QOSF Cohert-10(Optimization)

Haris Ansari

November 18, 2024

1 Formulation

The Bin Packing Problem (BPP) is a classic optimization problem. In this problem, we are given a set of items, each with a specific weight, and a set of bins with a fixed capacity. The objective is to pack all the items into the minimum number of bins without exceeding the capacity of any bin.

In more formal terms:

Given n items with weights $w_1, w_2, \ldots, w_n$. Each bin has a maximum capacity C. We aim to minimize the number of bins used to hold all items without exceeding C for any bin.

1.1 Integer Linear Programming (ILP) Formulation for Bin Packing Problem

The problem can be formulated as an ILP as follows:

1.1.1 Decision Variables

- x_{ij}: A binary variable that takes the value 1 if item i is placed in bin j, and 0 otherwise.

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is placed in bin } j \\ 0 & \text{otherwise} \end{cases}$$

- y_j: A binary variable that takes the value 1 if bin j is used, and 0 otherwise.

$$y_j = \begin{cases} 1 & \text{if bin } j \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

1.1.2 Objective Function

The objective is to minimize the number of bins used:

$$\text{Minimize } \sum_{j=1}^{m} y_j$$

where m is an upper bound on the number of bins (usually taken as the number of items, n, because in the worst case, each item could go in a separate bin).

1.1.3 Constraints

1. Each item must be assigned to exactly one bin:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, n\}$$

This ensures that every item is placed in exactly one bin.

2. The total weight in each bin cannot exceed its capacity:

$$\sum_{i=1}^{n} w_i x_{ij} \leq C y_j, \quad \forall j \in \{1, 2, \ldots, m\}$$

This constraint ensures that if a bin j is used ($y_j = 1$), the sum of the weights of items assigned to it does not exceed the bin capacity C.

3. Binary constraints on the variables:

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, 2, \ldots, n\}, \, j \in \{1, 2, \ldots, m\}$$

$$y_j \in \{0, 1\}, \quad \forall j \in \{1, 2, \ldots, m\}$$

1.1.4 Explanation of Each Term

- **Objective Function** $\sum_{j=1}^{m} y_j$: This term represents the total number of bins used. By minimizing this sum, we aim to use the fewest bins necessary to pack all items.
- **Constraint 1** $\sum_{j=1}^{m} x_{ij} = 1$: This ensures that each item i is assigned to exactly one bin j.
- **Constraint 2** $\sum_{i=1}^{n} w_i x_{ij} \leq C y_j$: For each bin j, this ensures that the sum of the weights of the items in the bin does not exceed the bin's capacity C. The variable y_j acts as an indicator: if $y_j = 0$, then no items can be assigned to bin j, and if $y_j = 1$, bin j can hold items but must not exceed the capacity C.
- **Binary Constraints** $x_{ij} \in \{0, 1\}$ and $y_j \in \{0, 1\}$: These enforce that x_{ij} and y_j are binary variables. x_{ij} takes a value of 1 if item i is placed in bin j, and 0 otherwise. Similarly, y_j is 1 if bin j is used and 0 otherwise.

1.2 Formulation of the Bin Packing Problem

Problem Statement:

- **Weights of Items:** $\texttt{weights} = [3, 2, 3, 4]$
- **Bin Capacity:** $C = \max(\texttt{weights}) + 2 = 6$
- **Objective:** Minimize the number of bins used.

Variables:

- **Item Assignment Variables:** $x_{ij} \in \{0, 1\}$ ($x_{ij} = 1$: Item i is assigned to bin j).
- **Bin Usage Variables:** $y_j \in \{0, 1\}$ ($y_j = 1$: Bin j is used).

Objective Function:

$$\text{Minimize: } y_0 + y_1$$

Constraints:

1. **Item Assignment:** Each item must be assigned to one bin:

$$x_{i0} + x_{i1} = 1 \quad \forall i \in \{0, 1, 2, 3\}$$

2. **Bin Capacity:** Total weight in each bin cannot exceed its capacity:

$$3x_{0j} + 2x_{1j} + 3x_{2j} + 4x_{3j} - 6y_j \leq 0 \quad \forall j \in \{0, 1\}$$

3. **Pre-assignment:** Item 0 is pre-assigned to bin 0:

$$x_{0,0} = 1$$

4. **Binary Variables:**

$$x_{ij} \in \{0,1\},\ y_j \in \{0,1\} \quad \forall i \in \{0,1,2,3\},\ j \in \{0,1\}$$

2 Brute Force Approach

2.1 STEP 1 : Select the minimum number of bins required

This method reduced the number of Binary variables significantly.

1. **Input**: Accepts item weights and bin capacity C.
2. **Initial Bin Count**: Initializes the bin count n to the number of items.
3. **Bin Reduction**: It checks if $w_s \leq n \times C$; if not, the previous count $n+1$ is returned as the minimum. If $w_s = n \times C$, the current n is returned.
4. **Loop Continuation**: Repeats the bin reduction process until the minimum feasible bin count is found.

To find the most efficient configuration, we will use the

3 Brute Force Algorithm for Bin Packing Problem

3.1 Algorithm Description

1. **Input and Parameters**:
   - Let `weights` represent the list of item weights to be packed.
   - Let C be the capacity of each bin, calculated as $C = \max(\texttt{weights}) + 2$.
2. **Filtering Valid Combinations**:
   - Enumerate all possible combinations of binary variables x_{ij}, where $x_{ij} = 1$ indicates that item i is placed in bin j, and $x_{ij} = 0$ indicates it is not.
   - Filter combinations of x_{ij} that satisfy the binary constraint:

     $$\sum_j x_{ij} = 1, \quad \forall i$$

     ensuring each item is placed in exactly one bin.
3. **Setting Initial Bin Count**:
   - Initialize with $Y = 1$, setting $y_1 = 1$ (indicating the use of the first bin) and all other $y_j = 0$.
4. **Checking Capacity Constraints**:
   - For $Y = 1$, check if any valid combination of x_{ij} satisfies the capacity constraint:

     $$\sum_i x_{ij} \times \texttt{weight}_i \leq C, \quad \forall j \leq Y$$

   - If no valid combination is found, increment Y by 1, i.e., set $y_1 = y_2 = 1$ and remaining $y_j = 0$, then repeat the process.
   - Continue increasing Y until a valid combination of x_{ij} is found that satisfies the capacity constraints.
5. **Output**: The solution matrix for the minimum number of bins Y, with each entry representing the value of x_{ij} for each item i and bin j.

3.2 Example

Input:

- Weights of items: `weights` $= [3, 2, 3, 4]$
- Bin capacity: $C = \max(\texttt{weights}) + 2 = 6$

Output:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Interpretation:

- Item 1 is placed in Bin 2, represented by $x_{12} = 1$.
- Item 2 is placed in Bin 1, represented by $x_{21} = 1$.
- Item 3 is placed in Bin 2, represented by $x_{32} = 1$.
- Item 4 is placed in Bin 1, represented by $x_{41} = 1$.

Thus, the minimum number of bins required is $Y = 2$, as shown by the configuration above.

4 ILP to QUBO Conversion

To solve this problem using quantum methods, the ILP formulation is transformed into a Quadratic Unconstrained Binary Optimization (QUBO) problem. The key steps involved are:

4.1 Objective Reformulation

The original objective of minimizing

$$\sum_{i=1}^{m} y_i$$

is retained, as it directly aligns with the goal of reducing the number of bins used.

To handle the capacity constraint in the QUBO formulation, slack variables and penalty terms are introduced. The penalty term

$$(Cy_i - \sum_{j=1}^{n} w_j x_{ij})^2$$

enforces that the weight of items assigned to a bin does not exceed its capacity when $y_i = 1$. The penalty ensures that any infeasible solution is penalized, steering the optimization towards feasible configurations.

4.2 Unconstraining the Assignment Constraint

The constraint

$$\sum_{i=1}^{m} x_{ij} = 1$$

ensures each item is assigned to exactly one bin. This is converted into a penalty term added to the objective:

$$P_{\text{assign}} = \beta \sum_{j=1}^{n} \left(1 - \sum_{i=1}^{m} x_{ij}\right)^2$$

where β is a large penalty weight.

4.3 Capacity Constraint Quadratic Penalty

The capacity constraint

$$\sum_{j=1}^{n} w_j x_{ij} \leq C y_i$$

is expressed using a quadratic penalty term:

$$P_{\text{capacity}} = \alpha \sum_{i=1}^{m} \left(C y_i - \sum_{j=1}^{n} w_j x_{ij}\right)^2$$

Here, the quadratic nature aligns with the QUBO framework, as it introduces interactions between x_{ij} and y_i.

4.4 Binary Variable Encoding

All variables (x_{ij} and y_i) are inherently binary, which fits directly into the QUBO representation.

4.5 Final QUBO Objective

The final QUBO objective combines the original objective and the penalty terms:

$$Q(x,y) = \sum_{i=1}^{m} y_i + \alpha \sum_{i=1}^{m} \left(C y_i - \sum_{j=1}^{n} w_j x_{ij} \right)^2 + \beta \sum_{j=1}^{n} \left(1 - \sum_{i=1}^{m} x_{ij} \right)^2$$

5 Quantum Annealing Approach

Initialize the Sampler:

- An embedding-aware sampler (`EmbeddingComposite`) is created, wrapping the `DWaveSampler`.
- This handles the mapping of logical qubits in the QUBO to physical qubits on the D-Wave hardware.

Run the Sampler:

- The sampler executes the QUBO model using `sampler.sample_qubo()` with 100 reads (`num_reads=100`).
- The sampler generates a set of samples, each representing a possible solution.

Extract the Best Solution:

- From the sampled solutions, the one with the lowest energy is identified as the best solution.
- The solution variables (`best_solution`) and the corresponding energy (`best_energy`) are extracted from the sample set.

5.1 Results:

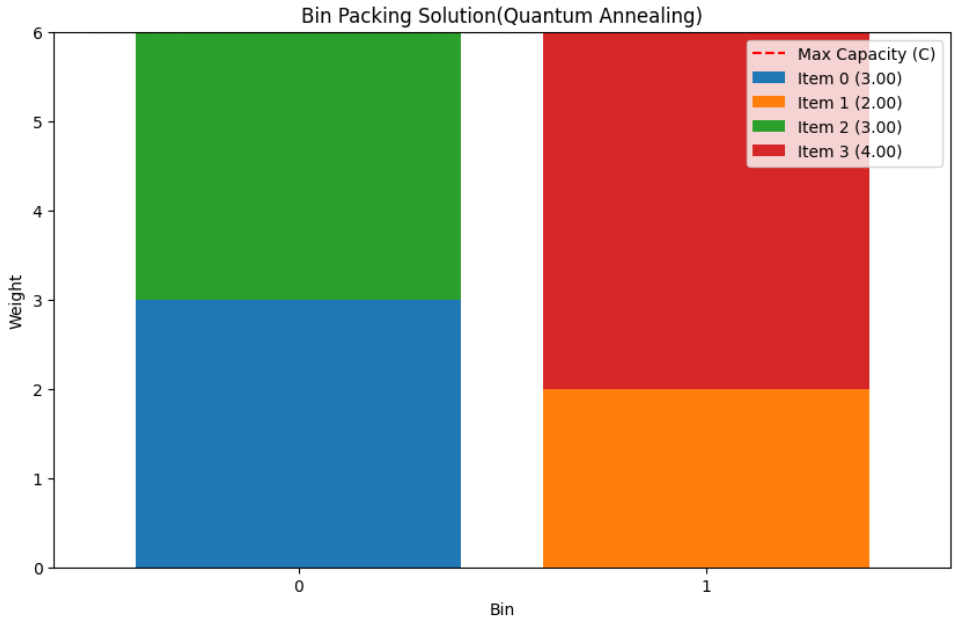

Figure 1: An example image.

5.2 Steps in the VQA Approach

The Variational Quantum Algorithm (VQA) solves optimization problems by iterating between quantum and classical computations. Below is a summary of the key steps in the VQA approach:

1. Problem Formulation:

The optimization problem is mapped to an Ising Hamiltonian, expressed as:

$$H = \sum_i h_i Z_i + \sum_{i<j} J_{ij} Z_i Z_j$$

where h_i and J_{ij} are problem-specific coefficients, and Z_i are Pauli-Z operators acting on qubits. The goal is to find the ground state of the Hamiltonian H, corresponding to the minimum energy configuration.

2. Parameter Initialization:

A quantum circuit is designed with parameterized gates, such as single-qubit rotations and entangling gates, to prepare a trial quantum state $|\psi(\vec{\theta})\rangle$. The parameters $\vec{\theta}$ are initialized randomly, and the state is prepared accordingly.

3. Energy Measurement:

The energy expectation value is measured on the quantum hardware for the current parameters $\vec{\theta}$. The energy expectation is given by:

$$\langle H \rangle = \langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle$$

This represents the energy of the trial quantum state with respect to the Hamiltonian H.

4. Classical Optimization:

A classical optimizer (e.g., gradient-based or gradient-free methods) is used to update the parameters $\vec{\theta}$ in order to minimize the energy expectation value $\langle H \rangle$. The optimizer seeks the optimal set of parameters that reduce the energy.

5. Iterative Refinement:

Steps 3 and 4 are repeated iteratively until the energy converges to a minimum, yielding the optimized parameters $\vec{\theta}^*$. The process continues until the algorithm reaches a solution that is sufficiently close to the ground state.

6. Solution Interpretation:

Once the optimal parameters $\vec{\theta}^*$ are found, the final quantum state $|\psi(\vec{\theta}^*)\rangle$ is measured in the computational basis. This measurement provides the solution to the Ising problem, which corresponds to the minimum energy configuration or the best solution to the optimization problem.

6 Mapping Eigenstates to QUBO Variables

Below, we describe the representation of different variables used in such models for bin packing or similar optimization problems.

6.1 1. Identify the Ordering of Variables

In the QUBO problem statement, the binary variables are listed in a specific order, which corresponds to how they are mapped to the bits in the eigenstate. The list of binary variables may look like the following:

$$x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}, x_{2,0}, x_{2,1}, x_{3,0}, x_{3,1}, y_0, y_1, \text{capacity\_bin\_0\_@int\_slack\_@0}, \ldots, \text{capacity\_bin\_1\_@int\_slack\_@2}$$

The variables in the QUBO model are directly mapped to the bits of the eigenstate, which represents a potential solution. Each binary variable corresponds to a specific bit in the binary string.

6.2 2. Map the Eigenstate to Variables

Once we have an eigenstate, such as:

$$\text{eigenstate} = [1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]$$

We can map these bits back to the variables as follows:

Binary Variable	**Eigenstate (0/1)**
$x_{0,0}$	1
$x_{0,1}$	0
$x_{1,0}$	0
$x_{1,1}$	1
$x_{2,0}$	1
$x_{2,1}$	0
$x_{3,0}$	0
$x_{3,1}$	1
y_0	1
y_1	1
capacity_bin_0_@int_slack_@0	0
capacity_bin_0_@int_slack_@1	0
capacity_bin_1_@int_slack_@0	0
capacity_bin_1_@int_slack_@1	0
capacity_bin_2_@int_slack_@0	0
capacity_bin_2_@int_slack_@1	0

Table 1: Mapping of Binary Variables to Eigenstate Values

6.3 Results:

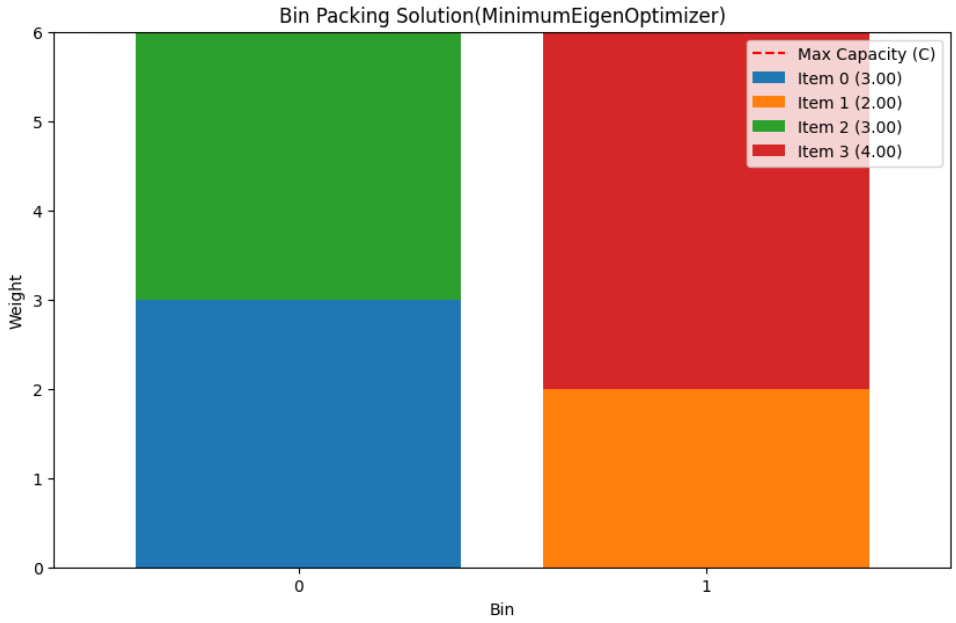

Figure 2: An example image.