.; Gelatt, C.D., Jr.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
38. González-Bermejo, S.; Alonso-Linaje, G.; Atchade-Adelomou, P. GPS: Improvement in the formulation of the TSP for its generalizations type QUBO. *arXiv* **2021**, arXiv:2110.12158.
39. Miller, C.E.; Tucker, A.W.; Zemlin, R.A. Integer programming formulation of traveling salesman problems. *J. ACM (JACM)* **1960**, *7*, 326–329. [CrossRef]
40. Ayodele, M. Penalty Weights in QUBO Formulations: Permutation Problems. In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar), Madrid, Spain, 20–22 April 2022; pp. 159–174.
41. Roch, C.; Impertro, A.; Linnhoff-Popien, C. Cross Entropy Optimization of Constrained Problem Hamiltonians for Quantum Annealing. In Proceedings of the International Conference on Computational Science, Krakow, Poland, 16–18 June 2021; pp. 60–73.
42. D-Wave. *Hybrid Solvers for Quadratic Optimization*; Technical Report; D-Wave: Burnaby, BC, Canada, 2022.