Modeling the Line Balancing Problem as a QUBO Model and Solving It by Quantum Annealing

Luis A. Moncayo–Martínez[1,2] and Naihui He[2]

[1] Department of Industrial Engineering and Operations, Instituto Tecnológico Autónomo de México (ITAM), Mexico City, Mexico
luis.moncayo@itam.mx; l.a.moncayo-martinez@exeter.ac.uk
[2] Department of Engineering, Faculty of Environment, Science and Economy, University of Exeter, Exeter, UK

Abstract. The line balancing problem is a critical challenge in Industry 4.0 due to its significant impact on the efficiency, flexibility, and productivity of modern manufacturing systems. While this problem is classified as NP-hard, various optimization tools have been developed to obtain feasible solutions. In this work, the line balancing problem (SALB-1 and SALB-2) is addressed using Quantum Annealing heuristics, implemented on D-Wave's Leap technology. The SALB-1 and SALB-2 problems are first formulated as Quadratic Unconstrained Binary Optimization (QUBO) models. Then, using the Python package `dwave.system`, SALB-1 is solved with the `Binary Quadratic Model` class, and SALB-2 is tackled using the `Constrained Quadratic Model` class. To evaluate the implementation, three problem instances are solved, achieving the optimal solution in five out of six experiments. The code is available for download, allowing readers to replicate the results or solve larger instances using accounts with unrestricted access to D-Wave's QPUs and hybrid solvers.

Keywords: line balancing problem, D-Wave, Quantum Annealing

1 Introduction

Industry 4.0 changes conventional manufacturing systems into intelligent, connected, and highly adaptable ecosystems, facilitating remarkable efficiency, innovation, and competitiveness improvements. Therefore, manufacturers can deliver the final products to the customer at the right time and in the right quantity[1]. One important element in any manufacturing system is the assembly line, which is well-known as a challenging optimisation problem that directly impacts productivity, costs, and the ability to meet customer demands.; it enables manufacturers to deliver the final product on time, provided that the workloads are distributed along the line.

The line balancing problem is a critical challenge in production and assembly systems that involves efficiently allocating tasks to workstations along a

production line to optimise performance. The aim is to ensure that tasks are distributed in a way that minimises idle time, balances workloads, and meets desired production rates. Even though balancing a production line in Industry 4.0 is important, finding a solution to the problem is not straightforward. The line balancing problem is NP-hard [2] because of its binary combinatorial nature; thus, there is no known polynomial-time algorithm to find an optimal solution for all instances. Even for small instances, finding the optimal solution can be computationally expensive.

In recent years, the Quantum Annealing (QA) heuristic has been a promising approach to solving binary combinatorial problems, particularly those that are computationally infeasible for classical methods[3]. While it is not a universal quantum computing solution, its focus on optimisation makes it a powerful tool in industries such as logistics, finance, and machine learning, especially as hardware capabilities improve[4].

The optimisation problem is expressed as a mathematical function called the Ising model or the Quadratic Unconstrained Binary Optimisation (QUBO) problem. The goal is to find the configuration of binary variables (spins in the Ising model) that minimises the system's energy.

One of the key strengths of the QA heuristic is its ability to efficiently solve QUBO problems, which appear in many real-world optimisation tasks.

QA has been applied in transportation to optimise traffic flow and improve scheduling. For instance, research has demonstrated how quantum annealers can address complex logistics[5] and job shop scheduling problems[6] by finding optimal routes or schedules under tight constraints. In finance, quantum annealing is utilised for portfolio optimisation, assisting in selecting the ideal mix of assets to maximise returns while minimising risk[7]. This application is especially valuable because portfolio optimisation typically involves solving large-scale, complex problems that can be computationally intensive for classical systems[8]. Furthermore, QA has been explored in quantum chemistry and biology, particularly for molecular simulations and solving problems related to protein folding[9]. The ability of quantum annealers to efficiently explore large solution spaces makes them an attractive tool for simulating molecular systems that are computationally expensive using classical methods. In machine learning, quantum annealing has been leveraged for tasks like clustering, classification, and feature selection, where optimisation of models is crucial[10]. Another application includes nurse scheduling, simulation in physics, finite-element design, and material design, see [4].

Overall, while quantum annealing remains in the experimental phase for many applications, its potential to provide faster and more efficient solutions to optimisation problems in industries makes it a promising area of research and development. Using QA heuristics to solve the line balancing problem is a promising approach due to the problem's combinatorial nature and complex optimisation landscape. Therefore, this work aims to model the line balancing

problem as a QUBO model and implement it in D-Wave Leap to find optimal and/or near-optimal solutions for complex configurations. The hypothesis suggests that QA can reduce the number of workstations by leveraging quantum parallelism to explore various configurations simultaneously. This approach optimises workstation utilisation while adhering to the task's precedence requirements.

Our initial approach involves modelling the Simple Assembly Line Balancing (SALB) problem. This is done in two cases: when the cycle time is known (SALB-1) and when the number of workstations is known (SALB-2), both implemented as a QUBO model. After that, those models are implemented in D-Wave Leap using the class `BinaryQuadraticModel` for implementing the SALB-1 and the `ConstrainedQuadraticModel` for the SALB-2 problem from the Python package `dwave.system`. Three instances were implemented using a free developer access account that limits access to D-Wave QPUs and hybrid solvers.

According to recent literature-review papers, there are no works that report the application of the QA heuristic to solve the SALB-1 and SALB-2 problems [4,11,12,13,14]; thus, this is the first attempt to test the QA heuristic in solving the SALB problem.

2 Material and Methods

The SALB problem is modelled as a graph $G = \{V, E\}$, in which the set of vertices $V = \{1, \ldots, i, \ldots, j, \ldots, N\}$ represents the tasks that must be completed to assemble a final product. The set of edges $E = \{(1,2), (2,i) \ldots, (i,j)\}$ represents the precedence constraints; e.g., (i,j) means that task i must be finished before begining with task j. Moreover, there are K workstations $k = 1, 2, \ldots, K$ in which every task i must be assigned. The binary decision variables, defined in Eq. 1, are x_{ik} and y_k. The variable x_{ik} equals 1 if task i is assigned to the workstation k; while y_k represents whether workstation k is open.

$$x_{ik} = \begin{cases} 1, \text{ if } i \in k. \\ 0, \text{ otherwise} \end{cases}; \qquad y_k = \begin{cases} 1, \text{ if workstation } k \text{ is open.} \\ 0, \text{ otherwise} \end{cases} \tag{1}$$

Finally, every task i has a processing time t_i, and the cycle time C refers to the time allowed for completing a set of tasks assigned to a workstation. In practice, it is computed by dividing the available production time per period by the customer's demand per period.

In the SALB-1 problem, the objective is to minimise the number of workstations K, given that the cycle time C is known. In this work, the minimum number of workstations is computed by $K = \lceil \sum_{i=1}^{N} t_i / C \rceil$. Therefore, the optimisation problem involves assigning tasks to the K open stations without exceeding the cycle time C.

Eq. 2a is the objective function that minimises the number of open stations. Equation 2b guarantees that the sum of the assigned tasks' times does

not exceed the desired cycle time. Eq. 2c assures that every task is assigned to a single workstation. Eq. 2d assures that the predecessor relationships are guaranteed. For example, suppose the precedence relationship (i, j), and there are three workstations ($K = 3$); then $1x_{i1} + 2x_{i2} + 3x_{i3} \leq 1x_{j1} + 2x_{j2} + 3x_{j3}$. If task j is assigned to station 2, $2x_{i2} \leq 2x_{j2} + 3x_{j3}$. In other words, if task i is placed in workstation 2, task j can only be placed in workstation 2 or 3. Finally, Eq. 2e explicitly defines the domain of the decision variables.

$$\text{minimize} \quad \sum_{k=1}^{K} y_k \tag{2a}$$

$$\text{subject to} \quad \sum_{i=1}^{N} t_i x_{ik} \leq C y_k, \qquad k = 1, 2, \ldots, K, \tag{2b}$$

$$\sum_{k=1}^{K} x_{ik} = 1, \qquad i = 1, 2, \ldots, N, \tag{2c}$$

$$\sum_{k=1}^{K} k \times x_{ik} \leq \sum_{k=1}^{K} k \times x_{jk}, \qquad \forall\ (i, j) \mid i \prec j, \tag{2d}$$

$$x_{ik} \in \{0, 1\}\ \forall\ i, k,\ \ y_k \in (0, 1)\ \forall\ k \tag{2e}$$

The SALB-2 problem aims to minimise the cycle time C given that K workstations are open. Mathematically, there are two differences from the SALB-1 problem. First, the cycle time is a continuous variable that must be minimised; thus, the objective function changes as shown in Eq. 3a, and its domain is defined in constraint Eq. 3e. Second, the binary decision variable y_k is unnecessary, as the number of open workstations is known.

$$\text{minimize} \quad C \tag{3a}$$

$$\text{subject to} \quad \sum_{i=1}^{N} t_i x_{ik} \leq C, \qquad k = 1, 2, \ldots, K, \tag{3b}$$

$$\sum_{k=1}^{K} x_{ik} = 1, \qquad i = 1, 2, \ldots, N, \tag{3c}$$

$$\sum_{k=1}^{K} k \times x_{ik} \leq \sum_{k=1}^{K} k \times x_{jk}, \qquad \forall\ (i, j) \mid i \prec j, \tag{3d}$$

$$x_{ik} \in \{0, 1\}\ \forall\ i, k,\ \ C \geq 0 \tag{3e}$$

In the two versions of the SALB problem, the solution involves assigning tasks to the workstations. Thus, the time content c_k of each workstation is computed as the sum of the assigned tasks' processing times, and the efficiency of the

workstation E_k can be computed according to Equation 4. The efficiency is less than or equal to 1 in any feasible solution.

$$E_k = \frac{c_k}{C} = \frac{\sum_{i \in k} t_i}{C} \leq 1 \quad \forall\, k \tag{4}$$

2.1 The Line Balancing Problem as QUBO Model

A general quadratic *constrained* binary optimisation model is defined as minimise $\mathbf{cx} + \mathbf{x}^T\mathbf{Dx}$ subject to $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{x} \in (0, 1)$, where $\mathbf{c}$ is the row vector of objective function coefficients, $\mathbf{D}$ is the matrix defining interaction terms[4] between the column vector of decision variables $\mathbf{x}$. It is the assumed symmetric and negative definite, i.e., the objective function is strictly concave. Note that the objective function has a linear term $\mathbf{cx}$ and a quadratic term $\mathbf{x}^T\mathbf{Dx}$, which captures interactions between pairs of binary variables. $\mathbf{A}$ is the constraint matrix, with rows representing the linear inequalities for the set of constraints, and $\mathbf{b}$ is the vector representing the bounds on the resources available for each constraint.

To transform the constrained model into a QUBO problem, the equivalent quadratic representation of the constraints $\mathbf{Ax} \geq \mathbf{b}$ must be defined. Consider the constraint in row r with n variables, such that $\sum_n a_{rn}x_n \geq b_r$. To convert this to equality, a surplus variable $e_r \geq 0$ is introduced, resulting in the equation $\sum_n a_{rn}x_n - e_r = b_r$ (in case of a $\leq$ constraint a slack variable s is added). The binary equivalent of the continuous surplus variable e_r must hold for all binary variable values x_n, satisfying the original inequality by selecting appropriate values for the surplus variables. Specifically, we have $e = \sum_{p=0}^{l} 2^p e_p$, where l is the smallest integer such that finite upper bound u is $u \leq 2^{l+1} - 1$, and $e_p \in \{0, 1\}$[15]; e.g., consider $x_1 + x_2 + x_3 \leq 2$, adding the slack variable $x_1 + x_2 + x_3 + s = 2$. The slack variable has an upper bound $s \leq 2 = u$ when the values of the variables is zero; thus, when $l = 0$ then $2 \nleq 2^{0+1} - 1$, when $l = 1$ then $2 \nleq 2^{1+1} - 1$; thus $e = \sum_{p=0}^{1} 2^p s_p = s + 2s$. The resulting quadratic constraint is $\lambda_r \left(\sum_n a_{rn}x_n - e_r - b_r\right)^2 = \lambda_r \left(\sum_n a_{rn}x_n - \sum_{p=0}^{l} 2^p e_p - b_r\right)^2$, where λ_r is a positive constant for a minimisation problem and negative for a maximisation one. In D-Wave implementation λ_r are the lagrange multipliers.

The SALB-1 problem, as defined in Eqs. 2a-2e, includes two constraints of the type $\leq$ and one equality constraint. The constraint in Eq. 3b, which addresses the time content of workstations k, requires an additional surplus continuous variable, $s = \sum_{p=0}^{l} 2^p s_p$. Similarly, the slack variable $s' = \sum_{q=0}^{m} 2^q s'_q$ is introduced in Eq. 2d to satisfy the precedence constraints. Those binary variables are store in the following vectors $\mathbf{s} = \{s_0, s_1 \ldots\}$ and $\mathbf{s'} = \{s'_0, s'_1 \ldots\}$.

$$\min_{\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{s'}\in\{0,1\}} \quad \sum_{k=1}^{K} y_k + \lambda_1 \left(\sum_{i=1}^{N} t_i x_{ik} + \sum_{p=0}^{l} 2^p s_p - C y_k \right)^2 + \lambda_2 \left(\sum_{k=1}^{K} x_{ik} - 1 \right)^2 + \\ + \lambda_3 \left(\sum_{k=1}^{K} k \times x_{ik} - \sum_{k=1}^{K} k \times x_{jk} + \sum_{q=0}^{m} 2^q s'_q \right)^2 \tag{5}$$

Once the quadratic representation is computed using Eq. 5, the SALB-1 problem is modelled as an unconstrained problem of the form $\mathbf{cx} + \mathbf{x}^T\mathbf{Dx}$, $\mathbf{x} \in (0, 1)$. The whole model is presented in Eq. 6, where the elements d_{ab} of the matrix $\mathbf{D}$ is the relation among variable a and b, which can be any y_k, x_{ik}, or the added slack variables s_p in constraints Eq. 3b and Eq. 2d. z stands for a constant value.

$$\min_{\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{s'}\in\{0,1\}} (1,0,0,0,0,0) \begin{pmatrix} y_k \\ \cdots \\ x_{ik} \\ \cdots \\ s_p \\ \cdots \end{pmatrix} + \\ + \begin{pmatrix} y_k \\ \cdots \\ x_{ik} \\ \cdots \\ s_p \\ \cdots \end{pmatrix}^T \begin{pmatrix} d_{1,1} & \cdots & d_{1,K+1} & \cdots & d_{1,NK+1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & & \cdots \\ d_{K+1,1} & \cdots & d_{K+1,K+1} & \cdots & d_{K+1,NK+1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ d_{NK+1,1} & \cdots & d_{NK+1,1K+1} & \cdots & d_{NK+1,NK+1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} y_k \\ \cdots \\ x_{ik} \\ \cdots \\ s_p \\ \cdots \end{pmatrix} + z \tag{6}$$

The QUBO model that is solved in the D-Wave annealers is of the form $\min_{\mathbf{x}\in\{0,1\}} \mathbf{x}^T Q \mathbf{x}$, where $\mathbf{Q}$ is an upper–triangular–form or symmetric square matrix of constants[12]. An alternative representation[16] is shown in Eq. 7.

$$\min_{\mathbf{x}\in\{0,1\}} = \sum_a c_a x_a + \sum_a \sum_{b>a} q_{ab} x_a x_b + z \tag{7}$$

The c_a coefficient is place in the matrix $\mathbf{Q}$ in position (a, a) and the coefficients q_{ab} (such as $q_{ab}x_a x_b$) in position (a, b), and z is a constant. In symmetric form, every element q_{ab} is computed using Eq. 8. For example, if variable $a = y_k$ and variable $b = x_{ik}$, there is a q_{ab} given $q_{ab} \times y_k \times x_{ik}$. Moreover, if $a = y_k$, there is a terms $q_{aa} \times y_k$.

$$q_{ab} = \begin{cases} \frac{1}{2}\left(d_{ab} + d_{ba}\right), & \text{if } \ b \neq a \\ \\ c_a + d_{aa} & \text{otherwise} \end{cases} \tag{8}$$

The QUBO model of the SALB-1 problem to be solved by D-Wave annealers is in Eq. 9.

$$\text{QUBO}: \min_{\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{s}' \in \{0,1\}} [\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{s}']^T Q [\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{s}'] =$$
$$= \begin{pmatrix} y_k \\ \cdots \\ x_{ik} \\ \cdots \\ s_p \\ \cdots \end{pmatrix}^T \begin{pmatrix} q_{1,1} & \cdots & q_{1,K+1} & \cdots & q_{1,NK+1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & & \cdots \\ q_{K+1,1} & \cdots & q_{K+1,K+1} & \cdots & q_{K+1,NK+1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ q_{NK+1,1} & \cdots & q_{NK+1,1K+1} & \cdots & q_{NK+1,NK+1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} y_k \\ \cdots \\ x_{ik} \\ \cdots \\ s_p \\ \cdots \end{pmatrix} + z \tag{9}$$

To compute the QUBO model for the SALB-2 problem, the process must be repeated with Eq. 5, Eq. 6, Eq. 8, and Eq. 9.

2.2 The Quantum Annealing Algorithm

QA heuristic is inspired by simulated annealing, a classical optimisation technique that uses thermal fluctuations to explore solution spaces. In QA, instead of thermal energy, it uses quantum fluctuations to escape local minima and converge to the global minimum. Here's how it works:

1. *Problem Mapping*: The optimisation problem is expressed as a mathematical function called the QUBO problem as the one in Eq. 9.
2. *Initialization*: The system starts in a superposition of all possible states, with equal probability, represented by a quantum wavefunction. A simple Hamiltonian is applied initially, corresponding to a known ground state.
3. *Adiabatic Evolution*: The system evolves over time by gradually transforming the Hamiltonian from the simple initial state to a final one representing the problem's cost function. The adiabatic theorem ensures that if the evolution is slow enough and the system remains in its ground state, it will end up in the ground state of the final Hamiltonian, which corresponds to the optimal solution.
4. *Quantum Tunneling*: Quantum tunnelling allows the system to explore the solution space by "tunnelling" through energy barriers, avoiding local minima that could trap classical methods.
5. *Measurement*: At the end of the annealing process, the quantum state is measured, collapsing it to a classical state representing the optimal or near-optimal solution.

2.3 Computational Implementation

To solve the SALB-1 and 2 problems, the D-Wave package called `dwave.system` is used in this implementation. In the case of the SALB-1 problem, the `Binary Quadratic Model('BINARY')` is called, while to solve the SALB-2, the `Constrained Quadratic Model()` is used, as shown in Listening 1.

Listing 1.1: Package used in the implementation.

```
from dimod import ConstrainedQuadraticModel,
    BinaryQuadraticModel, Binary, Real, quicksum
from dwave.system import LeapHybridCQMSampler, DWaveSampler,
    EmbeddingComposite
```

Listing 1.2 shows the complete function for implementing the SALB-1 problem. The *dwave.system* package requires the model to be formatted as shown in Eqs. 2a-2e and to input the value of the Lagrange multipliers λ_1, λ_2 and λ_3 of the model in Eq. 5. The function requires the cycle time (C) (known in SALB-1).

The number of lines in the following description refers to Listing 2. In line 2, the number of minimum workstations is computed. In this work, The aim is to assign the tasks to the minimum number of workstations K given that their time content (see Eq. 4) is less than or equal to the known cycle time.

Listing 1.2: Implementation SALB-1 Proble in Eq 2a-2e.

```
def quantum_salb_1(self,cycle_time):
        workstations = math.ceil(sum(self.times)/cycle_time)

        y_list = [f"y{i}" for i in range(1,workstations+1)]
        x = []
        for t in self.tasks:
            for w in range(1,workstations+1):
                x.append(f't{t}_w{w}')

        bqm = BinaryQuadraticModel('BINARY')

        #create variables - Eq. 2e
        for r in range(len(x)):
            bqm.add_variable(x[r])
        for y in y_list:
            bqm.add_variable(v=y,bias=1)

        # Objective function - Eq. 2a
        c0 = [(y_list[u],1) for u in range(len(y_list))]
        bqm.add_linear_equality_constraint(terms = c0,
            lagrange_multiplier =1000, constant = -
            workstations)

        #workstation constraints - Eq. 2b
        for q in range(workstations):
            work = []
            increment = q
            while len(work) < len(self.tasks):
                work.append(x[increment])
                increment+=workstations
            c1 = [(work[a],self.times[a]) for a in range(len(
                self.tasks))]
```

```
30              bqm.add_linear_inequality_constraint(c1+[(y_list[
                    q],-cycle_time)], constant = 0,ub=0,lb=-20,
                    lagrange_multiplier = 1000, label = 'c1_time_
                    ')
31
32          #task assigned to one station - Eq. 2c
33          for w in range(len(self.tasks)):
34              e = workstations*w
35              task_workstation = x[e:e+workstations]
36              c2 = [(task_workstation[r],1) for r in range(
                    workstations)]
37              bqm.add_linear_equality_constraint(terms = c2,
                    lagrange_multiplier = 2000, constant = -1)
38
39           #precedence constraints - Eq. 2d
40           for p in self.precedence:
41              first_task = (p[0]-1)*workstations
42              second_task = (p[1]-1)*workstations
43              to_task = x[first_task:first_task+workstations]
44              from_task = x[second_task:second_task+
                    workstations]
45              c3_to = [(to_task[a],a+1) for a in range(
                    workstations)]
46              c3_from = [(from_task[a],-(a+1)) for a in range(
                    workstations)]
47              c3 = c3_to+c3_from
48              bqm.add_linear_inequality_constraint(c3,constant
                    =0,ub=0,lb=-workstations+1,
                    lagrange_multiplier = 10000, label = p)
49            #compute the solution
50            # Create a D-Wave sampler
51            sampler = DWaveSampler()
52            # Use EmbeddingComposite to handle embedding
                  automatically
53            embedding_sampler = EmbeddingComposite(sampler)
54            # Sample your problem
55            response = embedding_sampler.sample(bqm, num_reads
                  =3600)
```

On line 10, the binary quadratic model is created with all variables defined as *BINARY*. The variables are initialised on lines 13 to 16, as described in Eq. 2e. The objective function, represented by Eq. 2a, is implemented on lines 19 to 20, assuming the number of binary variables y_k is K calculated on line 2.

The constraint ensuring that the time content of each workstation does not exceed the cycle time (Eq. 2b) is implemented on lines 23 to 30. Similar to the objective function, the implementation uses the `add linear inequality constraint` function. The constraint in Eq. 2c, which enforces the assignment of each task to only one workstation, is implemented using the `add linear equality constraint` function. Lastly, the precedence constraints (Eq. 2d) are

implemented on lines 40 to 48, using the same function as in the objective function and the first constraint. On lines 51 to 55, the model is sent to the D-Wave hardware using a free developer access account. This account imposes certain limitations on access to D-Wave QPUs and hybrid solvers, including a maximum of 3600 reads (`num_reads=3600`) to obtain a solution from the hardware.

Listing 1.3 presents the implementation of the SALB-2 problem (Eqs. 3a-3e) using the `Constrained Quadratic Model` class. (From this point onward, line numbers refer to Listing 1.3.) In this model, the number of workstations K is predefined, and the objective is to minimise the cycle time C. Consequently, two types of variables are introduced: binary variables x_{ik} (line 12) and a real variable for the cycle time (line 13), as specified in the constraint in Eq. 3e.

The objective function (Eq. 3a) is implemented on line 16, while the constraints in Eqs. 3b to 3d are added using the `add constraint` function. In this implementation, it is not necessary to explicitly define the Lagrange multipliers, and the results are generated using the `Leap Hybrid CQM Sampler`.

Listing 1.3: Implementation SALB-2 Proble in Eq 3a-3e.

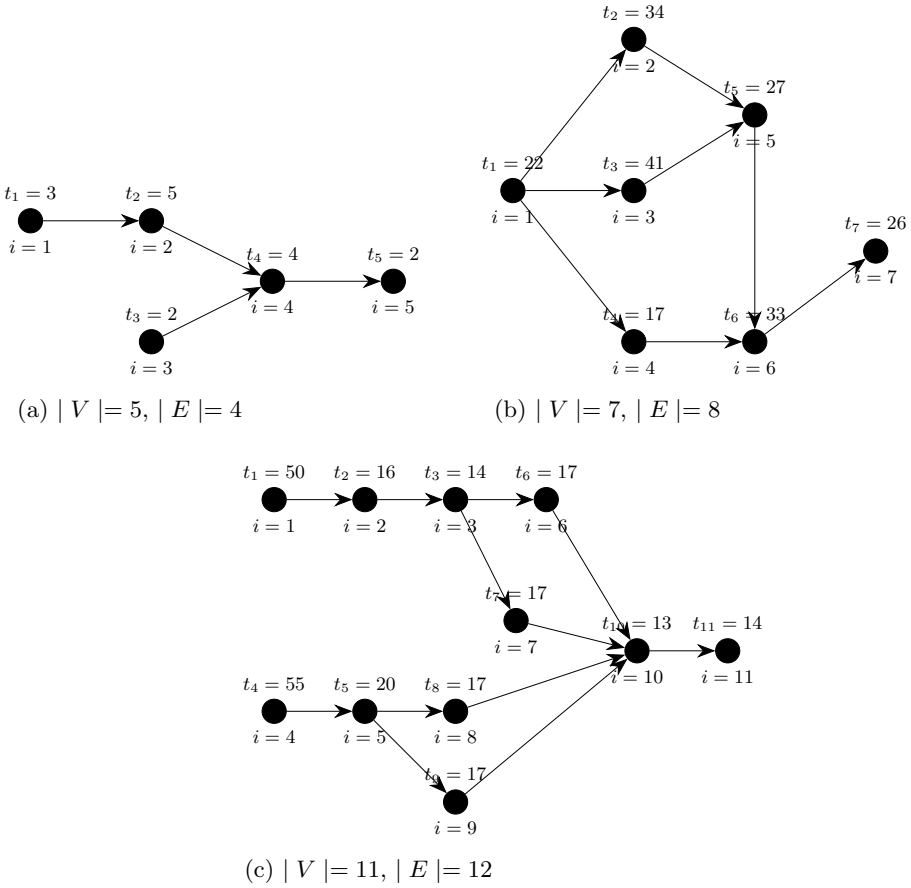
```
1  def quantum_salb_2(self,num_stations):
2          self.num_stations = num_stations
3          workstations = list(range(1,self.num_stations+1))
4
5          cqm = ConstrainedQuadraticModel()
6
7          # Define the variables
8          name = []
9          for t in self.tasks:
10             for w in workstations:
11                 name.append(f't{t}_w{w}')
12         x = [Binary(label=i) for i in name]
13         cycle_time = Real(label="cycle_time", upper_bound=sum
               (self.times))
14
15         # Set the objective function
16         cqm.set_objective(cycle_time)
17
18         # workstation constraints
19         for i in range(len(workstations)):
20             work = []
21             increment = i
22             while len(work) < len(self.tasks):
23                 work.append(x[increment])
24                 increment+=len(workstations)
25             cqm.add_constraint(quicksum(self.times[j]*work[j]
                   for j in range(len(self.tasks))) -
                  cycle_time <= 0, label='w'+str(i+1))
26
27         # tasks constrains
```

```
        for i in range(len(self.tasks)):
            j = len(workstations)*i
            task_to_ = x[j:j+len(workstations)]
            cqm.add_constraint(quicksum(task_to_[i] for i in
                range(len(task_to_))) == 1, label='t'+str(i
                +1))

        # precedences constraints
        for p in self.precedence:
            first_task = (p[0]-1)*len(workstations)
            second_task = (p[1]-1)*len(workstations)
            to_task = x[first_task:first_task+len(
                workstations)]
            from_task = x[second_task:second_task+len(
                workstations)]
            cqm.add_constraint(quicksum((i+1)*to_task[i] for
                i in range(len(workstations)))-quicksum((j+1)
                *from_task[j] for j in range(len(workstations
                ))) <= 0, label=p)

        # Send the problem
        cqm_sampler = LeapHybridCQMSampler()
        sampleset = cqm_sampler.sample_cqm(cqm, label=self.
            model_name)
```

3 Results

Figure 1 presents three simple instances used to test the D-Wave implementation. For example, the first instance consists of five tasks, $V = \{1, 2, 3, 4, 5\}$, and four precedence relationships, $E = \{(1, 2), (2, 4), (2, 4), (4, 5)\}$. The processing time for each task is displayed above its corresponding node. For each instance, the SALB-1 and SALB-2 problems are solved. In Table 1 and Table 2 are shown only some solutions. The reader can download the source code from the following Github repository to reproduce the results or taste with bigger instances with another type of account.

Table 1 presents the results of solving the SALB-1 problem using the implementation shown in Listing 1.2. In this problem, the cycle time is fixed. For the instance depicted in Figure 1a, the cycle time is set to $C = 7$. Solution 1 is a feasible one, where tasks $i = 1, 3$ are assigned to workstation $k = 1$, task 2 is assigned to workstation $k = 2$, and tasks $i = 4, 5$ are assigned to the last workstation. The minimum number of workstations required is three, $K = 3$. The time content for each workstation is $c_1 = 5$, $c_2 = 5$, and $c_3 = 6$. Thus, the efficiency (as given by Eq. 4) is 71% for workstations 1 and 2, and 86% for the third one. Solutions 1 and 2 for this instance are infeasible, as the cycle time exceeds $C = 7$. However, the precedence constraints are satisfied; e.g., for solution



(a) $| V |= 5, | E |= 4$ (b) $| V |= 7, | E |= 8$

(c) $| V |= 11, | E |= 12$

Fig. 1: Simple instances to test the D-Wave implementation

2, the resulting cycle time is 8 instead of the required of 7; thus, the efficiency (E_k) computed by Eq. 4 is greater than 1.

The results for the instance shown in Figure 1b are similar to the first one. In this case, the known cycle time is $C = 73$, and the implementation finds Solution 1, which is feasible. Meanwhile, the other two solutions satisfy the precedence constraint but not the workstation time content.

In the third instance, with 11 tasks and 5 workstations, as shown in Table 1, the implementation does not find a feasible solution; thus, the solution in this table is computed solving the SALB-2 problem by running the implementation with a specific number of workstations until the result is similar to $C = 60$.

The values of the Lagrange multipliers to run the Listing 1.2 according to the model in Eq. 5 are: for instance in Fig. 1a $\lambda_1 = 4000$, $\lambda_2 = 5700$, and $\lambda_3 = 4000$; for instance in Fig. 1b $\lambda_1 = 109$, $\lambda_2 = 200$, and $\lambda_3 = 4000$; and for instance in

Fig. 1c $\lambda_1 = 1000$, $\lambda_2 = 2000$, and $\lambda_3 = 10000$. These values were not optimized, but rather are the result of running multiple experiments.

Table 1: Solution of SALB-1 problem for the three instances Fig. 1

Results for instance Fig. 1a							
Solution	Cycle time ± (C)	Workstations ‡ (k)	1	2	3		
1	7	Tasks (i)	1,3	2	4,5		
		Efficiency (E_k)	0.71	0.71	0.86		
2†	~~7~~ 8	Tasks (i)	1,2	3,4	5		
		Efficiency (E_k)	1.14	0.86	0.29		
3†	~~7~~ 9	Tasks (i)	1,3	2,4	5		
		Efficiency (E_k)	0.71	1.29	0.29		
Instance in Fig. 1b							
Solution	Cycle time ± (C)	Workstations ± (k) (k)	1	2	3		
1	73	Tasks (i)	1,2,4	3,5	6,7		
		Efficiency (E_k)	1.00	0.93	0.81		
2†	~~73~~ 80	Tasks (i)	1,3	2,5	4,6,7		
		Efficiency (E_k)	0.86	0.84	1.04		
3†	~~73~~ 97	Tasks (i)	1,2,3	5	4,6,7		
		Efficiency (E_k)	1.33	0.37	1.04		
Instance in Fig.1c							
Solution	Cycle time ± (C)	Workstations ± (k) (k)	1	2	3	4	5
1∗	~~60~~ 55	Tasks (i)	1	2,3,7	4	5,6,8	9,10,11
		Efficiency (E_k)	0.83	0.78	0.92	0.90	0.73

† Infeasible solution.
∗ The implementation in Listing 1.2 does not return a feasible solution.
∗ This solution is computed by solving the SALB-2 problem.
± Input parameter.
‡ Decision variable to optimise.

In the case of the SALB-2 problem, the results are shown in Table 2. The implementation in Listing 1.3 requires the number of workstations to be open, and the objective is to minimise the cycle time. Three solutions with the same number of workstations $K = 3$ are displayed but with different cycle times. For the instance in Fig. 1a, the minimum cycle time is 6, the same solution as the one in Table 1. In this case, the implementation finds a feasible solution for all the instances. In the two cases, the solution was computed almost instantaneously.

Table 2: Solution of SALB-2 problem for the three instances Fig. 1

Results for instance Fig. 1a							
Solution	Cycle time ‡ (C)	Workstations ± (k)	1	2	3		
1	6	Tasks (i)	1,3	2	4,5		
		Efficiency (E_k)	0.83	0.83	1.00		
2	7	Tasks (i)	1	2,3	4,5		
		Efficiency (E_k)	0.43	1.00	0.86		
3	8	Tasks (i)	1,2	3	4,5		
		Efficiency (E_k)	1.00	0.25	0.75		
Instance in Fig. 1b							
Solution	Cycle time ‡ (C)	Workstations ± (k)	1	2	3		
1	73	Tasks (i)	1,2,4	3,5	6,7		
		Efficiency (E_k)	1.00	0.93	0.81		
2	78	Tasks (i)	1,3	2,4,5	6,7		
		Efficiency (E_k)	0.81	1.00	0.76		
3	86	Tasks (i)	1,4	2,3	5,6,7		
		Efficiency (E_k)	0.45	0.87	1.00		
Instance in Fig.1c							
Solution	Cycle time ‡ (C)	Workstations ± (k)	1	2	3	4	5
1	55	Tasks (i)	4	1	2,3,5	6,7,9	8,10,11
		Efficiency (E_k)	1.00	0.91	0.91	0.93	0.8
2	55	Tasks (i)	4	1	5,9,8	2,3,6	7,10,11
		Efficiency (E_k)	1.00	0.91	0.98	0.85	0.8
3	67	Tasks (i)	1	4	2,3,5,7	6,8,9	10,11
		Efficiency (E_k)	0.75	0.82	1.00	0.76	0.4

± Input parameter.
‡ Decision variable to optimise.

4 Discussion

In this study, we explored the application of quantum annealing to the line-balancing problem, a well-known NP-hard problem in production and manufacturing systems. Using the D-Wave quantum annealer and its associated tools, we implemented two variations of the problem, SALB-1 and SALB-2. We solved three small instances of the problem to assess the feasibility and effectiveness of this approach.

The results from the quantum annealing implementation were promising, with the heuristic successfully finding the optimum solution for all three instances tested. This is particularly noteworthy given the complexity of the line

balancing problem and the fact that quantum annealing is a relatively new method in the context of combinatorial optimisation problems.

The quantum annealing heuristic utilised in this study, through the D-Wave system, effectively navigated the solution space of the problem, finding the minimal cycle time and the optimal allocation of tasks to workstations. In all three cases, the results were obtained in a reasonable time frame, demonstrating the potential of quantum annealing to solve line-balancing problems efficiently. Despite the small scale of the instances tested, these results suggest that quantum annealing could be a promising tool for tackling more complex, real-world line-balancing problems.

However, several aspects of this approach require further investigation. For example, although the heuristic found the optimal solution in these small cases, the performance of quantum annealing in larger, more complex instances remains uncertain. Additionally, the sensitivity of the quantum annealing process to parameter settings, such as the Lagrange multipliers used in this study, suggests that further tuning may be needed to enhance performance in more challenging cases.

Future research should focus on testing quantum annealing with larger instances of the line-balancing problem, exploring hybrid solutions that combine classical and quantum approaches, and investigating the impact of various tuning parameters on solution quality. Furthermore, comparisons with other state-of-the-art optimisation techniques, such as exact algorithms and metaheuristics, would provide a broader perspective on the potential of quantum annealing for solving line-balancing and related optimisation problems.

Overall, this study provides an encouraging initial exploration of quantum annealing for line-balancing problems, with promising results for small instances and a clear path for future work to expand and refine these findings.

5 Conclusions

This study demonstrates the potential of quantum annealing for solving the line-balancing problem, a key challenge in modern manufacturing. We could obtain the optimal solution in each case by applying quantum annealing to small instances of the SALB-1 and SALB-2 problems. These results suggest that quantum annealing can effectively navigate complex optimisation landscapes and offer a promising approach for line-balancing tasks in industrial settings. While the current study is limited to small instances, the findings provide a foundation for future research to explore the scalability of quantum annealing to larger, more complex problems. With continued advancements in quantum computing technology, quantum annealing may become a valuable tool for optimising production processes and enhancing efficiency in Industry 4.0.

References

1. J. Morgan, M. Halton, Y. Qiao, and J. G. Breslin, "Industry 4.0 smart reconfigurable manufacturing machines," *Journal of Manufacturing Systems*, vol. 59,

pp. 481–506, 2021.
2. S. V. Ravelo, "Approximation algorithms for simple assembly line balancing problems," *Journal of Combinatorial Optimization*, vol. 43, pp. 432–443, Mar. 2022.
3. A. Rajak, S. Suzuki, A. Dutta, and B. K. Chakrabarti, "Quantum annealing: An overview," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 381, p. 20210417, Dec. 2022.
4. S. Yarkoni, E. Raponi, T. Bäck, and S. Schmitt, "Quantum annealing for industry applications: Introduction and review," *Reports on Progress in Physics*, vol. 85, p. 104001, Sept. 2022.
5. S. J. Weinberg, F. Sanches, T. Ide, K. Kamiya, and R. Correll, "Supply chain logistics with quantum and classical annealing algorithms," *Scientific Reports*, vol. 13, p. 4770, Mar. 2023.
6. P. Schworm, X. Wu, M. Glatt, and J. C. Aurich, "Solving flexible job shop scheduling problems in manufacturing with Quantum Annealing," *Production Engineering*, vol. 17, pp. 105–115, Feb. 2023.
7. E. Grant, T. S. Humble, and B. Stump, "Benchmarking Quantum Annealing Controls with Portfolio Optimization," *Physical Review Applied*, vol. 15, p. 014012, Jan. 2021.
8. D. Herman, C. Googin, X. Liu, Y. Sun, A. Galda, I. Safro, M. Pistoia, and Y. Alexeev, "Quantum computing for finance," *Nature Reviews Physics*, vol. 5, pp. 450–465, Aug. 2023.
9. M. Motta and J. E. Rice, "Emerging quantum computing algorithms for quantum chemistry," *WIREs Computational Molecular Science*, vol. 12, no. 3, p. e1580, 2022.
10. S. B. Ramezani, A. Sommers, H. K. Manchukonda, S. Rahimi, and A. Amirlatifi, "Machine Learning Algorithms in Quantum Computing: A Survey," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2020.
11. N. Boysen, P. Schulze, and A. Scholl, "Assembly line balancing: What happened in the last fifteen years?," *European Journal of Operational Research*, vol. 301, pp. 797–814, Sept. 2022.
12. F. Glover, G. Kochenberger, R. Hennig, and Y. Du, "Quantum bridge analytics I: A tutorial on formulating and using QUBO models," *Annals of Operations Research*, vol. 314, pp. 141–183, July 2022.
13. L. P. Yulianti and K. Surendro, "Implementation of Quantum Annealing: A Systematic Review," *IEEE Access*, vol. 10, pp. 73156–73177, 2022.
14. P. Sivasankaran and P. Shahabudeen, "Literature review of assembly line balancing problems," *The International Journal of Advanced Manufacturing Technology*, vol. 73, pp. 1665–1694, Aug. 2014.
15. L. J. Watters, "Letter to the Editor—Reduction of Integer Polynomial Programming Problems to Zero-One Linear Programming Problems," *Operations Research*, vol. 15, pp. 1171–1174, Dec. 1967.
16. D-Wave, "Problem Formulation Guide," whitepaper, D-Wave. The Quantum Computing Company, 2022.