Algorithmic QUBO formulations for k-SAT and hamiltonian cycles

Nüßlein, Jonas; Gabor, Thomas; Linnhoff-Popien, Claudia; Feld, Sebastian

DOI
10.1145/3520304.3533952

Publication date
2022

Document Version
Final published version

Published in
GECCO 2022 Companion - Proceedings of the 2022 Genetic and Evolutionary Computation Conference

Algorithmic QUBO Formulations for k-SAT and Hamiltonian Cycles

Jonas Nüßlein
jonas.nuesslein@ifi.lmu.de
LMU Munich
Germany

Thomas Gabor
thomas.gabor@ifi.lmu.de
LMU Munich
Germany

Claudia Linnhoff-Popien
linnhoff@ifi.lmu.de
LMU Munich
Germany

Sebastian Feld
s.feld@tudelft.nl
TU Delft
Netherlands

ABSTRACT

Quadratic Unconstrained Binary Optimization (QUBO) can be seen as a generic language for optimization problems. QUBOs attract particular attention since they can be solved with quantum hardware, like quantum annealers or quantum gate computers running QAOA. In this paper, we present two novel QUBO formulations for k-SAT and Hamiltonian Cycles that scale significantly better than existing approaches. For k-SAT we reduce the growth of the QUBO matrix from $O(k)$ to $O(log(k))$. For Hamiltonian Cycles the matrix no longer grows quadratically in the number of nodes, as currently, but linearly in the number of edges and logarithmically in the number of nodes.

We present these two formulations not as mathematical expressions, as most QUBO formulations are, but as meta-algorithms that facilitate the design of more complex QUBO formulations and allow easy reuse in larger and more complex QUBO formulations.

KEYWORDS

QUBO, Ising, Satisfiability, k-SAT, Hamiltonian Cycle

ACM Reference Format:
Jonas Nüßlein, Thomas Gabor, Claudia Linnhoff-Popien, and Sebastian Feld. 2022. Algorithmic QUBO Formulations for k-SAT and Hamiltonian Cycles. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3520304.3533952

1 INTRODUCTION

Solving optimization and decision problems is a central task in computer science with numerous real-world applications [2, 6, 12]. However, not all problems can be solved in polynomial time, assuming that the conjecture $P \neq NP$ [11] holds. The complexity class P contains the problems which can be solved with a polynomial-time algorithm. In NP are the problems for which a given solution can be verified in polynomial time, whereby $P \subseteq NP$. Another complexity class worth mentioning is NP-hard, which consists of the problems for which probably no polynomial solution algorithm exists. A problem is called NP-complete if it is in both NP and NP-hard.

However, numerous relevant problems, such as the Traveling Salesman Problem (TSP) or Satisfiabiliy (SAT), lie in NP-hard [2, 6, 12, 21]. For this reason, among others, there has been a recent growth of interest in quantum computers, with the hope that their non-deterministic computations will provide advantages in solving optimization and decision problems.

There are two basic approaches to quantum computing, the Quantum Gate Model [19, 34, 37] and Adiabatic Quantum Computing [1, 22], with Quantum Annealing [15, 28] in particular. While Quantum Annealing is specifically designed to solve optimization problems, Gate Model Quantum Computing is somewhat more general, with the Quantum Approximate Optimization Algorithm (QAOA) [14] existing here, which can solve optimization problems. Since current quantum computers are still small and error-prone [32], recent work has also been done on quantum-inspired classical computers, such as the Digital Annealer [3], which is also specifically designed to solve optimization problems. All of the solution methods mentioned here use Quadratic Unconstrained Binary Optimization (QUBO) [16] or isomorphic Ising [4] as the formulation language for the optimization problems to be solved. Thus, in order to solve an optimization problem using the above methods, it must first be translated to QUBO. The translation should be as efficient as possible in order to be able to solve larger problem instances on the limited hardware.

In this paper, we would like to contribute by presenting two QUBO formulations for (Max) k-SAT and Hamiltonian Cycles which scale better than the currently existing ones. We further present the QUBO formulations not as mathematical expressions, but as algorithmic functions, making them easier to understand and also easier to reuse, for example as part of bigger and more complex QUBO formulations. In arithmetic QUBO formulations (for example using pyqubo [38]), all objectives are formulated together as a QUBO matrix. The idea behind our algorithmic QUBOs is that imperative control structures are used to select which objectives (with which parameters) are inserted into the QUBO matrix.

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07…$15.00
https://doi.org/10.1145/3520304.3533952

2 BACKGROUND

In this section, we want to formally introduce the problem classes QUBO, (Max) SAT, and Hamiltonian Cycles.

2.1 Quadratic Unconstrained Binary Optimization (QUBO)

Given a symmetric (n×n)-matrix Q and a binary vector x of length n, a QUBO [29] is a function of the form:

$$H(x, Q) = \sum_{i=1}^{n} \sum_{j=i}^{n} x_i \, x_j \, Q_{ij} \tag{1}$$

The function H is called Hamiltonian. We will refer to the matrix Q as the "QUBO matrix" in this paper.

The optimization task is to find a binary vector x which is as close to the optimum $x^* = argmin_x \; H(x, Q)$ as possible. This we want to delegate to the machine. Our task, on the other hand, is to specify a function for a problem class such as TSP, which maps a concrete problem instance P from this problem class to a QUBO matrix Q in such a way that the solution p (e.g. the shortest route in TSP) for the problem instance P can be derived from the solution vector x^*.

Numerous well-known optimization problems such as boolean formula satisfiability (SAT), knapsack, graph coloring, the traveling salesman problem (TSP), or max clique have already been translated to QUBO form [9, 10, 17, 25, 26]. Throughout literature these translations have mostly been given via purely arithmetic expressions without imperative control structures [5, 9, 17, 25, 26].

To solve a QUBO matrix using quantum annealing (QA), it must first be embedded on a special graph [25, 29, 31], where the nodes represent the qubits and the edges represent the connections. In this paper, we will not elaborate on embedding the QUBO matrices of our algorithms on these graphs.

2.2 Satisfiability (SAT)

Satisfiability (SAT) is one of the best known and most fundamental problems in computer science. In SAT a set of boolean variables X is given, as well as a boolean formula f, which contains only variables from X. The question to be answered is whether there is an assignment $\underline{X}$ for the variables X with truth values (1 and 0), so that f evaluates to 1. If there is such an assignment $\underline{X}$, it is called model for f and f is called satisfiable.

Any boolean function can be written in conjunctive normal form. Conjunctive normal form is a conjunction over any number of clauses, where a clause C is a disjunction over any number of literals (a literal l is a variable x or its negation $\neg x$). Assuming there are $|f|$ clauses, then f can be written as

$$f = \bigwedge_{i=1}^{|f|} \bigvee_{l \in C_i} l \tag{2}$$

Thus, for f to evaluate to 1, each clause $(\bigvee_{l \in C_i} l)$ must evaluate to 1. k-SAT is a special case of SAT where each clause contains exactly k literals. For $k \geq 3$ k-SAT is NP-complete [20], for 2-SAT there exists a polynomial algorithm [24]. Max k-SAT is the optimization problem where the aim is to find the assignment that satisfies as many clauses as possible. If f is satisfiable, then every solution for Max k-SAT is also a model for k-SAT. Algorithms for Max k-SAT therefore trivially also solve k-SAT.

2.3 Hamiltonian Cycle

In graph theory a cycle is a path following the edges of that graph where only the start and the end vertices are equal. In the Hamiltonian Cycle problem, a graph with a set of vertices V and a set of directed or undirected edges E is given. The question now is whether there is a cycle that visits every vertex of the graph. For each consecutive pair of vertices (a, b) in the cycle there must be a corresponding (directed) edge in E. The Hamiltonian Cycle problem is NP-complete as well [23].

3 RELATED WORK

Chancellor et al. [8] presented a QUBO formulation for Max k-SAT in which each clause is represented by Hamiltonian $H^{(2)}_{clause}$. For a clause with k literals (σ^z_i), k ancillae $(\sigma^z_{i,a})$ are necessary. Each clause Hamiltonian is given via

$$H^{(2)}_{clause} = J \sum_{i=1}^{k} \sum_{j=1}^{i-1} c(i)c(j)\sigma^z_i\sigma^z_j + h \sum_{i=1}^{k} c(i)\sigma^z_i + J^a \sum_{i=1}^{k} \sum_{j=1}^{i-1} c(i)\sigma^z_i\sigma^z_{j,a} + \sum_{i=1}^{k} h^a_i\sigma^z_{i,a} \tag{3}$$

where $c(i) = 1$ if literal i is positive (not a negation) and $c(i) = -1$ if literal i is a negation. The parameters J, h, and h^a_i are chosen as $J = J^a$, $h = -J^a$, $h^a_i = -J^a(2i - k) + q_i$ with

$$q_i = \begin{cases} g/2 & \text{if } i = 0 \\ 0 & \text{else,} \end{cases}$$

where $g/2 \ll J_a$. The Hamiltonian of the entire formula is then obtained by adding all the clause Hamiltonians.

Choi [9, 10] used a different approach for 3-SAT, which can be easily generalized to k-SAT and also grows linearly in k.

Lucas [26] presented the current state-of-the-art QUBO formulation for Hamiltonian Cycle, which grows quadratically in the number of vertices N. The QUBO matrix is given via

$$H = A \sum_{v=1}^{n} \left(1 - \sum_{j=1}^{N} x_{v,j}\right)^2 + A \sum_{j=1}^{n} \left(1 - \sum_{v=1}^{N} x_{v,j}\right)^2 + A \sum_{(uv) \notin E} \sum_{j=1}^{N} x_{u,j}x_{v,j+1} \tag{4}$$

The first summand represents the constraint that each node appears at exactly one position of the cycle, the second summand represents the constraint that there is only one node at each position of the cycle, and the third summand ensures that two adjacent nodes in the cycle really have an edge connecting them.

Vargas-Calderón et al. [36] presented a TSP formulation (Hamiltonian Cycle is TSP with equal edge weights), which requires only $N \cdot \log(N)$ qudits, where N-level qudits must be available. Our approach, however, requires only qubits (2-level qudits).

Besides quantum annealing, there are also gate model approaches to solve the Hamiltonian Cycle problem [27, 33], where [33] is an explicit algorithm for Hamiltonian Cycles, while [27] proposes the Grover algorithm [18] to find the Hamiltonian Cycle, whereby Grover provides quadratic speedup compared to brute force.

Although there has been an approach [30] to code the constrains of an optimization problem with a programming language, which allows for loops and control structures which is then translated via hard-wired mechanisms into a QUBO matrix, still most QUBO formulations are presented via a mathematical expression as we have seen above. Our approach (algorithmic QUBO formulation) goes in a different direction: We formulate the QUBO not directly (neither via a mathematical expression nor via classical code which can be directly translated to a QUBO matrix) but indirectly via classical code which *outputs* a QUBO. If we want to consider the QUBO matrix as the "program" for a quantum annealer, then the classical program thus acts as a meta-program.

In this paper we want to show why algorithmic QUBO formulations are superior to mathematical QUBO formulations since mathematical QUBO formulations get complicated and chaotic very quickly as the complexity of the underlying constraints increases. We present algorithmic QUBO formulations for (Max) k-SAT and Hamiltonian Cycles for which we make use of base2 encoding, which was already mentioned in [25], [26], [35].

4 ALGORITHMIC QUBO FORMULATION FOR (MAX) k-SAT

In arithmetic QUBO formulations (for example using pyqubo [38]), all objectives are formulated together as a QUBO matrix. The idea behind our algorithmic QUBOs is that imperative control structures are used to select which objectives (with which parameters) are inserted into the QUBO matrix (at which position in the matrix, i.e., with which qubits).

As mentioned in Section 3, the QUBO matrix grows linearly in k in the current state-of-the-art formulation for (Max) k-SAT. We now show an algorithmic QUBO formulation for (Max) k-SAT which grows only logarithmically in k. We first present the basic idea behind our algorithm, followed by the more detailed description of the algorithm, an analysis of the scaling function of the QUBO matrix as a function of k, and a discussion.

4.1 Idea

Assume C is a clause consisting of k literals. The idea is to model a linear equation which counts how many literals of C are satisfied. To count from 0 to k we need $h = \lceil \log_2 (k+1) \rceil$ binary variables $(A_1, \ldots, A_h)$, which we call auxiliary variables. If the clause is satisfied by at least one literal, at least one auxiliary will have value 1. We can thus form an artificial clause $newC = [A_1 \vee \ldots \vee A_h]$, which will evaluate to true if C evaluates to true and which evaluates to false if C evaluates to false. We proceed recursively with anchor 2-SAT and 3-SAT for which we use the known formulations [8, 17]. Since the formulations for 2-SAT and 3-SAT actually solve Max 2-SAT and Max 3-SAT, our algorithm actually also solves Max k-SAT, which trivially solves k-SAT.

4.2 Algorithm

We first present the linear equation which counts by how many literals a clause is satisfied.

$n(C)$ is the number of negative literals in C and $x(l)$ returns the assignment (0 or 1) of the variable of literal l. $sign(l)$ returns -1 if l is a negation and +1 if it is not a negation. The linear equation is thus given by formula (5)

$$n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) = \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j) \tag{5}$$

The auxiliary variables (right side of the equation) model the number of literals that satisfy clause C. If all h auxiliary variables are 0, then clause C is not satisfied. For C to be satisfied, at least one literal must satisfy the clause and thus the right-hand side of the equation must be greater than 0, i.e., at least one of the h auxiliary variables must be 1.

Ensuring that at least one of the auxiliary variables is 1 corresponds to another clause $[A_1 \vee \ldots \vee A_h]$, which only has h (positive) literals.

We proceed in this way recursively until the clause consists of two or three literals, for which we then use the well-known formulations for OR (for clauses with two literals) or the 3-SAT matrices from [8, 17] as anchors of the recursion.

In order to integrate the linear equation into a QUBO, it must be represented as a quadratic optimization function. This is easily done by putting all terms of the equation on one side and then squaring it (see formula (6)):

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j) \right)^2 \tag{6}$$

Let f be the set of clauses, X the set of variables, and $n(C)$ the number of negative literals in the clause C. $C[i]$ is the i-th literal of clause C, $x(C[i])$ is therefore the assignment (0 or 1) of the corresponding variable of that literal. $x(A_j)$ is correspondingly the assignment of the j-th auxiliary variable. $len(C)$ returns the number of literals in clause C. Algorithm 1 shows the computation of the QUBO matrix Q.

To facilitate the understanding of this QUBO algorithm, we would like to demonstrate it with a small example.

Let $C = [-1, 2, 3, -4, -5]$ be a clause with five literals. For this clause three auxiliary variables (6,7,8) are needed, which model with the help of the objective (6), by how many literals the clause is satisfied. For example, for the assignment $1 = False, 2 = True, 3 = False, 4 = False, 5 = False$, C is satisfied by four literals (1, 2, 4 and 5). The three auxiliary variables would therefore have the assignment $6 = False, 7 = False, 8 = True$, since $2^0 \cdot 0 + 2^1 \cdot 0 + 2^2 \cdot 1 = 4$. The auxiliary variables (6,7,8) now form a new clause for which we proceed in an analogous manner. Since len(C)=3 is the anchor of our recursion, we use the well-known 3-SAT formulation with

the help of another auxiliary variable (9). If the clause C would not be satisfied (i.e. $1 = True, 2 = False, 3 = False, 4 = True, 5 = True$), the three auxiliary variables would have the assignment $6 = False, 7 = False, 8 = False$. Since (6,7,8) constitute a new clause and all three literals are positive and have the assignment *False* the new clause is also not satisfied.

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
1   init empty QUBO matrix Q;
2   for C ∈ f do
3       implementClause(Q, C);
    end
4   return Q;

implementClause(QUBO matrix: Q, Clause: C)
5   if len(C) = 2 then
6       Q ← OR(C);
7   else if len(C) = 3 then
8       X ← [A_1];
9       Q ← 3-SAT(C, A_1);
10  else
11      h = ⌈log_2 (len(C) + 1)⌉;
12      X ← [A_1 , ... , A_h];
13      Q ← formula (6);
14      newC = [A_1 ∨ .. ∨ A_h];
15      implementClause(Q, newC);
    end
16  return Q;
```

In the algorithm "$Q \leftarrow Formula$" can be interpreted as the addition of *Formula* to the current matrix Q (simple element-wise addition). Note that different auxiliary variables are needed for every clause. Thus, in lines 8 and 12 of the algorithm, the size of Q grows by 1 and h, respectively.

If f is satisfiable, then the assignment that satisfies f is encoded in the first $|X|$ elements of the solution vector x^*. The QUBO matrix built from this algorithm only grows logarithmically in k and it needs additionally one QUBO variable for each SAT variable. The algorithm uses imperative statements and control structures as well as recursion. Besides that, it reuses the QUBO formulation of the OR function, the 3-SAT function, and recursively the function itself. Once a better way to encode 3-SAT is found, we only have to plug in the new code in lines 8 and 9 of our algorithm, without any further changes.

4.3 Discussion

As in [8], we also formulate each clause individually and the QUBO matrix for the whole formula is then obtained as the (element-wise) addition of all clause QUBO matrices. Figure 1 plots the scaling behaviors of the previous state-of-the-art QUBO formulation and our QUBO algorithm for k-SAT. The x-axis of this graph represents the number of literals in the clause k and the y-axis represents the

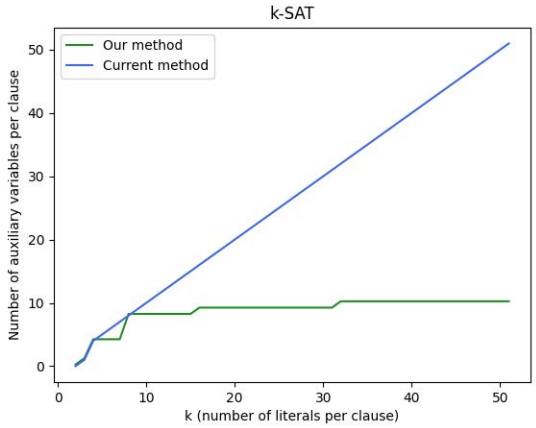

Figure 1: The number of necessary auxiliary variables per clause as a function of the number of literals in that clause for the current state-of-the-art implementation for k-SAT [8] compared to our QUBO algorithm for k-SAT.

number of auxiliary variables needed. As can be seen, for $k = 8$ our algorithm still needs as many auxiliary variables as [8], i.e., 8 auxiliary variables ($4 + 3 + 1$). Thus, our QUBO algorithm for k-SAT is really advantageous only for $k \gg 8$, which is not feasible on current quantum computers.

Therefore, to verify the correctness of our QUBO algorithm, we solved QUBO matrices using a classical QUBO solving method, more specifically QbSolv [7]. We created 30 random, solvable formulas for $k = 4, 6, 8, 10$ for different clause-to-variable ratios. By "solvable" we mean that we used a classical SAT solver (Minisat [13]) to check whether a model exists for the formula (i.e., whether it is satisfiable). Then we solved the QUBO matrices using QbSolv. For all formulas, the best QUBO solution corresponded to a model of the formula, which means that the QUBO formulation was indeed correct for these 120 formulas.
As stated earlier, we use a base2 encoding for both algorithms presented in this paper. As analyzed in [35], current hardware seems to have more problems with base2 encodings than with one-hot or unary encodings. Therefore, it seems to us that an important future work regarding the hardware is the improvement of the solving capability of base2 encodings.

Algorithmic QUBOs, such as the QUBO algorithm for (Max) k-SAT presented in this chapter, differ from arithmetic QUBOs (such as pyqubo) in that not all specified objectives are formulated together as a QUBO, but instead imperative control structures can be used to control which objectives (with which parameters) are inserted (at which position of the matrix, i.e. using which qubits) into the QUBO matrix using a special (element-wise) QUBO addition function $Q \leftarrow Formula$.

4.4 Size of Q as a function of k

Assuming the original boolean formula f has $|X|$ variables and $|f|$ clauses, the size of the QUBO matrix now no longer grows linearly

in k, as in the current state-of-the-art, but logarithmically. The size of the resulting QUBO matrix Q (the quadratic $(n \times n)$-matrix) can be recursively calculated via $n = |X| + |f| \cdot r(k)$, where

$$r(k) = \begin{cases} 0 & \text{if } k = 2, \\ 1 & \text{if } k = 3, \\ \lceil \log_2(k+1) \rceil + r(\lceil \log_2(k+1) \rceil) & \text{if } k \geq 4. \end{cases}$$

5 ALGORITHMIC QUBO FORMULATION FOR HAMILTONIAN CYCLE

In this section, we consider the Hamiltonian circle problem. Without loss of generality we assume all edges to be directed. We use the logarithm trick [25, 26] and present an algorithmic QUBO formulation which only grows linearly in the number of edges and logarithmically in the number of nodes of the graph, which together can be sub-quadratic given that not all possible edges are used.

5.1 Idea

The current SOTA formulation for Hamiltonian Cylce formulates the optimization problem as a search for which of the $n-1$ vertices is positioned at which of the $n-1$ positions of the cycle.

We propose a different approach which is based on the search for the edges which together constitute the Hamiltonian Cycle. Thus, a solution x consists of the position numbers in the cycle for all directed edges of the graph. Each position number is represented by $z = \lceil log_2(|V|+1) \rceil$ binary variables of x.

$P_{ab} \in \mathbb{N}_0$ is the position of the edge (a, b), which is given by the z position variables $x_{(a,b),1}, ..., x_{(a,b),z}$ and the linear equation

$$P_{ab} = \sum_{i=1}^{z} 2^{(i-1)} \cdot x_{(a,b),i} \tag{7}$$

Here, $x_{(a,b),i}$ is the assignment value (0 or 1) of the i-th variable of the subgroup of the QUBO solution vector x belonging to the edge (a, b). If the edge (a, b) is not part of the cycle, then by definition $P_{ab} = 0$.

The Hamiltonian Cycle problem is fully described by the following three hard-constraints:

(1) The continuation of the cycle is deterministic, i.e., for all edge pairs (a, b) and (c, d) in the Hamiltonian Cycle $a \neq c$ and $b \neq d$ holds.
(2) Every edge has a continuation, i.e., if the edge (a, b) is part of the cycle, there exists an edge (b, c) for some c in the cycle.
(3) There is an edge out of the start node V_1 (an arbitrary node of the graph) at position 1 and an edge back into the start node at position $|V|$, where $|V|$ is the number of vertices of the graph.

Constraint (1) is very easy to implement by preventing the simultaneous activation of two variables from the subgroups of two conflicting edges by an appropriately high value in the matrix Q. The value $2|V|^2$ is sufficiently high. Constraint (2) can be ensured by the following objective: $Q \leftarrow (P_{bc} - P_{ab} - 1)^2$. This objective ensures that if the edges (a, b) and (b, c) are part of the Hamiltonian Cycle, then for the position P_{bc} it holds that $P_{bc} = P_{ab} + 1$.

Constraint (3) actually allows us to reduce the number of needed variables further: Since edges of the form (V_1, a) must be either at position 0 (i.e., not part of the Hamiltonian Cycle) or at position 1, only $z = 1$ variables are needed for these edges. The same holds for edges of the form (z, V_1), which must be either at position 0 or $|V|$. The objectives that model these constraints as a quadratic function are given by

$$Q \leftarrow (P_{V_1 a} - 1)^2 \quad \text{and} \quad Q \leftarrow (P_{z V_1} - |V|)^2 .$$

5.2 Algorithm

To summarize the idea, an edge (b, c) which has no contact with the starting node (i.e., $b \neq V_1$ and $c \neq V_1$) must satisfy the two linear equations $P_{bc} = P_{ab} + 1$ and $P_{cd} = P_{bc} + 1$. The addition of the two associated quadratic optimization problems $(P_{bc} - P_{ab} - 1)^2$ and $(P_{cd} - P_{bc} - 1)^2$ results in: $(-2P_{bc}P_{ab} - 2P_{cd}P_{bc}) + 2{P_{bc}}^2 + ({P_{ab}}^2 + 2P_{ab}) + ({P_{cd}}^2 - 2P_{cd})$. As one can see, the part concerning only the edge (b, c) is given by $2{P_{bc}}^2$. Plugging in the definition of P_{bc} results in $2\left(\sum_{i=1}^{z} 2^{(i-1)} \cdot x_{(b,c),i}\right)^2$. This constraint is now in a usual quadratic form, which can be added (element-wise addition) to the QUBO matrix.

If the edge is an edge from the start node, i.e., of the form (V_1, a), then the sum of the two objectives $(P_{V_1 a} - 1)^2$ and $(P_{ab} - P_{V_1 a} - 1)^2$ results in $2{P_{V_1 a}}^2$ as the part which only concerns the edge (V_1, a). If it is an edge into the start node, i.e., of the form (z, V_1), then the entries for the QUBO matrix Q are the sum of $(P_{z V_1} - |V|)^2$ and $(P_{z V_1} - P_{yz} - 1)^2$. Expanded, the part concerning only the edge (z, V_1) is equal to $2{P_{z V_1}}^2 - 2P_{z V_1} \cdot (|V| + 1)$. Putting it all together, the QUBO is generated by Algorithm 2.

Note that only a few of the above constraints are actually satisfied by the correct solution x^*. For example, given there is an edge (a, b) and three possible continuations (b, c), (b, d) and (b, e). For each edge pair there is a corresponding optimization problem: $(P_{bc} - P_{ab} - 1)^2$, $(P_{bd} - P_{ab} - 1)^2$ and $(P_{be} - P_{ab} - 1)^2$.

However, due to constraint (1), only one of the edges (b, c), (b, d) and (b, e) will be part of the cycle and thus have a position number not equal to 0. The other two edges will have position number 0. Thus, these two optimization problems are not satisfied (minimized). For this reason, it would have led to an incorrect result if we had simply inserted all the optimization problems directly into the QUBO matrix, since this would have caused, for example, the term $2P_{ab}^2$ to be inserted into the QUBO matrix three times (in our example), even though only one of these optimization problems can be satisfied.

Our solution to this problem is simple: we just extracted the part of the optimization problems that concerns only one edge and inserted this part *exactly once* into the QUBO matrix.

A brief word on the value $H(x, Q)$: H is usually called "energy", following the tradition of physics. The energy for the optimal solution x^* for the presented QUBO algorithm for Hamiltonian Cycle is given by $H(x^*, Q) = -|V| \cdot (|V| + 1)$, so we can already infer from the energy value of a solution whether it is indeed correct or not.

Algorithm 2: QUBO algorithm for Hamiltonian Cycles

```
fillQ(Vertices : V, Edges : E)
init empty QUBO matrix Q;
for (a, b) ∈ E do
    if b = V_1 then
        Q ← 2P_ab^2 − 2P_ab · (|V| + 1);
    else
        Q ← 2P_ab^2;
    end
    for (c, d) ∈ E do
        if a = c XOR b = d then
            Q[(a, b), (c, d)] ← 2|V|^2;
        else if (b = c ∧ b ≠ V_1) ∨ (a = d ∧ a ≠ V_1) then
            Q ← −2P_cd P_ab;
        end
    end
end
return Q;
```

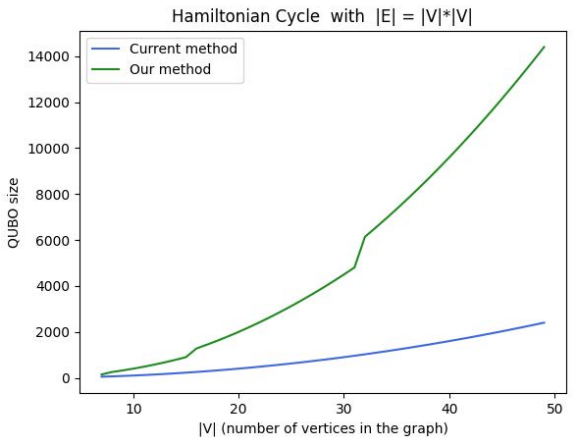

Figure 2: Size of the QUBO matrix as a function of the number of nodes of the graph for the case when the graph is fully-connected.

5.3 Discussion

To verify the correctness of our QUBO algorithm for Hamiltonian Cycles, we implemented it and applied it to a total of 100 randomly generated graphs with up to 40 nodes and different meshing degrees and solved it using QbSolv [7]. Since Hamiltonian Cycles, like k-SAT, is NP-complete (which means it is also contained in the complexity class NP), it is easy to check whether the solution is indeed correct or not. In all cases, the best QUBO solution corresponded to a valid Hamiltonian cycle which means that the QUBO formulation was indeed correct for these 100 graphs.

Since in our QUBO algorithm, unlike in [26], the size of the QUBO matrix depends on the number of edges of the graph, we evaluated the scaling behavior for different values of $|E|$ (number of edges) as a function of $|V|$ (number of nodes). Figure 2 shows the scaling behavior for the case of fully-connected graphs.[1] The x-axis for the figure describes the number of nodes of the graph and the y-axis represents the size of the resulting QUBO matrix.

As can be seen, our algorithm scales significantly worse than the current state-of-the-art formulation in the case of the fully-connected graph.

Figure 3 shows the case where the number of edges grows only linearly in the number of nodes of the graph (in this example $|E| = 4|V|$). Here, our method shows a much better scaling behavior than the current method.

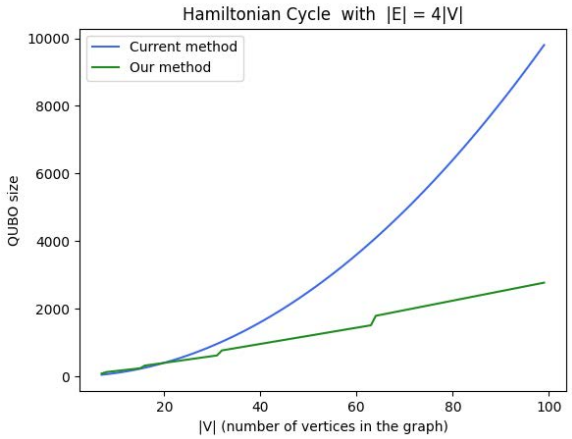

Figure 3: Size of the QUBO matrix as a function of the number of nodes of the graph for the case where the number of edges of the graph grows only linearly in the number of nodes ($|E| = 4|V|$).

5.4 Size of Q as a function of $|V|$ and $|E|$

The graph contains at most $2|V|$ edges from or into the start node, which only require one variable each to model the position. For the remaining edges, $\lceil log_2(|V|+1) \rceil$ position variables are needed to model the position of each edge. Since there are $|E|$ edges, the size of the QUBO matrix is equal to $n \leq |E| \cdot \lceil log_2(|V|+1) \rceil$

In the worst case of a fully-connected graph, this QUBO formulation is worse than state-of-the-art. But in settings where the number of edges grows subquadratically as a function of the number of nodes of the graph (as for examples in social networks), the size of Q also grows sub-quadratically. If the number of edges grows linearly in the number of vertices, then this algorithmic QUBO formulation only grows by $O(|V| \cdot log(|V|))$ instead of $O(|V| \cdot |V|)$.

[1] We only want to show the scaling behavior here and disregard the meaningfulness, since fully-connected graphs of course trivially contain Hamiltonian Cycles.

6 CONCLUSION

In this paper, we presented QUBO algorithms (meta-programs) for (Max) k-SAT and Hamiltonian Cycles. Our algorithmic QUBOs differ from arithmetic QUBOs (such as pyqubo) in that not all specified objectives are formulated together as a QUBO, but instead imperative control structures can be used to control which objectives are inserted into the QUBO matrix (with which parameters and with which qubits) using a special (element-wise) QUBO addition function $Q \leftarrow \mathit{Formula}$.

Our QUBO algorithm for (Max) k-SAT only grows logarithmically in k, while the current best formulation grows linearly in k. We have also shown an algorithmic QUBO formulation for the Hamiltonian Cycle problem that is more efficient if the edges are sufficiently sparse.

To enable larger and more complex QUBO formulations, the devlopment of these formulations must become simpler and more concise. In this paper, we have filled the QUBO piecewise with reusable functions (meta-programs) using the familiar control structures from classical programming, such as loops, branching and recursion. We have exemplified this method on (Max) k-SAT and Hamiltonian Cycles, but we see potential for improvement in numerous other problems in the future.

The Python code for the two QUBO algorithms presented here is available on GitHub:
https://github.com/JonasNuesslein/AlgorithmicQUBOs

REFERENCES

[1] Tameem Albash and Daniel A Lidar. 2018. Adiabatic quantum computation. *Reviews of Modern Physics* 90, 1 (2018), 015002.
[2] David Applegate, Robert Bixby, Vasek Chevátal, and William Cook. 2006. *The traveling salesman problem: a computational study.*
[3] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G Katzgraber. 2019. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics* 7 (2019), 48.
[4] Francisco Barahona. 1982. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General* 15, 10 (1982), 3241.
[5] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. 2018. Solving SAT and MaxSAT with a Quantum Annealer: Foundations, Encodings, and Preliminary Results. (2018).
[6] Avrim Blum and Ronald Rivest. 1988. Training a 3-node neural network is NP-complete. (1988).
[7] M Booth, SP Reinhardt, and A Roy. [n. d.]. Partitioning optimization problems for hybrid classica/quantum execution (2017).
[8] Nick Chancellor, S Zohren, P A Warburton, S C Benjamin, and S Roberts. 2016. A Direct Mapping of Max k-SAT and High Order Parity Checks to a Chimera Graph. (2016).
[9] Vicky Choi. 2010. Adiabatic Quantum Algorithms for the NP-Complete Maximum-Weight Independent Set, Exact Cover and 3SAT Problems. (2010).
[10] Vicky Choi. 2011. Different Adiabatic Quantum Optimization Algorithms for the NP-Complete Exact Cover and 3SAT Problems. (2011).
[11] Stephen Cook. 2000. The P versus NP problem. *Clay Mathematics Institute* 2 (2000).
[12] Sanjoy Dasgupta. 2008. *The Hardness of K-means Clustering.*
[13] Niklas Eén and Niklas Sörensson. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*. Springer, 502–518.
[14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).
[15] Aleta Berk Finnila, MA Gomez, C Sebenik, Catherine Stenson, and Jimmie D Doll. 1994. Quantum annealing: A new method for minimizing multidimensional functions. *Chemical physics letters* 219, 5-6 (1994), 343–348.
[16] Fred Glover, Gary Kochenberger, and Yu Du. 2018. A tutorial on formulating and using QUBO models. *arXiv preprint arXiv:1811.11538* (2018).
[17] Fred Glover, Gary Kochenberger, and Yu Du. 2019. Quantum Bridge Analytics I: A Tutorial on Formulating and Using QUBO Models. (2019).
[18] Lov Grover. 1996. A fast quantum mechanical algorithm for database search. (1996).
[19] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.
[20] R Impagliazzo and R Paturi. 1999. Complexity of k-SAT. (1999).
[21] Russell Impagliazzo and Ramamohan Paturi. 2001. On the complexity of k-SAT. *J. Comput. System Sci.* 62, 2 (2001), 367–375.
[22] William M Kaminsky and Seth Lloyd. 2004. Scalable architecture for adiabatic quantum computing of NP-hard problems. *Quantum computing and quantum bits in mesoscopic systems* (2004), 229–236.
[23] Richard Manning Karp. 1972. Reducibility Among Combinatorial Problems. (1972).
[24] M R Krom. 1967. The Decision Problem for a Class of First-Order Formulas in Which all Disjunctions are Binary. (1967).
[25] Bas Lodewijks. 2020. Mapping NP-hard and NP-complete Optimisation Problems to Quadratic Unconstrained Binary Optimisation Problems. (2020).
[26] Andrew Lucas. 2014. Ising formulations of many NP problems. (2014).
[27] Anuradha Mahasinghe, Richard Hua, Michael Dinneen, and Rajni Goyal. 2019. Solving the Hamiltonian Cycle Problem using a Quantum Computer. (2019).
[28] Catherine C McGeoch. 2014. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing* 5, 2 (2014), 1–93.
[29] Gary Mooney, Sam Tonetto, Charles Hill, and Lloyd Hollenberg. 2019. Mapping NP-hard Problems to restructed Adiabatic Quantum Architectures. (2019).
[30] Scott Pakin. 2018. Performing Fully Parallel Constraint Logic Programming on a Quantum Annealer. (2018).
[31] Date Prasanna, Robert Patton, Catherine Schuman, and Thomas Potok. 2019. Efficiently embedding QUBO problems on adiabatic quantum computers. (2019).
[32] Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout. 2022. Measuring the capabilities of quantum computers. *Nature Physics* 18, 1 (2022), 75–79.
[33] T. Rudolph. 1995. Quantum Computing Hamiltonian cycles. (1995).
[34] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.
[35] Kensuke Tamura, Tatsuhiko Shirai, Hosho Katsura, Shu Tanaka, and Nozomu Togawa. 2021. Performance Comparison of Typical Binary-Integer Encodings in an Ising Machine. *IEEE Access* 9 (2021), 81032–81039. https://doi.org/10.1109/ACCESS.2021.3081685
[36] Vladimir Vargas-Calderón, Nicolas Parra-A., and Herbert Vinck-Posada. 2021. Many-Qudit representation for the Travelling Salesman Problem Optimisation. (2021).
[37] Ehsan Zahedinejad and Arman Zaribafiyan. 2017. Combinatorial optimization on gate model quantum computers: A survey. *arXiv preprint arXiv:1708.05294* (2017).
[38] Mashiyat Zaman, Kotaro Tanahashi, and Shu Tanaka. 2021. PyQUBO: Python library for mapping combinatorial optimization problems to QUBO form. *arXiv preprint arXiv:2103.01708* (2021).