Quadratic Unconstrained Binary Optimization

Sam Muir

June 2024

1 Introduction

The Quadratic Unconstrained Binary Optimization (QUBO) problem is an optimization problem which consists of finding the binary vector that minimizes a quadratic function. The QUBO model encompasses many important combinatorial optimization problems which have applications in computer science, operations research and physics. QUBO problems are also at the forefront of quantum computing since they are particularly well-suited to being solved using quantum annealing.

In this project we begin by covering how QUBO models can be constructed. Then we will explore QUBO formulations of some well known combinatorial problems. We can apply these insights to the graph isomorphism, graph matching and the quadratic assignment problems. Finally, we discuss the feasibility of using quantum annealers to solve QUBO problems.

This project will follow 'Continuous optimization methods for the graph isomorphism problem' by Stefan Klus and Patrick Gelß [3].We also included MATLAB code for the QUBO implementations of each problem. This code has been uploaded to GitHub at https://github.com/muirsam/QUBO/.

This project was funded by the Edinburgh Mathematical Society.

2 What are QUBO problems?

Definition 1 *A QUBO problem is an optimisation problem of the form*

$$\min_{\mathbf{x}\in B(n)} f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x} + d,$$

where $B(n)$ is the set of vectors with n binary entries, $Q \in \mathbb{R}^{n\times n}$ and $d \in \mathbb{R}$.

Note that since d is a constant, we can omit it without changing the optimal solution of the function.

At first glance this definition seems quite limited. But using a few tricks we can convert many problems to this form.

2.0.1 Simple Example

Suppose we are given a set of four numbers $\alpha = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and we need to find the two-element subset of α with the largest sum. We can convert this into a QUBO problem as follows.

First define the binary decision vector $\mathbf{x} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}^T$. Then the sum function can be written as $\sum_{i=1}^{4} \alpha_i x_i$. But since x_i is binary, $x_i = x_i^2$ and so we can express the sum function as

$$\sum_{i=1}^{4} \alpha_i x_i = \sum_{i=1}^{4} \alpha_i x_i^2 = \mathbf{x}^T \begin{pmatrix} \alpha_1 & 0 & 0 & 0 \\ 0 & \alpha_2 & 0 & 0 \\ 0 & 0 & \alpha_3 & 0 \\ 0 & 0 & 0 & \alpha_4 \end{pmatrix} \mathbf{x}.$$

Maximising this function will give the subset with the largest sum. But we want the largest two-element subset so we need to add a penalty term to guarantee that the solution is a two-element subset.
By writing the two-element constraint as $\sum_{i=1}^{n} x_i = 2$, we can construct the penalty

$$\left(\sum_{i=1}^{4} x_i - 2 \right)^2 = \sum_{i=1}^{4} x_i \sum_{j=1}^{4} x_j - 4 \sum_{i=1}^{4} x_i + 4$$

which is 0 for two-element subsets and positive otherwise. Then since $x_i = x_i^2$ this penalty is a quadratic form plus a constant

$$\begin{aligned} \left(\sum_{i=1}^{4} x_i - 2 \right)^2 &= \sum_{i=1}^{4} \sum_{j=1}^{4} x_i x_j - 4 \sum_{i=1}^{4} x_i^2 + 4 \\ &= \mathbf{x}^T \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \mathbf{x} + \mathbf{x}^T \begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & -4 \end{pmatrix} \mathbf{x} + 4 \\ &= \mathbf{x}^T \begin{pmatrix} -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 1 & 1 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{pmatrix} \mathbf{x} + 4. \end{aligned}$$

Since the sum of α is bounded above by $\lambda = \sum_{i=1}^{4} |\alpha_i|$ and the penalty is a positive integer we have

$$\lambda \left(\sum_{i=1}^{4} x_i - 2 \right)^2 \geq \sum_{i=1}^{4} \alpha_i x_i$$

for subsets which do not have exactly two elements. Then we can construct the function

$$\sum_{i=1}^{4} \alpha_i x_i - 2\lambda\left(\sum_{i=1}^{4} x_i - 2\right)^2$$

which is equal to the sum function for a two-element subset and less than or equal to $-\lambda$ otherwise. But $-\lambda$ is also a lower bound for the sum and therefore the optimal value of this function will be the two-element subset with the largest sum.
So the function we want to maximise is

$$\begin{aligned}
&\sum_{i=1}^{4} \alpha_i x_i - 2\lambda\left(\sum_{i=1}^{4} x_i - 2\right)^2 \\
&= \mathbf{x}^T \begin{pmatrix} \alpha_1 & 0 & 0 & 0 \\ 0 & \alpha_2 & 0 & 0 \\ 0 & 0 & \alpha_3 & 0 \\ 0 & 0 & 0 & \alpha_4 \end{pmatrix} - 2\lambda \mathbf{x}^T \begin{pmatrix} -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 1 & 1 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{pmatrix} \mathbf{x} - 8\lambda.
\end{aligned}$$

Now we can write the problem as

$$\max_{\mathbf{x}\in B(n)} \mathbf{x}^T \begin{pmatrix} \alpha_1 + 6\lambda & -2\lambda & -2\lambda & -2\lambda \\ -2\lambda & \alpha_2 + 6\lambda & -2\lambda & -2\lambda \\ -2\lambda & -2\lambda & \alpha_3 + 6\lambda & -2\lambda \\ -2\lambda & -2\lambda & -2\lambda & \alpha_4 + 6\lambda \end{pmatrix} \mathbf{x} - 8\lambda.$$

Then we can negate this and omit the constant term to get

$$\min_{\mathbf{x}\in B(n)} \mathbf{x}^T \begin{pmatrix} -\alpha_1 - 6\lambda & 2\lambda & 2\lambda & 2\lambda \\ 2\lambda & -\alpha_2 - 6\lambda & 2\lambda & 2\lambda \\ 2\lambda & 2\lambda & -\alpha_3 - 6\lambda & 2\lambda \\ 2\lambda & 2\lambda & 2\lambda & -\alpha_4 - 6\lambda \end{pmatrix} \mathbf{x}$$

where $\lambda = \sum_{i=1}^{4} |\alpha_i|$. This is the QUBO formulation of the problem.

We can solve this QUBO in MATLAB as follows

```
a = [a_1, a_2, a_3, a_4];
lambda = sum(abs(a));
Q = diag(-a - 8*lambda) + 2*lambda*ones(4);

solve(qubo(Q))
```

In this example we were able to find a theoretical penalty weight λ which ensured the solutions were two-element subsets. For most sets α we could have used a smaller weight and obtained the same solution.
Since it is not always practical to derive the optimal weight and larger weights

increase the runtime of most QUBO solvers, it is common practise to estimate the weights and check the solution for feasibility instead.

For more examples and a list of penalties corresponding to more complicated constraints, see [2, p. 10]. This tutorial improved my understanding of QUBO problems immensely.

2.1 Maximal Independent Set problem

The maximal independent set of a graph is the largest subset of the vertex set such that no two vertices share an edge.

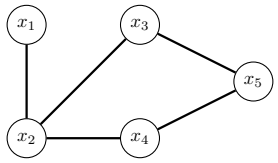


Figure 1: This graph has maximal independent set $\{x_1, x_3, x_4\}$.

The maximal independent set problem is identifying the maximal independent set. We can create a QUBO formulation of this problem as follows.

Suppose the vertices of the graph are labelled $x_1, \ldots, x_n$. For each edge connecting vertices x_i and x_j, we add the constraint $x_i + x_j \leq 1$.
This constraint corresponds to the penalty $2x_ix_j - x_i - x_j$ which is 0 when both vertices are included and -1 otherwise. Since $x_a = x_a^2$ this penalty becomes

$$\begin{aligned} 2x_ix_j - x_i - x_j &= 2x_ix_j - x_i^2 - x_j^2 \\ &= \begin{pmatrix} x_i & x_j \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_i \\ x_j \end{pmatrix}. \end{aligned}$$

This penalty can then be expressed as $\mathbf{x}^T P_{E_{ij}} \mathbf{x}$ where $P_{E_{ij}}$ is the matrix of zeros except at the intersections of rows i and j and columns i and j. The entries at the intersections on the main diagonal are -1 and the others are 1.

Since the penalties are -1 when one vertex is not excluded and 0 otherwise, the maximal independent set will be the subset which minimises the sum of these penalties. So the maximal independent set corresponds to the solution of

$$\min_{\mathbf{x} \in B(n)} \mathbf{x}^T Q \mathbf{x},$$

where $Q = \sum_{E_{ij} \in E} P_{E_{ij}}$ and E is the set of edges of the graph.

Letting A be the adjacency matrix of the graph we can solve this in MATLAB as follows:

```
Q = zeros(n);
for i=1:n
    for j=1:n
        if A(i,j) == 1
            Q(i,i) = Q(i,i) - 1;
            Q(i,j) = Q(i,j) + 1;
        end
    end
end

solve(qubo(Q))
```

2.2 Linear Assignment problem

The linear assignment problem can be stated as:

Suppose we have n machines and n tasks which must be completed. We can assign any machine to do any task, but each task-machine pair has an associated cost. We want to allocate one machine to each task such that the sum of the costs is minimised.

If we let A be the matrix of machine-task costs then the problem can be expressed as:

$$\min_{P \in \mathcal{P}(n)} g(P) = \mathrm{tr}(PA)$$

where $\mathcal{P}(n)$ is the set of all $n \times n$ permutation matrices.

We model this problem as a QUBO as follows. First, let

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}, \quad P = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_{n+1} & x_{n+2} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n(n-1)+1} & x_{n(n-1)+2} & \dots & x_{n^2} \end{pmatrix}.$$

Then the function to be minimized is,

$$\begin{aligned} \mathrm{tr}(PA) &= \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \cdot \begin{pmatrix} a_{11} \\ \vdots \\ a_{n1} \end{pmatrix} + \begin{pmatrix} x_{n+1} \\ \vdots \\ x_{2n} \end{pmatrix} \cdot \begin{pmatrix} a_{12} \\ \vdots \\ a_{n2} \end{pmatrix} + \dots + \begin{pmatrix} x_{n(n-1)+1} \\ \vdots \\ x_{n^2} \end{pmatrix} \cdot \begin{pmatrix} a_{1n} \\ \vdots \\ a_{nn} \end{pmatrix} \\ &= \sum_{i=1}^{n} \begin{pmatrix} x_{n(i-1)+1} \\ \vdots \\ x_{ni} \end{pmatrix} \cdot \begin{pmatrix} a_{1i} \\ \vdots \\ a_{ni} \end{pmatrix} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ji} x_{n(i-1)+j}. \end{aligned}$$

But x_i is binary so $x_i = x_i^2$, and

$$\begin{aligned}\operatorname{tr}(PA) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ji}x_{n(i-1)+j} \\ &= \mathbf{x}^T \operatorname{diag}(\operatorname{vec}(A))\mathbf{x}\end{aligned}$$

where $\operatorname{vec}(A) = \begin{pmatrix} A_1 \\ \vdots \\ A_n \end{pmatrix}$ and $A_1, \dots A_n$ are the columns of A.

So we can write the problem as

$$\min g(\mathbf{x}) = \mathbf{x}^T \operatorname{diag}(\operatorname{vec}(A))\mathbf{x} \tag{1}$$

where $\mathbf{x} = \operatorname{vec}(P^T)$ and P is a permutation matrix.

Now we need a penalty function which forces P to be a permutation matrix. To accomplish this we use the equation $C\mathbf{x} = d$ given in [3, p. 8] where

$$C = \begin{pmatrix} \mathbb{1}_n^T & & \\ & \ddots & \\ & & \mathbb{1}_n^T \\ e_1^T & \dots & e_1^T \\ \vdots & \ddots & \vdots \\ e_n^T & \dots & e_n^T \end{pmatrix} \in \mathbb{R}^{2n \times n^2}, \quad d = \mathbb{1}_{2n} \tag{2}$$

with $\mathbb{1}_n$ as the vector of n ones and e_i is the ith standard basis vector (with n entries).

This equation is 0 if and only if P is doubly stochastic. But if P is doubly stochastic it is also a permutation matrix since each entry is binary. So this constraint forces P to be a permutation matrix.

We can transform the constraint $C\mathbf{x} = d$ into the penalty $(C\mathbf{x} - d) \cdot (C\mathbf{x} - d)$. Adding this to (1) will favour solutions which are permutation matrices. Expanding this penalty we get,

$$\begin{aligned}(C\mathbf{x} - d) \cdot (C\mathbf{x} - d) &= C\mathbf{x} \cdot C\mathbf{x} - 2d \cdot C\mathbf{x} + d \cdot d \\ &= \mathbf{x}^T C^T C\mathbf{x} - 2\mathbf{x}^T C^T d + d^T d\end{aligned}$$

Then since $\mathbf{x}$ is binary $x_i = x_i^2$ the linear term $-2\mathbf{x}^T C^T d$ can be expressed as the quadratic term $-2\mathbf{x}^T \operatorname{diag}(C^T d)\mathbf{x}$. So the penalty term becomes

$$\mathbf{x}^T (C^T C - 2\operatorname{diag}(C^T d))\mathbf{x} + d^T d. \tag{3}$$

And so we can rewrite problem (1) as

$$\min g(\mathbf{x}) = \mathbf{x}^T \Big(\text{diag}(\text{vec}(A)) + \lambda C^T C - 2\lambda \text{diag}(C^T d) \Big) \mathbf{x} + \lambda d^T d$$

where λ is the penalty weight. Then we can drop the constant to get

$$\min g(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$$

where $Q = \text{diag}(\text{vec}(A) - 2\lambda C^T d) + \lambda C^T C$.

This can be solved with the code:

```
Ct = transpose(C)
v = reshape(A, 1, [])
Q = diag(v - weight*2*Ct*d) + weight*Ct*C

x = solve(qubo(Q));
P = transpose(reshape(x.BestX, n, []))
```

2.2.1 Example: Linear Assignment problem

Consider the cost matrix

$$A = \begin{pmatrix} 7 & 9 & 1 \\ 4 & 2 & 6 \\ 7 & 8 & 7 \end{pmatrix},$$

where each column corresponds to a task and each row corresponds to the machine costs.
Using A and setting $\lambda = 10$ we obtain the matrix

$$Q = \begin{pmatrix} -13 & 10 & 10 & 10 & 0 & 0 & 10 & 0 & 0 \\ 10 & -16 & 10 & 0 & 10 & 0 & 0 & 10 & 0 \\ 10 & 10 & -13 & 0 & 0 & 10 & 0 & 0 & 10 \\ 10 & 0 & 0 & -11 & 10 & 10 & 10 & 0 & 0 \\ 0 & 10 & 0 & 10 & -18 & 10 & 0 & 10 & 0 \\ 0 & 0 & 10 & 10 & 10 & -12 & 0 & 0 & 10 \\ 10 & 0 & 0 & 10 & 0 & 0 & -19 & 10 & 10 \\ 0 & 10 & 0 & 0 & 10 & 0 & 10 & -14 & 10 \\ 0 & 0 & 10 & 0 & 0 & 10 & 10 & 10 & -13 \end{pmatrix}.$$

Solving this gives $X = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}^T$ and so we have

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad PA = \begin{pmatrix} 7 & 8 & 7 \\ 4 & 2 & 6 \\ 7 & 9 & 1 \end{pmatrix}$$

which tells us that the minimum cost is $\text{tr}(PA) = 10$.

2.3 Sudoku

In the game of Sudoku we are given a partially filled 9×9 grid (81 squares) and the aim is to populate the remainder of the grid such that every column, row and box contains each of the numbers from 1 to 9. These boxes split the 9×9 grid into 9 non-overlapping 3×3 grids.

6	5				7	9		3
		2	1			6		
9				6	3			4
1	2	9						
3		4	9		8	1		
			3			4	7	9
		6		8		3		5
7	4		5					1
5	8	1	4				2	6

We can formulate the Sudoku problem as follows. Following [5, p. 1] the vector $\mathbf{x}$ will have 9 binary entries for each Sudoku square $x_{i,j,k} = x_{81i+9j+k}$. So each variable $x_{i,j,k}$ is defined as

$$x_{i,j,k} = \begin{cases} 1, & \text{if row } i \text{ column } j \text{ has value } k, \\ 0, & \text{otherwise.} \end{cases}$$

The constraints of this problem are:

	Constraint
a	Each column contains 1 to 9
b	Each row contains 1 to 9
c	Each box contains 1 to 9
d	Every square must have a value

Using [1, p. 326] the constraints above become

$$\begin{aligned}
&\text{(a)} \quad \sum_{i=1}^{9} x_{i,j,k} && = 1 \quad \forall j,k \in \{1 \dots 9\},\\
&\text{(b)} \quad \sum_{j=1}^{9} x_{i,j,k} && = 1 \quad \forall i,k \in \{1 \dots 9\},\\
&\text{(c)} \quad \sum_{j=3q-2}^{3q} \sum_{i=3p-2}^{3p} x_{i,j,k} && = 1 \quad \forall k \in \{1 \dots 9\}, \forall p,q \in \{1 \dots 3\},\\
&\text{(d)} \quad \sum_{k=1}^{9} x_{i,j,k} && = 1 \quad \forall i,j \in \{1 \dots 9\},
\end{aligned}$$

where F is the set containing all triples corresponding to the indices of the filled squares.

We can transform these constraints into penalty functions by subtracting 1 and squaring them. Then to obtain a solution to these constraints we minimise the sum of these penalties. This means that the solved Sudoku will always be the optimal solution so we don't need to use penalty weights.

Consider constraint (a) for some fixed j and k. Expanding the corresponding penalty gives us

$$\begin{aligned}
\left(\sum_{i=1}^{9} x_{i,j,k} - 1 \right)^2 &= \sum_{i=1}^{9} x_{i,j,k} \sum_{p=1}^{9} x_{p,j,k} - 2 \sum_{i=1}^{9} x_{i,j,k} + 1\\
&= \sum_{i=1}^{9} \sum_{p=1}^{9} x_{i,j,k} x_{p,j,k} - 2 \sum_{i=1}^{9} x_{i,j,k}^2 + 1\\
&= \begin{pmatrix} x_{1,j,k} & \dots & x_{9,j,k} \end{pmatrix} (J_9 - 2I) \begin{pmatrix} x_{1,j,k} \\ \vdots \\ x_{9,j,k} \end{pmatrix} + 1,
\end{aligned}$$

where J_n is the $n \times n$ matrix of ones.

This penalty can be expressed as $\mathbf{x}^T A_{j,k} \mathbf{x}$ where $A_{j,k}$ is the matrix of zeros except at the intersections of the columns and rows representing $x_{1,j,k}, \dots, x_{9,j,k}$. The entries at the intersections are 1 except on the main diagonal where they are -1.

Now we can define the matrix

$$A = \sum_{j=1}^{9} \sum_{k=1}^{9} A_{j,k},$$

which is the penalty matrix for constraint (a). Using this same idea we can construct the matrices

$$B = \sum_{i=1}^{9} \sum_{k=1}^{9} B_{i,k}, \quad D = \sum_{i=1}^{9} \sum_{j=1}^{9} D_{i,j},$$

where $B_{i,k}$ is the matrix of zeros except at the intersections of the columns and rows representing $x_{i,1,k}, \ldots, x_{i,9,k}$ and $D_{i,j}$ is the matrix of zeros except at the intersections of the columns and rows representing $x_{i,j,1}, \ldots, x_{i,j,9}$.
The entries at these intersections are 1 except on the main diagonal where the entries are -1.

The matrices B and D are penalty matrices corresponding to constraints (b) and (d) respectively.
Then we define $C_{p,q,k}$ as the matrix of zeros except at the intersections of the rows and columns representing $\{x_{i,j,k} : i \in \{3p-2, 3p\}, j \in \{3q-2, 3q\}\}$. Again, the entries at the intersections are 1 except on the main diagonal where the entries are -1. So we can construct the matrix

$$C = \sum_{p=1}^{3} \sum_{q=1}^{3} \sum_{k=1}^{9} C_{p,q,k},$$

which is the penalty matrix corresponding to constraint (c). Now we can define the matrix

$$U = A + B + C + D$$

which corresponds to the constraints (a), (b), (c) and (d)[1].

[1]This matrix is provided on the GitHub page under sudokuQ.mat

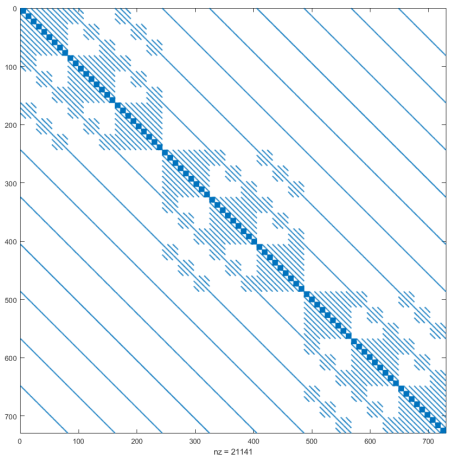


Figure 2: Sparsity pattern of the matrix U

Then we can take the squares which have already been filled and turn them into constraints. So the constraint $x_{i,j,k} = 1$ becomes

$$\begin{aligned}(x_{i,j,k} - 1)^2 &= x_{i,j,k}^2 - 2x_{i,j,k} + 1 \\ &= -x_{i,j,k}^2 + 1\end{aligned}$$

which can be interpreted as subtracting 1 from the (i, j, k)th element on the diagonal of U. Doing this for every square which has been filled creates the matrix Q.

Finally, solving the QUBO problem

$$\min_{\mathbf{x} \in B(n)} \mathbf{x}^T Q \mathbf{x}$$

will give us the $\mathbf{x}$ which is a solution of the original Sudoku problem.

Using this QUBO formulation the solution to the Sudoku is:

6	5	8	2	4	7	9	1	3
4	3	2	1	9	5	6	8	7
9	1	7	8	6	3	2	5	4
1	2	9	6	7	4	5	3	8
3	7	4	9	6	8	1	6	2
8	6	5	3	1	2	4	7	9
2	9	6	7	8	1	3	4	5
7	4	3	5	2	6	8	9	1
5	8	1	4	3	9	7	2	6

The code for this QUBO implementation is on the GitHub under `sudoku.m`.

Unfortunately, since Q is so large it is often not feasible to use this implementation to obtain solutions to Sudoku problems. A more efficient implementation of Sudoku is derived in [5].

In this implementation the matrix Q is 729×729. We could make this smaller by not including variables for squares which are filled.
Then assuming we are given 17 filled squares we can remove the variables representing these squares and reduce the number of variables by $17 \times 9 = 153$.
Since each filled square tells us the other squares on the same row, column or box have different values, 24 more variables are removed for each filled square.
If we consider that $\frac{17}{81}$ squares were already going to be removed, we can remove another $\frac{64}{81} \times 24 \times 17 \approx 320$.

So we should be able to reduce the number of variables to ≈ 260 which would make this more efficient.

3 Graph Isomorphism and Matching problems

3.1 The Graph Isomorphism problem

Two graphs G_A and G_B are isomorphic if and only if there exists a bijection between the vertices of the graphs such that every edge which connects vertices in G_A corresponds to an edge connecting vertices in G_B.
Essentially, an isomorphism means either graph can be redrawn and relabelled to be identical to the other.

So if the graphs have a different number of vertices or a different amount of edges, they cannot be isomorphic. You might think now that you can always find or rule out an isomorphism easily, but isomorphisms between graphs are not always obvious. For instance, can you tell if the graphs below are isomorphic?

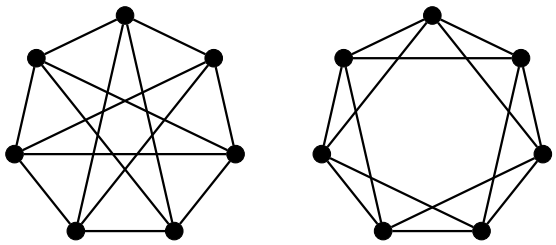


Figure 3: The graphs G_A (left) and G_B (right)

Now imagine trying to find an isomorphism between even larger graphs. Even-

tually, it is no longer possible to identify them by inspection and so we use algorithms to search for isomorphisms.

The graph isomorphism problem is the problem of determining whether any two graphs are isomorphic.

To interpret this as a QUBO problem we use some results from [3]. First,

Lemma 2 *[3, p. 6] Suppose we have graphs G_A and G_B with adjacency matrices A and B respectively. The doubly stochastic relaxation of the graph isomorphism problem can be formulated as*

$$c_D = \min_{X \in D(n)} ||XA - BX||_F^2,$$

where $D(n)$ is the set of doubly stochastic matrices and $||\cdot||_F$ denotes the Frobenius norm.

Lemma 3 *[3, p. 8] This optimisation problem can be written as*

$$\min_{\substack{\mathbf{x} \in B(n) \\ C\mathbf{x} = d}} \mathbf{x}^T H \mathbf{x},$$

where $\mathbf{x} = vec(X)$, $H = (A \otimes I_n - I_n \otimes B)^2$, and C, d are defined by (2).

Since we have shown in (3) that the constraint $C\mathbf{x} = d$ corresponds to the penalty $\mathbf{x}^T(C^TC - 2\text{diag}(C^Td))\mathbf{x}$, a QUBO formulation of this problem can be written as

$$\min_{x \in B(n)} \mathbf{x}^T Q \mathbf{x}, \tag{4}$$

where $Q = H + \lambda C^TC - 2\lambda\text{diag}(C^Td)$ and λ is the penalty weight.

Alternatively, we can use Lemma 4.15 which states:

Lemma 4 *[3, p. 13] The optimisation problem can be written as*

$$\min_{\substack{\mathbf{x} \in B(n) \\ C\mathbf{x} = d \\ H\mathbf{x} = 0}} -\mathbf{x}^T \mathbf{x},$$

where $\mathbf{x}, C, d$ and H are the same as in Lemma 3.

Since the constraint $H\mathbf{x} = 0$ corresponds to the penalty $H\mathbf{x} \cdot H\mathbf{x} = \mathbf{x}^T H^T H \mathbf{x}$ we can write a QUBO formulation of this problem as

$$\min_{x \geq 0} \mathbf{x}^T Q \mathbf{x} \tag{5}$$

where $Q = -I_n + \lambda_1 C^TC - 2\lambda_1\text{diag}(C^Td) + \lambda_2 H^T H$ with penalty weights λ_1 and λ_2.

When we compare both QUBO implementations for the problem we find that generally implementation (4) is faster.

3.1.1 Example: Graph Isomorphism problem

Consider the graphs G_A and G_B from Figure 3.

These graphs have the adjacency matrices

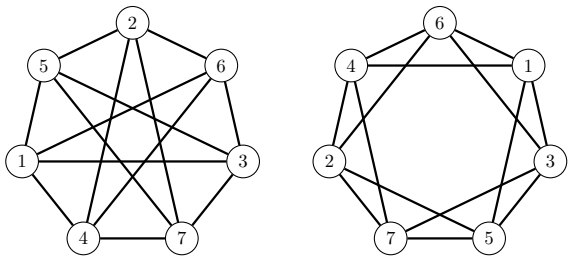
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \text{ and } B = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix},$$

when we enumerate their vertices clockwise. If we substitute A, B into (4), choose $\lambda = 10$, and solve the QUBO problem we get

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

which we find represents an isomorphism between G_A and G_B.

Using this we find that the labelling



is an isomorphism.

The code for this implementation is:

```
H = (kron(A, eye(n)) - kron(eye(n), B))^2;
Q = H + weight*(Ct*C - 2*diag(Ct*d));

```

```
x = solve(qubo(Q));
mat = reshape(x.BestX, [n,n]);
```

3.2 Graph Matching and Quadratic Assignment problem

3.2.1 Graph Matching

Given two weighted graphs G_A and G_B, the graph matching problem is the problem of finding the bijection between the vertices of G_A and G_B which minimises the edge difference.

Essentially, the graph matching problem is a generalisation of the graph isomorphism problem but instead of determining if the graphs are isomorphic we want to find the closest bijection between the graphs. The graph matching problem can be formulated as (2) where A, B are weighted adjacency matrices. This means that it is equivalent to (3) following the steps in [3].

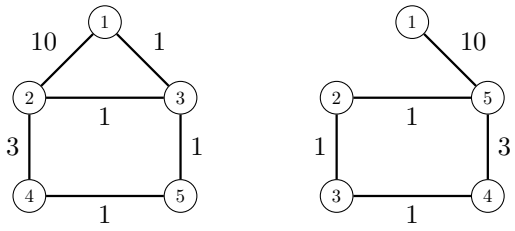
So, the QUBO formulation of the graph matching problem is

$$\min_{\mathbf{x}\in B(n)} \mathbf{x}^T Q \mathbf{x},$$

where $Q = (A \otimes I_n - I_n \otimes B)^2 + \lambda C^T C - 2\lambda \text{diag}(C^T d)$, λ is the penalty weight and C, d are defined by 2.

3.2.2 Example: Graph Matching

Consider the graphs G_L and G_R on the left and right respectively.



These graphs have the adjacency matrices

$$L = \begin{pmatrix} 0 & 10 & 1 & 0 & 0 \\ 10 & 0 & 1 & 3 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 3 \\ 10 & 1 & 0 & 3 & 0 \end{pmatrix}.$$

Using these matrices, we find the solution of the graph matching problem is the permutation matrix

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

which corresponds to the mapping from G_L to G_R which takes $1 \mapsto 1, 2 \mapsto 5, 3 \mapsto 2, 4 \mapsto 4$ and $5 \mapsto 3$.

3.2.3 Quadratic Assignment

The quadratic assignment problem can be stated as:

Suppose we are given a set of n facilities and n locations, our job is to allocate each facility to a location such that we minimise the total cost. Each pair of facilities has a required flow and each pair of locations has a distance. The cost function is the sum of the flows multiplied by the distance for each facility-location pair.

Many real-world problems such as optimising building layouts, hospital transportation and scheduling can be expressed as quadratic assignment problems.

We can obtain a QUBO formulation of this as follows:
First let F and D be the flow and distance matrices where entries F_{ij} are the required flow between facilities i and j and D_{ij} is the distance between locations l_i and l_j. Then the problem can be expressed as:

$$\min_{X \in \mathcal{P}(n)} \operatorname{tr}(FXD^T X^T).$$

Using [3, p. 3] since X is a permutation matrix we have

$$2\operatorname{tr}(FXD^T X^T) = ||F||_F^2 + ||D||_F^2 - ||XF - DX||_F^2.$$

Therefore minimising $\operatorname{tr}(FXD^T X^T)$ is equivalent to maximising $||XF - DX||_F^2$ which is the negation of (2) so the QUBO formulation of the quadratic assignment problem is the negation of (3), i.e.,

$$\min_{\mathbf{x} \in B(n)} \mathbf{x}^T Q \mathbf{x}$$

where $Q = -(A \otimes I_n - I_n \otimes B)^2 + \lambda C^T C - \lambda \operatorname{diag}(C^T d)$, λ is the penalty weight and C, d are defined by (2).

4 Quantum Annealing

One of the main reasons for constructing QUBO formulations of problems is that they are well-suited to being solved using quantum annealing. In this section we detail the connection between QUBO problems and quantum annealers.

4.1 Ising models

First we need to modify our QUBO problems slightly to obtain the Ising models which quantum annealers are designed to solve. Recall that the general form of a QUBO problem is

$$\min_{x \in B(n)} f(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x} + d.$$

An Ising problem is an optimisation problem where the objective function is a quadratic function of multiple variables and each variable is either -1 or 1 (which are referred to as spins). We can convert a QUBO problem to an Ising formulation by applying the linear mapping $s_i = 2x_i - 1$. The objective function values will then differ by

$$\mathbf{x}^T Q \mathbf{x} + d = \frac{1}{4}(\mathbf{s}^T + J_{1,n})Q(\mathbf{s} + J_{n,1}) + d$$

where $J_{i,j}$ is the $i \times j$ matrix of ones.

The objective function of the Ising formulation is referred to as the energy (so that the optimal solution corresponds to the lowest energy).

4.2 Quantum Annealers

This section is an abridged explanation of some of the ideas behind quantum annealing. For a more detailed explanation see [8] and [9].

The idea behind quantum annealing is as follows. First we create an Ising formulation of the target problem, and label the objective we are looking to minimise the final Hamiltonian.
Then we prepare a quantum system in its ground state of an easy to solve function known as the initial Hamiltonian. If we slowly interpolate between the initial and the final Hamiltonian, the Adiabatic theorem [9] ensures that the quantum system remains in the ground state. At the end, this ground state will correspond to the optimal solution of the target problem.

This process is analogous the the technique of annealing metal where a metal is heated above its recrystallization temperature and slowly cooled to increase strength and reduce stress.
A great explanation of the physical mechanism underlying quantum annealing is given in [7] by D-Wave Systems who are the leading providers of quantum

annealers.

4.3 Limitations

As shown in [8], there is some evidence that quantum annealers can provide a performance advantage for certain problems. But this advantage has not been conclusively demonstrated over the fastest classical algorithms.

The largest commercial quantum annealer is D-Wave System's 5000 qubit Advantage system [4]. This system allows 15 couplers per qubit which effectively means each column in the QUBO matrix must have less than 16 entries. This could mean that some QUBO formulations are infeasible to solve using the Advantage system.

Unfortunately the qubo function from the MATLAB Support Package for Quantum Computing is not directly compatible with any quantum computers.

5 Conclusion

In this project we showed how many combinatorial optimization problems can be naturally expressed as QUBO problems (with particular attention to the graph isomorphism problem). As research into QUBO problems and quantum annealers continues, we will hopefully develop formulations for more problems and overcome the limitations in section 4.3.

5.1 Acknowledgements

Thank you to my supervisor Dr Stefan Klus for his guidance and help throughout this project.

This project was funded by the Edinburgh Mathematical Society.

References

[1] Fahren Bukhari et al. "Formulation of Sudoku Puzzle Using Binary Integer Linear Programming and Its Implementation in Julia, Python, and Minizinc". In: *Jambura Journal of Mathematics* 4.2 (2022), pp. 323–331. ISSN: 2656-1344. DOI: 10.34312/jjom.v4i2.14194.

[2] Fred Glover, Gary Kochenberger, and Yu Du. *A Tutorial on Formulating and Using QUBO Models*. 2019. arXiv: 1811.11538 [cs.DS]. URL: https://arxiv.org/abs/1811.11538.

[3] Stefan Klus and Patrick Gelß. *Continuous optimization methods for the graph isomorphism problem*. 2023. arXiv: 2311.16912.

[4] Catherine McGeoch and Pau Farré. *Advantage Processor Overview*. dwavesys.com, 2022. URL: https://www.dwavesys.com/resources/white-paper/the-d-wave-advantage-system-an-overview/ (visited on 07/08/2024).

[5] Sascha Mücke. *A Simple QUBO Formulation of Sudoku*. 2024. arXiv: 2403.04816.

[6] Atanu Rajak et al. "Quantum annealing: an overview". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 381.2241 (Dec. 2022). ISSN: 1471-2962. DOI: 10.1098/rsta.2021.0417. URL: http://dx.doi.org/10.1098/rsta.2021.0417.

[7] D-Wave Systems. *What is Quantum Annealing?* URL: https://docs.dwavesys.com/docs/latest/c_gs_2.html (visited on 07/07/2024).

[8] Byron Tasseff et al. *On the Emerging Potential of Quantum Annealing Hardware for Combinatorial Optimization*. 2022. arXiv: 2210.04291 [math.OC]. URL: https://arxiv.org/abs/2210.04291.

[9] Sheir Yarkoni et al. "Quantum annealing for industry applications: introduction and review". In: *Reports on Progress in Physics* 85.10 (Sept. 2022), p. 104001. ISSN: 1361-6633. DOI: 10.1088/1361-6633/ac8c54. URL: http://dx.doi.org/10.1088/1361-6633/ac8c54.